

SunOS Reference Manual

Sun Microsystems, Inc.
2550 Garcia Avenue
Mountain View, CA 94043
U.S.A.



© 1994 Sun Microsystems, Inc. All rights reserved.
2550 Garcia Avenue, Mountain View, California 94043-1100 U.S.A.

This product and related documentation are protected by copyright and distributed under licenses restricting its use, copying, distribution, and decompilation. No part of this product or related documentation may be reproduced in any form by any means without prior written authorization of Sun and its licensors, if any.

Portions of this product may be derived from the UNIX® and Berkeley 4.3 BSD systems, licensed from UNIX Systems Laboratories, Inc., a wholly owned subsidiary of Novell, Inc., and the University of California, respectively. Third-party software, including font technology, in this product is protected by copyright and licensed from Sun's Suppliers.

RESTRICTED RIGHTS LEGEND: Use, duplication, or disclosure by the government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 and FAR 52.227-19.

This product or the products described herein may be protected by one or more U.S., foreign patents, or pending applications.

TRADEMARKS

Sun, Sun Microsystems, the Sun Logo, SunSoft, Sun Microsystems Computer Corporation and Solaris, are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and certain other countries. UNIX is a registered trademark of Novell, Inc., in the United States and other countries; X/Open Company, Ltd., is the exclusive licensor of such trademark. OPEN LOOK® is a registered trademark of Novell, Inc. All other product names mentioned herein are the trademarks of their respective owners.

All SPARC trademarks, including the SCD Compliant Logo, are trademarks or registered trademarks of SPARC International, Inc. SPARCstation, SPARCserver, SPARCengine, SPARCstorage, SPARCware, SPARCcenter, SPARCclassic, SPARCcluster, SPARCdesign, SPARC811, SPARCprinter, UltraSPARC, microSPARC, SPARCworks, and SPARCcompiler are licensed exclusively to Sun Microsystems, Inc. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.

The OPEN LOOK® and Sun™ Graphical User Interfaces were 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.

X Window System is a product of the Massachusetts Institute of Technology.

THIS PUBLICATION IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

THIS PUBLICATION COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THE PUBLICATION. SUN MICROSYSTEMS, INC. MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THIS PUBLICATION AT ANY TIME.

Portions © AT&T 1983-1990 and reproduced with permission from AT&T.

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, etc.
 - 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. Literal characters (commands and options) are in **bold** font and variables (arguments, parameters and substitution characters) are in *italic* font. 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 ...'*.
- | Separator. Only one of the arguments separated by this character can be specified at time.

PROTOCOL

This section occurs only in subsection 3R to indicate the protocol description file. The protocol specification pathname is always listed in **bold** font.

AVAILABILITY

This section briefly states any limitations on the availability of the command. These limitations could be hardware or software specific.

A specification of a class of hardware platform, such as **x86** or **SPARC**, denotes that the command or interface is applicable for the hardware platform specified.

In Section 1 and Section 1M, **AVAILABILITY** indicates which package contains the command being described on the manual page. In order to use the command, the specified package must have been installed with the operating system. If the package was not installed, see **pkgadd(1)** for information on how to upgrade.

MT-LEVEL

This section lists the **MT-LEVEL** of the library functions described in the Section 3 manual pages. The **MT-LEVEL** defines the libraries' ability to support threads. See **Intro(3)** for more information.

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**.

IOCTLS

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. IOCTLS for a specific device are listed alphabetically (on the man page for that specific device). IOCTLS are used for a particular class of devices all which have an **io** ending, such as **mtio(7)**.

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.

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 as **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 super-user,

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

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

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.

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. Messages appear in **bold** font with the exception of variables, which are in *italic* font.

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.

NAME	Intro, 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 #include files indicated on each page. Certain major collections are identified by a letter after the section number:</p> <p>(3B) These functions constitute the Source Compatibility (with BSD functions) library. It is implemented as a shared object, libucb.so, and as an archive, libucb.a, but is not automatically linked by the C compilation system. Specify -lucb on the cc command line to link with this library, which is located in the /usr/ucb subdirectory. Header files for this library are located within /usr/ucbinclude.</p> <p>(3C) These functions, together with those of Section 2 and those marked (3S), constitute 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. C programs are linked with the shared object version of the standard C library by default. Specify -dn on the cc command line to link with the archive version. (See cc(1B) for other overrides, and the “C Compilation System” chapter of the <i>ANSI C Programmer’s Guide</i> for a discussion.)</p> <p>(3E) These functions constitute the ELF access library, libelf, (Extensible Linking Formats). This library provides the interface for the creation and analyses of “elf” files; executables, objects, and shared objects. libelf is implemented as a shared object, libelf.so, and as an archive, libelf.a, but is not automatically linked by the C compilation system. Specify -lelf on the cc command line to link with this library.</p> <p>(3G) These functions constitute the string pattern-matching & pathname manipulation library, libgen. This library is implemented as an archive, libgen.a, but not as a shared object, and is not automatically linked by the C compilation system. Specify -lgen on the cc command line to link with this library.</p> <p>(3I) These functions constitute the wide character libraries for multi-byte character support, and the international library for messaging. These libraries, libintl, and libw, are both implemented as shared objects, libintl.so and libw.so, and as archives, libintl.a and libw.a. However, they are not automatically linked by the C compilation system; specify -lintl or -lw on the cc command line, as needed to link the appropriate library.</p> <p>(3K) These functions allow access to the kernel’s virtual memory library, which is implemented as a shared object, libkvm.so, and as an archive, libkvm.a, but is not automatically linked by the C compilation system. Specify -lkvm on the cc command line to link with this library.</p> <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.</p>

- (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.
- (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.
- (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**.
- (3T) These functions constitute the threads library, **libthread**. This library is used for building multithreaded applications. **libthread** is implemented as a shared object, **libthread.so**, but not as an archived library. **libthread** is not automatically linked by the C compilation system. Specify **-lthread** on the **cc** command line to link with this library.
- (3X) Specialized libraries. The files in which these libraries are found are given on the appropriate pages.

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

Libraries are classified into four categories which define the level of the libraries' ability to support threads.

The MT-Level category of the libraries in this section are shown on each man page under **MT-Level**. Pages containing routines that are of multiple or differing MT-Levels show this under the **NOTES** section.

Safe Safe is simply an attribute of code that can be called from a multithreaded application. It is a generic term used to differentiate between code that is unsafe.

- Unsafe** An unsafe library contains global and static data that is not protected. It is not safe to use unless the application arranges for only one thread at time to execute within the library. Unsafe libraries may contain routines that are safe; however, most of the library's routines are unsafe to call.
- MT-Safe** An MT-Safe library is fully prepared for multithreaded access. It protects its global and static data with locks, and can provide a reasonable amount of concurrency. Note that a library can be safe to use, but not MT-Safe. For example, surrounding an entire library with a monitor makes the library safe, but it supports no concurrency so it is not considered MT-Safe. An MT-Safe library must permit a reasonable amount of concurrency. (This definition's purpose is to give precision to what is meant when a library is described as safe. The definition of a "safe" library does not specify if the library supports concurrency. The MT-Safe definition makes it clear that the library is safe, and supports some concurrency. This clarifies the safe definition, which can mean anything from being single threaded to being any degree of multithreaded.)
- Async-Safe** Async-Safe refers to particular library routines that can be safely called from a signal handler. A thread that is executing an Async-Safe routine will not deadlock with itself if interrupted by a signal. Signals are only a problem for MT-Safe routines that acquire locks.
- Signals are disabled when locks are acquired in Async-Safe routines. This prevents a signal handler that might acquire the same lock from being called.

MT-Safe with exceptions

See the **NOTES** sections of these pages for a description of the exceptions.

Safe with exceptions

See the **NOTES** sections of these pages for a description of the exceptions.

The following table contains reentrant counterparts for unsafe functions. This table is subject to change by SunSoft.

Reentrant functions for libc**Unsafe Function Reentrant counterpart**

ctime	ctime_r
localtime	localtime_r
asctime	asctime_r
gmtime	gmtime_r
ctermid	ctermid_r
getlogin	getlogin_r
rand	rand_r
readdir	readdir_r
strtok	strtok_r
tmpnam	tmpnam_r

FILES	<p><i>INCDIR</i> usually /usr/include</p> <p><i>LIBDIR</i> usually /usr/ccs/lib</p> <p><i>LIBDIR/libc.so</i></p> <p><i>LIBDIR/libc.a</i></p> <p><i>LIBDIR/libgen.a</i></p> <p><i>LIBDIR/libm.a</i></p> <p><i>LIBDIR/libsfm.sa</i></p> <p>/usr/lib/libc.so.1</p>
SEE ALSO	<p>ar(1), cc(1B), ld(1), nm(1), intro(2), stdio(3S)</p> <p><i>ANSI C Programmer's Guide</i></p>
DIAGNOSTICS	<p>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 instead of ±HUGE. (HUGE_VAL and HUGE are defined in math.h to be infinity and the largest-magnitude single-precision number, respectively.)</p>
NOTES ON MULTITHREAD APPLICATIONS	<p>When compiling a multithreaded application, the _REENTRANT flag must be defined on the compile line (-D_REENTRANT). This enables new definitions for functions only applicable to multithreaded applications.</p> <p>When building a singlethreaded application, the _REENTRANT flag should be undefined. This generates a binary that is executable on previous Solaris releases, which do not support multithreading.</p> <p>When linking, it is a requirement that a multithreaded application be constructed to ensure that libthread physically interposes upon the C library. For example, the command line for linking the application using ld should be ordered as follows:</p> <p style="text-align: center;">example% ld [options] .o's ... -lthread</p> <p>Note that the behaviour of the C library is undefined if -lc precedes -lthread. And, when linking with cc, -lthread must be last on the command line.</p> <p>Unsafe interfaces should be called only from the main thread to ensure the application's safety.</p> <p>MT-Safe interfaces are denoted in the NOTES section of the functions and libraries man pages. If a man page does not state explicitly that an interface is MT-Safe, the user should assume that the interface is unsafe.</p>
REALTIME APPLICATIONS	<p>Be sure to have set the environment variable LD_BIND_NOW to a non-NULL value to enable early binding.</p>
NOTES	<p>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.</p>

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 **-I** option to **lint**. (For example, **-lm** includes definitions for **libm**.) Use of **lint** is highly recommended. See the **lint** chapter in *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}**.

Name	Appears on Page	Description
__nis_map_group	nis_groups (3N)	NIS+ group manipulation functions
_longjmp	setjmp (3B)	non-local goto
_setjmp	setjmp (3B)	non-local goto
_tolower	conv (3C)	translate characters
_toupper	conv (3C)	translate characters
a64l	a64l (3C)	convert between long integer and base-64 ASCII string
abort	abort (3C)	terminate the process abnormally
abs	abs (3C)	return absolute value of integer
accept	accept (3N)	accept a connection on a socket
acos	trig (3M)	trigonometric functions

acosh	hyperbolic(3M)	hyperbolic functions
addch	 curs_addch(3X)	add a character (with attributes) to a curses window and advance cursor
addchnstr	 curs_addchnstr(3X)	add string of characters (and attributes) to a curses window
addchstr	 curs_addchstr(3X)	add string of characters (and attributes) to a curses window
addnstr	 curs_addnstr(3X)	add a string of characters to a curses window and advance cursor
addnwstr	 curs_addnwstr(3X)	add a string of wchar_t characters to a curses window and advance cursor
addsev	 addsev(3C)	define additional severities
addseverity	 addseverity(3C)	build a list of severity levels for an application for use with fmtmsg
addstr	 curs_addstr(3X)	add a string of characters to a curses window and advance cursor
addwch	 curs_addwch(3X)	add a wchar_t character (with attributes) to a curses window and advance cursor
addwchnstr	 curs_addwchnstr(3X)	add string of wchar_t characters (and attributes) to a curses window
addwchstr	 curs_addwchstr(3X)	add string of wchar_t characters (and attributes) to a curses window
addwstr	 curs_addwstr(3X)	add a string of wchar_t characters to a curses window and advance cursor
adjcurspos	 curs_alecompat(3X)	these functions are added to ALE curses library for moving the cursor by character.
advance	 regexpr(3G)	regular expression compile and match routines
aio_cancel	 aio_cancel(3R)	cancel asynchronous I/O request

aio_error	aio_return(3R)	retrieve return or error status of asynchronous I/O operation
aio_fsync	aio_fsync(3R)	asynchronous file synchronization
aio_read	aio_read(3R)	asynchronous read and write operations
aio_return	aio_return(3R)	retrieve return or error status of asynchronous I/O operation
aio_suspend	aio_suspend(3R)	wait for asynchronous I/O request
aio_write	aio_read(3R)	asynchronous read and write operations
aiocancel	aiocancel(3)	cancel an asynchronous operation
aioread	aioread(3)	asynchronous I/O operations
aiowait	aiowait(3)	wait for completion of asynchronous I/O operation
aiowrite	aioread(3)	asynchronous I/O operations
alloca	malloc(3C)	memory allocator
alphasort	scandir(3B)	scan a directory
asctime	strftime(3C)	convert date and time to string
asctime	ctime(3C)	convert date and time to string
asctime_r	ctime(3C)	convert date and time to string
asin	trig(3M)	trigonometric functions
asinh	hyperbolic(3M)	hyperbolic functions
assert	assert(3X)	verify program assertion
asystemem	systemem(3)	return physical memory information
atan2	trig(3M)	trigonometric functions
atan	trig(3M)	trigonometric functions
atanh	hyperbolic(3M)	hyperbolic functions
atexit	atexit(3C)	add program termination routine
atof	strtod(3C)	convert string to double-precision number
atoi	strtol(3C)	conversion routines
atol	strtol(3C)	conversion routines
atoll	strtol(3C)	conversion routines

attroff	curs_attr(3X)	curses character and window attribute control routines
attron	curs_attr(3X)	curses character and window attribute control routines
attrset	curs_attr(3X)	curses character and window attribute control routines
au_close	au_open(3)	construct and write audit records
au_open	au_open(3)	construct and write audit records
au_preselect	au_preselect(3)	preselect an audit event
au_to	au_to(3)	create audit record tokens
au_to_arg	au_to(3)	create audit record tokens
au_to_attr	au_to(3)	create audit record tokens
au_to_data	au_to(3)	create audit record tokens
au_to_groups	au_to(3)	create audit record tokens
au_to_in_addr	au_to(3)	create audit record tokens
au_to_ipc	au_to(3)	create audit record tokens
au_to_ipc_perm	au_to(3)	create audit record tokens
au_to_iport	au_to(3)	create audit record tokens
au_to_me	au_to(3)	create audit record tokens
au_to_opaque	au_to(3)	create audit record tokens
au_to_path	au_to(3)	create audit record tokens
au_to_process	au_to(3)	create audit record tokens
au_to_return	au_to(3)	create audit record tokens
au_to_socket	au_to(3)	create audit record tokens
au_to_text	au_to(3)	create audit record tokens
au_user_mask	au_user_mask(3)	get user's binary preselection mask
au_write	au_open(3)	construct and write audit records
auth_destroy	rpc_clnt_auth(3N)	library routines for client side remote procedure call authentication
authdes_create	rpc_soc(3N)	obsolete library routines for RPC
authdes_getucred	secure_rpc(3N)	library routines for secure remote procedure calls
authdes_seccreate	secure_rpc(3N)	library routines for secure remote procedure calls
authkerb_getucred	kerberos_rpc(3N)	library routines for remote procedure calls using Kerberos authentication

authkerb_seccreate	kerberos_rpc(3N)	library routines for remote procedure calls using Kerberos authentication
authnone_create	rpc_clnt_auth(3N)	library routines for client side remote procedure call authentication
authsys_create	rpc_clnt_auth(3N)	library routines for client side remote procedure call authentication
authsys_create_default	rpc_clnt_auth(3N)	library routines for client side remote procedure call authentication
authunix_create	rpc_soc(3N)	obsolete library routines for RPC
authunix_create_default	rpc_soc(3N)	obsolete library routines for RPC
basename	basename(3G)	return the last element of a path name
baudrate	curs_termattrs(3X)	curses environment query routines
bcmp	bstring(3B)	bit and byte string operations
bcopy	bstring(3B)	bit and byte string operations
beep	curs_beep(3X)	curses bell and screen flash routines
bessel	bessel(3M)	Bessel functions
bgets	bgets(3G)	read stream up to next delimiter
bind	bind(3N)	bind a name to a socket
bindtextdomain	gettext(3I)	message handling functions
bkgd	curs_bkgd(3X)	curses window background manipulation routines
bkgdset	curs_bkgd(3X)	curses window background manipulation routines
border	curs_border(3X)	create curses borders, horizontal and vertical lines
bottom_panel	panel_top(3X)	panels deck manipulation routines
box	curs_border(3X)	create curses borders, horizontal and vertical lines
bsdmalloc	bsdmalloc(3X)	memory allocator
bsearch	bsearch(3C)	binary search a sorted table

bstring	bstring(3B)	bit and byte string operations
bufsplit	bufsplit(3G)	split buffer into fields
byteorder	byteorder(3N)	convert values between host and network byte order
bzero	bstring(3B)	bit and byte string operations
calloc	malloc(3C)	memory allocator
calloc	malloc(3X)	memory allocator
calloc	mapmalloc(3X)	memory allocator
callrpc	rpc_soc(3N)	obsolete library routines for RPC
can_change_color	 curs_color(3X)	curses color manipulation routines
catclose	catopen(3C)	open/close a message catalog
catgets	catgets(3C)	read a program message
catopen	catopen(3C)	open/close a message catalog
cbreak	 curs_inopts(3X)	curses terminal input option control routines
cbrt	sqrt(3M)	square root, cube root
ceil	floor(3M)	round to integral value in floating-point format
cfgetispeed	termios(3)	general terminal interface
cfgetospeed	termios(3)	general terminal interface
cfree	mapmalloc(3X)	memory allocator
cfsetispeed	termios(3)	general terminal interface
cfsetospeed	termios(3)	general terminal interface
cftime	strftime(3C)	convert date and time to string
clear	 curs_clear(3X)	clear all or part of a curses window
clearerr	ferror(3S)	stream status inquiries
clearok	 curs_outopts(3X)	curses terminal output option control routines
clnt_broadcast	rpc_soc(3N)	obsolete library routines for RPC
clnt_call	rpc_clnt_calls(3N)	library routines for client side calls
clnt_control	rpc_clnt_create(3N)	library routines for dealing with creation and manipulation of CLIENT handles

clnt_create	rpc_clnt_create(3N)	library routines for dealing with creation and manipulation of CLIENT handles
clnt_create_vers	rpc_clnt_create(3N)	library routines for dealing with creation and manipulation of CLIENT handles
clnt_destroy	rpc_clnt_create(3N)	library routines for dealing with creation and manipulation of CLIENT handles
clnt_dg_create	rpc_clnt_create(3N)	library routines for dealing with creation and manipulation of CLIENT handles
clnt_freeres	rpc_clnt_calls(3N)	library routines for client side calls
clnt_geterr	rpc_clnt_calls(3N)	library routines for client side calls
clnt_pcreateerror	rpc_clnt_create(3N)	library routines for dealing with creation and manipulation of CLIENT handles
clnt_perrno	rpc_clnt_calls(3N)	library routines for client side calls
clnt_perror	rpc_clnt_calls(3N)	library routines for client side calls
clnt_raw_create	rpc_clnt_create(3N)	library routines for dealing with creation and manipulation of CLIENT handles
clnt_screateerror	rpc_clnt_create(3N)	library routines for dealing with creation and manipulation of CLIENT handles
clnt_sperrno	rpc_clnt_calls(3N)	library routines for client side calls
clnt_sperror	rpc_clnt_calls(3N)	library routines for client side calls
clnt_tli_create	rpc_clnt_create(3N)	library routines for dealing with creation and manipulation of CLIENT handles

clnt_tp_create	rpc_clnt_create(3N)	library routines for dealing with creation and manipulation of CLIENT handles
clnt_vc_create	rpc_clnt_create(3N)	library routines for dealing with creation and manipulation of CLIENT handles
clntraw_create	rpc_soc(3N)	obsolete library routines for RPC
clnttcp_create	rpc_soc(3N)	obsolete library routines for RPC
clntudp_bufcreate	rpc_soc(3N)	obsolete library routines for RPC
clntudp_create	rpc_soc(3N)	obsolete library routines for RPC
clock	clock(3C)	report CPU time used
clock_getres	clock_gettime(3R)	high-resolution clock operations
clock_gettime	clock_gettime(3R)	high-resolution clock operations
clock_settime	clock_settime(3R)	high-resolution clock operations
closedir	directory(3C)	directory operations
closelog	syslog(3)	control system log
clrtoeol	clear(3X)	clear all or part of a curses window
clrtoeol	clear(3X)	clear all or part of a curses window
color_content	color(3X)	curses color manipulation routines
compile	regex(3G)	regular expression compile and match routines
cond_broadcast	condition(3T)	condition variables
cond_destroy	condition(3T)	condition variables
cond_init	condition(3T)	condition variables
cond_signal	condition(3T)	condition variables
cond_timedwait	condition(3T)	condition variables
cond_wait	condition(3T)	condition variables
condition	condition(3T)	condition variables
connect	connect(3N)	initiate a connection on a socket
conv	conv(3C)	translate characters
copylist	copylist(3G)	copy a file into memory

copysign	ieee_functions(3M)	appendix and related miscellaneous functions for IEEE arithmetic
copywin	curs_overlay(3X)	overlap and manipulate overlapped curses windows
cos	trig(3M)	trigonometric functions
cosh	hyperbolic(3M)	hyperbolic functions
crypt	crypt(3C)	generate encryption
cset	cset(3I)	get information on EUC codesets
csetcol	cset(3I)	get information on EUC codesets
csetlen	cset(3I)	get information on EUC codesets
csetno	cset(3I)	get information on EUC codesets
ctermid	ctermid(3S)	generate path name for controlling terminal
ctermid_r	ctermid(3S)	generate path name for controlling terminal
ctime	ctime(3C)	convert date and time to string
ctime_r	ctime(3C)	convert date and time to string
ctype	ctype(3C)	character handling
current_field	form_page(3X)	set forms current page and field
current_item	menu_item_current(3X)	set and get current menus items
curs_addch	curs_addch(3X)	add a character (with attributes) to a curses window and advance cursor
curs_addchstr	curs_addchstr(3X)	add string of characters (and attributes) to a curses window
curs_addstr	curs_addstr(3X)	add a string of characters to a curses window and advance cursor
curs_addwch	curs_addwch(3X)	add a wchar_t character (with attributes) to a curses window and advance cursor

curs_addwchstr	curs_addwchstr(3X)	add string of wchar_t characters (and attributes) to a curses window
curs_addwstr	curs_addwstr(3X)	add a string of wchar_t characters to a curses window and advance cursor
curs_alecompat	curs_alecompat(3X)	these functions are added to ALE curses library for moving the cursor by character.
curs_attr	curs_attr(3X)	curses character and window attribute control routines
curs_beep	curs_beep(3X)	curses bell and screen flash routines
curs_bkgd	curs_bkgd(3X)	curses window background manipulation routines
curs_border	curs_border(3X)	create curses borders, horizontal and vertical lines
curs_clear	curs_clear(3X)	clear all or part of a curses window
curs_color	curs_color(3X)	curses color manipulation routines
curs_delch	curs_delch(3X)	delete character under cursor in a curses window
curs_deleteln	curs_deleteln(3X)	delete and insert lines in a curses window
curs_getch	curs_getch(3X)	get (or push back) characters from curses terminal keyboard
curs_getstr	curs_getstr(3X)	get character strings from curses terminal keyboard
curs_getwch	curs_getwch(3X)	get (or push back) wchar_t characters from curses terminal keyboard
curs_getwstr	curs_getwstr(3X)	get wchar_t character strings from curses terminal keyboard
curs_getyx	curs_getyx(3X)	get curses cursor and window coordinates
curs_inch	curs_inch(3X)	get a character and its attributes from a curses window

 curs_inchstr 	 curs_inchstr(3X) 	get a string of characters (and attributes) from a curses window
 curs_initscr 	 curs_initscr(3X) 	curses screen initialization and manipulation routines
 curs_inopts 	 curs_inopts(3X) 	curses terminal input option control routines
 curs_insch 	 curs_insch(3X) 	insert a character before the character under the cursor in a curses window
 curs_insstr 	 curs_insstr(3X) 	insert string before character under the cursor in a curses window
 curs_instr 	 curs_instr(3X) 	get a string of characters from a curses window
 curs_inswch 	 curs_inswch(3X) 	insert a wchar_t character before the character under the cursor in a curses window
 curs_inswstr 	 curs_inswstr(3X) 	insert wchar_t string before character under the cursor in a curses window
 curs_inwch 	 curs_inwch(3X) 	get a wchar_t character and its attributes from a curses window
 curs_inwchstr 	 curs_inwchstr(3X) 	get a string of wchar_t characters (and attributes) from a curses window
 curs_inwstr 	 curs_inwstr(3X) 	get a string of wchar_t characters from a curses window
 curs_kernel 	 curs_kernel(3X) 	low-level curses routines
 curs_move 	 curs_move(3X) 	move curses window cursor
 curs_outopts 	 curs_outopts(3X) 	curses terminal output option control routines
 curs_overlay 	 curs_overlay(3X) 	overlap and manipulate overlapped curses windows
 curs_pad 	 curs_pad(3X) 	create and display curses pads
 curs_printw 	 curs_printw(3X) 	print formatted output in curses windows
 curs_refresh 	 curs_refresh(3X) 	refresh curses windows and lines

 curs_scanw 	 curs_scanw(3X) 	convert formatted input from a curses widow
 curs_scr_dump 	 curs_scr_dump(3X) 	read (write) a curses screen from (to) a file
 curs_scroll 	 curs_scroll(3X) 	scroll a curses window
 curs_set 	 curs_kernel(3X) 	low-level curses routines
 curs_slk 	 curs_slk(3X) 	curses soft label routines
 curs_termattrs 	 curs_termattrs(3X) 	curses environment query routines
 curs_termcap 	 curs_termcap(3X) 	curses interfaces (emulated) to the termcap library
 curs_terminfo 	 curs_terminfo(3X) 	curses interfaces to terminfo database
 curs_touch 	 curs_touch(3X) 	curses refresh control routines
 curs_util 	 curs_util(3X) 	curses miscellaneous utility routines
 curs_window 	 curs_window(3X) 	create curses windows
 curses 	 curses(3X) 	CRT screen handling and optimization package
 cuserid 	 cuserid(3S) 	get character login name of the user
 data_ahead 	 form_data(3X) 	tell if forms field has off-screen data ahead or behind
 data_behind 	 form_data(3X) 	tell if forms field has off-screen data ahead or behind
 db_add_entry 	 nis_db(3N) 	NIS+ Database access functions
 db_checkpoint 	 nis_db(3N) 	NIS+ Database access functions
 db_create_table 	 nis_db(3N) 	NIS+ Database access functions
 db_destroy_table 	 nis_db(3N) 	NIS+ Database access functions
 db_first_entry 	 nis_db(3N) 	NIS+ Database access functions
 db_free_result 	 nis_db(3N) 	NIS+ Database access functions
 db_initialize 	 nis_db(3N) 	NIS+ Database access functions
 db_list_entries 	 nis_db(3N) 	NIS+ Database access functions

db_next_entry	nis_db(3N)	NIS+ Database access functions
db_remove_entry	nis_db(3N)	NIS+ Database access functions
db_reset_next_entry	nis_db(3N)	NIS+ Database access functions
db_standby	nis_db(3N)	NIS+ Database access functions
db_table_exists	nis_db(3N)	NIS+ Database access functions
db_unload_table	nis_db(3N)	NIS+ Database access functions
dbm	dbm(3B)	data base subroutines
dbm_clearerr	ndbm(3)	data base subroutines
dbm_close	ndbm(3)	data base subroutines
dbm_delete	ndbm(3)	data base subroutines
dbm_error	ndbm(3)	data base subroutines
dbm_fetch	ndbm(3)	data base subroutines
dbm_firstkey	ndbm(3)	data base subroutines
dbm_nextkey	ndbm(3)	data base subroutines
dbm_open	ndbm(3)	data base subroutines
dbm_store	ndbm(3)	data base subroutines
dbmclose	dbm(3B)	data base subroutines
dbminit	dbm(3B)	data base subroutines
dcgettext	gettext(3I)	message handling functions
decimal_to_double	decimal_to_floating(3)	convert decimal record to floating-point value
decimal_to_extended	decimal_to_floating(3)	convert decimal record to floating-point value
decimal_to_floating	decimal_to_floating(3)	convert decimal record to floating-point value
decimal_to_quadruple	decimal_to_floating(3)	convert decimal record to floating-point value
decimal_to_single	decimal_to_floating(3)	convert decimal record to floating-point value
def_prog_mode	curs_kernel(3X)	low-level curses routines
def_shell_mode	curs_kernel(3X)	low-level curses routines
del_curterm	curs_terminfo(3X)	curses interfaces to terminfo database
del_panel	panel_new(3X)	create and destroy panels
delay_output	curs_util(3X)	curses miscellaneous utility routines
delch	curs_delch(3X)	delete character under cursor in a curses window
delete	dbm(3B)	data base subroutines

deleteln	curs_deleteln(3X)	delete and insert lines in a curses window
delscreen	curs_initscr(3X)	curses screen initialization and manipulation routines
delwin	curs_window(3X)	create curses windows
derwin	curs_window(3X)	create curses windows
dgettext	gettext(3I)	message handling functions
dial	dial(3N)	establish an outgoing terminal line connection
difftime	difftime(3C)	computes the difference between two calendar times
directory	directory(3C)	directory operations
dirname	dirname(3G)	report the parent directory name of a file path name
div	div(3C)	compute the quotient and remainder
dladdr	dladdr(3X)	translate address to symbolic information.
dlclose	dlclose(3X)	close a shared object
dlerror	dlerror(3X)	get diagnostic information
dlopen	dlopen(3X)	open a shared object
dlsym	dlsym(3X)	get the address of a symbol in a shared object
dn_comp	resolver(3N)	resolver routines
dn_expand	resolver(3N)	resolver routines
doconfig	doconfig(3N)	execute a configuration script
double_to_decimal	floating_to_decimal(3)	convert floating-point value to decimal record
doupdate	curs_refresh(3X)	refresh curses windows and lines
drand48	drand48(3C)	generate uniformly distributed pseudo-random numbers
dup2	dup2(3C)	duplicate an open file descriptor
dup_field	form_field_new(3X)	create and destroy forms fields
dupwin	curs_window(3X)	create curses windows
dynamic_field_info	form_field_info(3X)	get forms field characteristics
echo	curs_inopts(3X)	curses terminal input option control routines

echochar	curs_addch(3X)	add a character (with attributes) to a curses window and advance cursor
echowchar	curs_addwch(3X)	add a wchar_t character (with attributes) to a curses window and advance cursor
econvert	econvert(3)	output conversion
ecvt	econvert(3)	output conversion
ecvt	ecvt(3C)	convert floating-point number to string
edata	end(3C)	last locations in program
elf32_fsize	elf32_fsize(3E)	return the size of an object file type
elf32_getehdr	elf32_getehdr(3E)	retrieve class-dependent object file header
elf32_getphdr	elf32_getphdr(3E)	retrieve class-dependent program header table
elf32_getshdr	elf32_getshdr(3E)	retrieve class-dependent section header
elf32_newehdr	elf32_getehdr(3E)	retrieve class-dependent object file header
elf32_newphdr	elf32_getphdr(3E)	retrieve class-dependent program header table
elf32_xlatetof	elf32_xlatetof(3E)	class-dependent data translation
elf32_xlatetom	elf32_xlatetof(3E)	class-dependent data translation
elf	elf(3E)	object file access library
elf_begin	elf_begin(3E)	process ELF object files
elf_cntl	elf_cntl(3E)	control an elf file descriptor
elf_end	elf_begin(3E)	process ELF object files
elf_errmsg	elf_errmsg(3E)	error handling
elf_errno	elf_errmsg(3E)	error handling
elf_fill	elf_fill(3E)	set fill byte
elf_flagdata	elf_flagdata(3E)	manipulate flags
elf_flagehdr	elf_flagdata(3E)	manipulate flags
elf_flagelf	elf_flagdata(3E)	manipulate flags
elf_flagphdr	elf_flagdata(3E)	manipulate flags
elf_flagscn	elf_flagdata(3E)	manipulate flags
elf_flagshdr	elf_flagdata(3E)	manipulate flags
elf_getarhdr	elf_getarhdr(3E)	retrieve archive member header

elf_getarsym	elf_getarsym(3E)	retrieve archive symbol table
elf_getbase	elf_getbase(3E)	get the base offset for an object file
elf_getdata	elf_getdata(3E)	get section data
elf_getident	elf_getident(3E)	retrieve file identification data
elf_getscn	elf_getscn(3E)	get section information
elf_hash	elf_hash(3E)	compute hash value
elf_kind	elf_kind(3E)	determine file type
elf_ndxscn	elf_getscn(3E)	get section information
elf_newdata	elf_getdata(3E)	get section data
elf_newscn	elf_getscn(3E)	get section information
elf_next	elf_begin(3E)	process ELF object files
elf_nextscn	elf_getscn(3E)	get section information
elf_rand	elf_begin(3E)	process ELF object files
elf_rawdata	elf_getdata(3E)	get section data
elf_rawfile	elf_rawfile(3E)	retrieve uninterpreted file contents
elf_strptr	elf_strptr(3E)	make a string pointer
elf_update	elf_update(3E)	update an ELF descriptor
elf_version	elf_version(3E)	coordinate ELF library and application versions
encrypt	crypt(3C)	generate encryption
end	end(3C)	last locations in program
endac	getacinfo(3)	get audit control file information
endauclass	getauclassent(3)	get audit_class entry
endauevent	getauevent(3)	get audit_user entry
endauuser	getauusernam(3)	get audit_user entry
endgrent	getgrnam(3C)	get group entry
endhostent	gethostbyname(3N)	get network host entry
endnetconfig	getnetconfig(3N)	get network configuration database entry
endnetent	getnetbyname(3N)	get network entry
endnetgrent	getnetgrent(3N)	get network group entry
endnetpath	getnetpath(3N)	get /etc/netconfig entry corresponding to NETPATH component
endprotoent	getprotobyname(3N)	get protocol entry
endpwent	getpwnam(3C)	get password entry
endrpcent	getrpcbyname(3N)	get RPC entry
endservent	getservbyname(3N)	get service entry
endspent	getspnam(3C)	get password entry
endutent	getutent(3C)	access utmp file entry

endutxent	getutxent(3C)	access utmpx file entry
endwin	curs_initscr(3X)	curses screen initialization and manipulation routines
erand48	drand48(3C)	generate uniformly distributed pseudo-random numbers
erase	curs_clear(3X)	clear all or part of a curses window
erasechar	curs_termattrs(3X)	curses environment query routines
erf	erf(3M)	error functions
erfc	erfc(3M)	error functions
errno	perror(3C)	print system error messages
etext	end(3C)	last locations in program
ethers	ethers(3N)	Ethernet address mapping operations
euccol	euclen(3I)	get byte length and display width of EUC characters
euclen	euclen(3I)	get byte length and display width of EUC characters
eucscol	euclen(3I)	get byte length and display width of EUC characters
exit	exit(3C)	terminate process
exp	exp(3M)	exponential, logarithm, power
expm1	exp(3M)	exponential, logarithm, power
extended_to_decimal	floating_to_decimal(3)	convert floating-point value to decimal record
fabs	ieee_functions(3M)	appendix and related miscellaneous functions for IEEE arithmetic
fattach	fattach(3C)	attach a STREAMS-based file descriptor to an object in the file system name space
fclose	fclose(3S)	close or flush a stream
fconvert	econvert(3)	output conversion
fcvt	econvert(3)	output conversion
fcvt	ecvt(3C)	convert floating-point number to string
fdatasync	fdatasync(3R)	synchronize a file's data
fdetach	fdetach(3C)	detach a name from a STREAMS-based file descriptor
fdopen	fopen(3S)	open a stream

feof	ferror(3S)	stream status inquiries
ferror	ferror(3S)	stream status inquiries
fetch	dbm(3B)	data base subroutines
fflush	fclose(3S)	close or flush a stream
ffs	ffs(3C)	find first set bit
fgetc	getc(3S)	get character or word from a stream
fgetgrent	getgrnam(3C)	get group entry
fgetgrent_r	getgrnam(3C)	get group entry
fgetpos	fsetpos(3C)	reposition a file pointer in a stream
fgetpwent	getpwnam(3C)	get password entry
fgetpwent_r	getpwnam(3C)	get password entry
fgets	gets(3S)	get a string from a stream
fgetspent	getspnam(3C)	get password entry
fgetspent_r	getspnam(3C)	get password entry
fgetwc	getwc(3I)	convert EUC character from the stream to Process Code
fgetws	getws(3I)	convert a string of EUC characters from the stream to Process Code
field_arg	form_field_validation(3X)	forms field data type validation
field_back	form_field_attributes(3X)	format the general display attributes of forms
field_buffer	form_field_buffer(3X)	set and get forms field attributes
field_count	form_field(3X)	connect fields to forms
field_fore	form_field_attributes(3X)	format the general display attributes of forms
field_index	form_page(3X)	set forms current page and field
field_info	form_field_info(3X)	get forms field characteristics
field_init	form_hook(3X)	assign application-specific routines for invocation by forms
field_just	form_field_just(3X)	format the general appearance of forms
field_opts	form_field_opts(3X)	forms field option routines
field_opts_off	form_field_opts(3X)	forms field option routines
field_opts_on	form_field_opts(3X)	forms field option routines
field_pad	form_field_attributes(3X)	format the general display attributes of forms

field_status	form_field_buffer(3X)	set and get forms field attributes
field_term	form_hook(3X)	assign application-specific routines for invocation by forms
field_type	form_field_validation(3X)	forms field data type validation
field_userptr	form_field_userptr(3X)	associate application data with forms
file_to_decimal	string_to_decimal(3)	parse characters into decimal record
fileno	ferror(3S)	stream status inquiries
filter	curs_util(3X)	curses miscellaneous utility routines
finite	isnan(3C)	determine type of floating-point number
firstkey	dbm(3B)	data base subroutines
flash	curs_beep(3X)	curses bell and screen flash routines
floating_to_decimal	floating_to_decimal(3)	convert floating-point value to decimal record
flock	flock(3B)	apply or remove an advisory lock on an open file
flockfile	flockfile(3S)	acquire and release stream lock
floor	floor(3M)	round to integral value in floating-point format
flushinp	curs_util(3X)	curses miscellaneous utility routines
fmod	ieee_functions(3M)	appendix and related miscellaneous functions for IEEE arithmetic
fmsg	fmsg(3C)	display a message on stderr or system console
fopen	fopen(3B)	open a stream
fopen	fopen(3S)	open a stream
form_cursor	form_cursor(3X)	position forms window cursor
form_data	form_data(3X)	tell if forms field has off-screen data ahead or behind
form_driver	form_driver(3X)	command processor for the forms subsystem
form_field	form_field(3X)	connect fields to forms

form_field_attributes	form_field_attributes(3X)	format the general display attributes of forms
form_field_buffer	form_field_buffer(3X)	set and get forms field attributes
form_field_info	form_field_info(3X)	get forms field characteristics
form_field_just	form_field_just(3X)	format the general appearance of forms
form_field_new	form_field_new(3X)	create and destroy forms fields
form_field_opts	form_field_opts(3X)	forms field option routines
form_field_userptr	form_field_userptr(3X)	associate application data with forms
form_field_validation	form_field_validation(3X)	forms field data type validation
form_fields	form_field(3X)	connect fields to forms
form_fieldtype	form_fieldtype(3X)	forms fieldtype routines
form_hook	form_hook(3X)	assign application-specific routines for invocation by forms
form_init	form_hook(3X)	assign application-specific routines for invocation by forms
form_new	form_new(3X)	create and destroy forms
form_new_page	form_new_page(3X)	forms pagination
form_opts	form_opts(3X)	forms option routines
form_opts_off	form_opts(3X)	forms option routines
form_opts_on	form_opts(3X)	forms option routines
form_page	form_page(3X)	set forms current page and field
form_post	form_post(3X)	write or erase forms from associated subwindows
form_sub	form_win(3X)	forms window and subwindow association routines
form_term	form_hook(3X)	assign application-specific routines for invocation by forms
form_userptr	form_userptr(3X)	associate application data with forms
form_win	form_win(3X)	forms window and subwindow association routines
forms	forms(3X)	character based forms package

fpclass	isnan(3C)	determine type of floating-point number
fpgetmask	fpgetround(3C)	IEEE floating-point environment control
fpgetround	fpgetround(3C)	IEEE floating-point environment control
fpgetsticky	fpgetround(3C)	IEEE floating-point environment control
fprintf	printf(3B)	formatted output conversion
fprintf	printf(3S)	print formatted output
fpsetmask	fpgetround(3C)	IEEE floating-point environment control
fpsetround	fpgetround(3C)	IEEE floating-point environment control
fpsetsticky	fpgetround(3C)	IEEE floating-point environment control
fputc	putc(3S)	put character or word on a stream
fputs	puts(3S)	put a string on a stream
fputwc	putwc(3I)	convert Process Code character to EUC and put on a stream
fputws	putws(3I)	convert a string of Process Code characters to EUC characters and put it on a stream
fread	fread(3S)	buffered binary input/output
free	bsdmalloc(3X)	memory allocator
free	malloc(3C)	memory allocator
free	malloc(3X)	memory allocator
free	mapmalloc(3X)	memory allocator
free_field	form_field_new(3X)	create and destroy forms fields
free_fieldtype	form_fieldtype(3X)	forms fieldtype routines
free_form	form_new(3X)	create and destroy forms
free_item	menu_item_new(3X)	create and destroy menus items
free_menu	menu_new(3X)	create and destroy menus
freenetconfig	getnetconfig(3N)	get network configuration database entry
freopen	fopen(3B)	open a stream
freopen	fopen(3S)	open a stream
frexp	frexp(3C)	manipulate parts of floating-point numbers
fscanf	scanf(3S)	convert formatted input

fseek	fseek(3S)	reposition a file pointer in a stream
fsetpos	fsetpos(3C)	reposition a file pointer in a stream
fsync	fsync(3C)	synchronize a file's in-memory state with that on the physical medium
ftell	fseek(3S)	reposition a file pointer in a stream
ftime	ftime(3B)	get date and time
ftok	stdipc(3C)	standard interprocess communication package
ftruncate	truncate(3C)	set a file to a specified length
ftw	ftw(3C)	walk a file tree
func_to_decimal	string_to_decimal(3)	parse characters into decimal record
funlockfile	flockfile(3S)	acquire and release stream lock
fwrite	fread(3S)	buffered binary input/output
gamma	lgamma(3M)	log gamma function
gamma_r	lgamma(3M)	log gamma function
gconvert	econvert(3)	output conversion
gcvt	econvert(3)	output conversion
gcvt	ecvt(3C)	convert floating-point number to string
get_myaddress	rpc_soc(3N)	obsolete library routines for RPC
getacdir	getacinfo(3)	get audit control file information
getacflg	getacinfo(3)	get audit control file information
getacinfo	getacinfo(3)	get audit control file information
getacmin	getacinfo(3)	get audit control file information
getacna	getacinfo(3)	get audit control file information
getaclassent	getaclassent(3)	get audit_class entry
getaclassent_r	getaclassent(3)	get audit_class entry
getaclassnam	getaclassent(3)	get audit_class entry
getaclassnam_r	getaclassent(3)	get audit_class entry
getauditflags	getauditflags(3)	convert audit flag specifications

getauditflagsbin	getauditflags(3)	convert audit flag specifications
getauditflagschar	getauditflags(3)	convert audit flag specifications
getauevent	getauevent(3)	get audit_user entry
getauevent_r	getauevent(3)	get audit_user entry
getauevnam	getauevent(3)	get audit_user entry
getauevnam_r	getauevent(3)	get audit_user entry
getauevnonam	getauevent(3)	get audit_user entry
getauevnum	getauevent(3)	get audit_user entry
getauevnum_r	getauevent(3)	get audit_user entry
getauserent	getausernam(3)	get audit_user entry
getausernam	getausernam(3)	get audit_user entry
getbegyx	curs_getyx(3X)	get curses cursor and window coordinates
getc	getc(3S)	get character or word from a stream
getc_unlocked	getc(3S)	get character or word from a stream
getch	curs_getch(3X)	get (or push back) characters from curses terminal keyboard
getchar	getc(3S)	get character or word from a stream
getchar_unlocked	getc(3S)	get character or word from a stream
getcwd	getcwd(3C)	get pathname of current working directory
getdate	getdate(3C)	convert user format date and time
getdtablesize	getdtablesize(3B)	get file descriptor table size
getenv	getenv(3C)	return value for environment name
getfauditflags	getfauditflags(3)	generates the process audit state
getgrent	getgrnam(3C)	get group entry
getgrent_r	getgrnam(3C)	get group entry
getgrgid	getgrnam(3C)	get group entry
getgrgid_r	getgrnam(3C)	get group entry
getgrnam	getgrnam(3C)	get group entry
getgrnam_r	getgrnam(3C)	get group entry
gethostbyaddr	gethostbyname(3N)	get network host entry
gethostbyaddr_r	gethostbyname(3N)	get network host entry
gethostbyname	gethostbyname(3N)	get network host entry

gethostbyname_r	gethostbyname(3N)	get network host entry
gethostent	gethostbyname(3N)	get network host entry
gethostent_r	gethostbyname(3N)	get network host entry
gethostid	gethostid(3B)	get unique identifier of current host
gethostname	gethostname(3B)	get/set name of current host
gethrtime	gethrtime(3C)	get high resolution time
gethrvtime	gethrtime(3C)	get high resolution time
getlogin	getlogin(3C)	get login name
getlogin_r	getlogin(3C)	get login name
getmaxyx	curs_getyx(3X)	get curses cursor and window coordinates
getmntany	getmntent(3C)	get mnttab file information
getmntent	getmntent(3C)	get mnttab file information
getnetbyaddr	getnetbyname(3N)	get network entry
getnetbyaddr_r	getnetbyname(3N)	get network entry
getnetbyname	getnetbyname(3N)	get network entry
getnetbyname_r	getnetbyname(3N)	get network entry
getnetconfig	getnetconfig(3N)	get network configuration database entry
getnetconfigent	getnetconfig(3N)	get network configuration database entry
getnetent	getnetbyname(3N)	get network entry
getnetent_r	getnetbyname(3N)	get network entry
getnetgrent	getnetgrent(3N)	get network group entry
getnetgrent_r	getnetgrent(3N)	get network group entry
getnetname	secure_rpc(3N)	library routines for secure remote procedure calls
getnetpath	getnetpath(3N)	get /etc/netconfig entry corresponding to NETPATH component
getnwstr	curs_getwstr(3X)	get wchar_t character strings from curses terminal keyboard
getopt	getopt(3C)	get option letter from argument vector
getpagesize	getpagesize(3B)	get system page size
getparyx	curs_getyx(3X)	get curses cursor and window coordinates
getpass	getpass(3C)	read a password
getpeername	getpeername(3N)	get name of connected peer
getpriority	getpriority(3B)	get/set scheduling priority for process, process group or user
getprotobyname	getprotobyname(3N)	get protocol entry

getprotobyname_r	getprotobyname(3N)	get protocol entry
getprotobyname	getprotobyname(3N)	get protocol entry
getprotobyname_r	getprotobyname(3N)	get protocol entry
getprotoent	getprotobyname(3N)	get protocol entry
getprotoent_r	getprotobyname(3N)	get protocol entry
getpublickey	getpublickey(3N)	retrieve public or secret key
getpw	getpw(3C)	get passwd entry from UID
getpwent	getpwnam(3C)	get password entry
getpwent_r	getpwnam(3C)	get password entry
getpwnam	getpwnam(3C)	get password entry
getpwnam_r	getpwnam(3C)	get password entry
getpwuid	getpwnam(3C)	get password entry
getpwuid_r	getpwnam(3C)	get password entry
getrpcbyname	getrpcbyname(3N)	get RPC entry
getrpcbyname_r	getrpcbyname(3N)	get RPC entry
getrpcbynumber	getrpcbyname(3N)	get RPC entry
getrpcbynumber_r	getrpcbyname(3N)	get RPC entry
getrpcent	getrpcbyname(3N)	get RPC entry
getrpcent_r	getrpcbyname(3N)	get RPC entry
getrusage	getrusage(3B)	get information about resource utilization
gets	gets(3S)	get a string from a stream
getsecretkey	getpublickey(3N)	retrieve public or secret key
getservbyname	getservbyname(3N)	get service entry
getservbyname_r	getservbyname(3N)	get service entry
getservbyport	getservbyname(3N)	get service entry
getservbyport_r	getservbyname(3N)	get service entry
getservent	getservbyname(3N)	get service entry
getservent_r	getservbyname(3N)	get service entry
getsockname	getsockname(3N)	get socket name
getsockopt	getsockopt(3N)	get and set options on sockets
getspent	getspnam(3C)	get password entry
getspent_r	getspnam(3C)	get password entry
getspnam	getspnam(3C)	get password entry
getspnam_r	getspnam(3C)	get password entry
getstr	curs_getstr(3X)	get character strings from curses terminal keyboard
getsubopt	getsubopt(3C)	parse suboptions from a string
getsyx	curs_kernel(3X)	low-level curses routines
gettext	gettext(3I)	message handling functions
gettimeofday	gettimeofday(3B)	get or set the date and time

gettimeofday	gettimeofday(3C)	get or set the date and time
gettext	gettext(3C)	retrieve a text string
getutent	getutent(3C)	access utmp file entry
getutid	getutent(3C)	access utmp file entry
getutline	getutent(3C)	access utmp file entry
getutmp	getutxent(3C)	access utmpx file entry
getutmpx	getutxent(3C)	access utmpx file entry
getutxent	getutxent(3C)	access utmpx file entry
getutxid	getutxent(3C)	access utmpx file entry
getutxline	getutxent(3C)	access utmpx file entry
getvfsany	getvfsent(3C)	get vfstab file entry
getvfsent	getvfsent(3C)	get vfstab file entry
getvfsfile	getvfsent(3C)	get vfstab file entry
getvfspec	getvfsent(3C)	get vfstab file entry
getw	getc(3S)	get character or word from a stream
getwc	getwc(3I)	convert EUC character from the stream to Process Code
getwch	curs_getwch(3X)	get (or push back) wchar_t characters from curses terminal keyboard
getwchar	getwc(3I)	convert EUC character from the stream to Process Code
getwd	getwd(3B)	get current working directory pathname
getwidth	getwidth(3I)	get codeset information
getwin	curs_util(3X)	curses miscellaneous utility routines
getws	getws(3I)	convert a string of EUC characters from the stream to Process Code
getwstr	curs_getwstr(3X)	get wchar_t character strings from curses terminal keyboard
getyx	curs_getyx(3X)	get curses cursor and window coordinates
gmatch	gmatch(3G)	shell global pattern matching
gmtime	ctime(3C)	convert date and time to string
gmtime_r	ctime(3C)	convert date and time to string
grantpt	grantpt(3C)	grant access to the slave pseudo-terminal device
gsignal	ssignal(3C)	software signals

halfdelay	 curs_inopts(3X)	curses terminal input option control routines
has_colors	 curs_color(3X)	curses color manipulation routines
has_ic	 curs_termattrs(3X)	curses environment query routines
has_il	 curs_termattrs(3X)	curses environment query routines
hasmntopt	 getmntent(3C)	get mnttab file information
havedisk	 rstat(3N)	get performance data from remote kernel
hcreate	 hsearch(3C)	manage hash search tables
hdestroy	 hsearch(3C)	manage hash search tables
hide_panel	 panel_show(3X)	panels deck manipulation routines
host2netname	 secure_rpc(3N)	library routines for secure remote procedure calls
hsearch	 hsearch(3C)	manage hash search tables
htonl	 byteorder(3N)	convert values between host and network byte order
htons	 byteorder(3N)	convert values between host and network byte order
hyperbolic	 hyperbolic(3M)	hyperbolic functions
hypot	 hypot(3M)	Euclidean distance
iconv	 iconv(3)	code conversion function
iconv_close	 iconv_close(3)	code conversion deallocation function
iconv_open	 iconv_open(3)	code conversion allocation function
idcok	 curs_outopts(3X)	curses terminal output option control routines
idlok	 curs_outopts(3X)	curses terminal output option control routines
ieee_functions	 ieee_functions(3M)	appendix and related miscellaneous functions for IEEE arithmetic
ieee_test	 ieee_test(3M)	IEEE test functions for verifying standard compliance
ilogb	 ieee_functions(3M)	appendix and related miscellaneous functions for IEEE arithmetic
immedok	 curs_outopts(3X)	curses terminal output option control routines

inch	 curs_inch(3X)	get a character and its attributes from a curses window
inchnstr	 curs_inchnstr(3X)	get a string of characters (and attributes) from a curses window
inchstr	 curs_inchstr(3X)	get a string of characters (and attributes) from a curses window
index	 index(3B)	string operations
inet	 inet(3N)	Internet address manipulation
inet_addr	 inet(3N)	Internet address manipulation
inet_lnaof	 inet(3N)	Internet address manipulation
inet_makeaddr	 inet(3N)	Internet address manipulation
inet_netof	 inet(3N)	Internet address manipulation
inet_network	 inet(3N)	Internet address manipulation
inet_ntoa	 inet(3N)	Internet address manipulation
init_color	 curs_color(3X)	curses color manipulation routines
init_pair	 curs_color(3X)	curses color manipulation routines
initgroups	 initgroups(3C)	initialize the supplementary group access list
initscr	 curs_initscr(3X)	curses screen initialization and manipulation routines
initstate	 random(3B)	better random number generator; routines for changing generators
innetgr	 getnetgrent(3N)	get network group entry
innstr	 curs_instr(3X)	get a string of characters from a curses window
innwstr	 curs_inwstr(3X)	get a string of wchar_t characters from a curses window
insch	 curs_insch(3X)	insert a character before the character under the cursor in a curses window

insdelln	 curs_deleteln(3X)	delete and insert lines in a curses window
insertln	 curs_deleteln(3X)	delete and insert lines in a curses window
insnstr	 curs_insstr(3X)	insert string before character under the cursor in a curses window
insnwstr	 curs_inswstr(3X)	insert wchar_t string before character under the cursor in a curses window
insque	 insque(3C)	insert/remove element from a queue
insstr	 curs_insstr(3X)	insert string before character under the cursor in a curses window
instr	 curs_instr(3X)	get a string of characters from a curses window
inwch	 curs_inwch(3X)	insert a wchar_t character before the character under the cursor in a curses window
inswstr	 curs_inswstr(3X)	insert wchar_t string before character under the cursor in a curses window
intrflush	 curs_inopts(3X)	curses terminal input option control routines
inwch	 curs_inwch(3X)	get a wchar_t character and its attributes from a curses window
inwchnstr	 curs_inwchstr(3X)	get a string of wchar_t characters (and attributes) from a curses window
inwchstr	 curs_inwchstr(3X)	get a string of wchar_t characters (and attributes) from a curses window
inwstr	 curs_inwstr(3X)	get a string of wchar_t characters from a curses window
is_linetouched	 curs_touch(3X)	curses refresh control routines
is_wintouched	 curs_touch(3X)	curses refresh control routines
isalnum	 ctype(3C)	character handling
isalpha	 ctype(3C)	character handling
isascii	 ctype(3C)	character handling

isastream	isastream(3C)	test a file descriptor
isatty	ttyname(3C)	find name of a terminal
iscntrl	ctype(3C)	character handling
isdigit	ctype(3C)	character handling
isencrypt	isencrypt(3G)	determine whether a buffer of characters is encrypted
isendwin	curs_initscr(3X)	curses screen initialization and manipulation routines
isenglish	iswalpha(3I)	Process Code character classification macros and functions
isgraph	ctype(3C)	character handling
isideogram	iswalpha(3I)	Process Code character classification macros and functions
islower	ctype(3C)	character handling
isnan	ieee_functions(3M)	appendix and related miscellaneous functions for IEEE arithmetic
isnan	isnan(3C)	determine type of floating-point number
isnand	isnan(3C)	determine type of floating-point number
isnanf	isnan(3C)	determine type of floating-point number
isnumber	iswalpha(3I)	Process Code character classification macros and functions
isphonogram	iswalpha(3I)	Process Code character classification macros and functions
isprint	ctype(3C)	character handling
ispunct	ctype(3C)	character handling
isspace	ctype(3C)	character handling
isspecial	iswalpha(3I)	Process Code character classification macros and functions
isupper	ctype(3C)	character handling
iswalnum	iswalpha(3I)	Process Code character classification macros and functions
iswalpha	iswalpha(3I)	Process Code character classification macros and functions

iswascii	iswalpha(3I)	Process Code character classification macros and functions
iswcntrl	iswalpha(3I)	Process Code character classification macros and functions
iswctype	iswctype(3I)	test character for specified class
iswdigit	iswalpha(3I)	Process Code character classification macros and functions
iswgraph	iswalpha(3I)	Process Code character classification macros and functions
iswlower	iswalpha(3I)	Process Code character classification macros and functions
iswprint	iswalpha(3I)	Process Code character classification macros and functions
iswpunct	iswalpha(3I)	Process Code character classification macros and functions
iswspace	iswalpha(3I)	Process Code character classification macros and functions
iswupper	iswalpha(3I)	Process Code character classification macros and functions
iswxdigit	iswalpha(3I)	Process Code character classification macros and functions
isxdigit	ctype(3C)	character handling
item_count	menu_items(3X)	connect and disconnect items to and from menus
item_description	menu_item_name(3X)	get menus item name and description
item_index	menu_item_current(3X)	set and get current menus items
item_init	menu_hook(3X)	assign application-specific routines for automatic invocation by menus
item_name	menu_item_name(3X)	get menus item name and description
item_opts	menu_item_opts(3X)	menus item option routines

item_opts_off	menu_item_opts(3X)	menus item option routines
item_opts_on	menu_item_opts(3X)	menus item option routines
item_term	menu_hook(3X)	assign application-specific routines for automatic invocation by menus
item_userptr	menu_item_userptr(3X)	associate application data with menus items
item_value	menu_item_value(3X)	set and get menus item values
item_visible	menu_item_visible(3X)	tell if menus item is visible
j0	bessel(3M)	Bessel functions
j1	bessel(3M)	Bessel functions
jn	bessel(3M)	Bessel functions
jrand48	drand48(3C)	generate uniformly distributed pseudo-random numbers
kerberos	kerberos(3N)	Kerberos authentication library
kerberos_rpc	kerberos_rpc(3N)	library routines for remote procedure calls using Kerberos authentication library routines for secure remote procedure calls
key_decryptsession	secure_rpc(3N)	library routines for secure remote procedure calls
key_encryptsession	secure_rpc(3N)	library routines for secure remote procedure calls
key_gendes	secure_rpc(3N)	library routines for secure remote procedure calls
key_secretkey_is_set	secure_rpc(3N)	library routines for secure remote procedure calls
key_setsecret	secure_rpc(3N)	library routines for secure remote procedure calls
keyname	curs_util(3X)	curses miscellaneous utility routines
keypad	curs_inopts(3X)	curses terminal input option control routines
killchar	curs_termattrs(3X)	curses environment query routines
killpg	killpg(3B)	send signal to a process group
krb_get_admhst	krb_realmofhost(3N)	additional Kerberos utility routines
krb_get_cred	kerberos(3N)	Kerberos authentication library

krb_get_krbhst	krb_realmofhost(3N)	additional Kerberos utility routines
krb_get_lrealm	krb_realmofhost(3N)	additional Kerberos utility routines
krb_get_phost	krb_realmofhost(3N)	additional Kerberos utility routines
krb_kntoln	kerberos(3N)	Kerberos authentication library
krb_mk_err	kerberos(3N)	Kerberos authentication library
krb_mk_req	kerberos(3N)	Kerberos authentication library
krb_mk_safe	kerberos(3N)	Kerberos authentication library
krb_net_read	krb_sendauth(3N)	Kerberos routines for sending authentication via network stream sockets
krb_net_write	krb_sendauth(3N)	Kerberos routines for sending authentication via network stream sockets
krb_rd_err	kerberos(3N)	Kerberos authentication library
krb_rd_req	kerberos(3N)	Kerberos authentication library
krb_rd_safe	kerberos(3N)	Kerberos authentication library
krb_realmofhost	krb_realmofhost(3N)	additional Kerberos utility routines
krb_recvauth	krb_sendauth(3N)	Kerberos routines for sending authentication via network stream sockets
krb_sendauth	krb_sendauth(3N)	Kerberos routines for sending authentication via network stream sockets
krb_set_key	kerberos(3N)	Kerberos authentication library
krb_set_tkt_string	krb_set_tkt_string(3N)	set Kerberos ticket cache file name
kstat	kstat(3K)	kernel statistics facility
kstat_chain_update	kstat_chain_update(3K)	update the kstat header chain
kstat_close	kstat_open(3K)	initialize kernel statistics facility
kstat_data_lookup	kstat_lookup(3K)	find a kstat by name
kstat_lookup	kstat_lookup(3K)	find a kstat by name

kstat_open	kstat_open(3K)	initialize kernel statistics facility
kstat_read	kstat_read(3K)	read or write kstat data
kstat_write	kstat_read(3K)	read or write kstat data
kvm_close	kvm_open(3K)	specify a kernel to examine
kvm_getcmd	kvm_getu(3K)	get the u-area or invocation arguments for a process
kvm_getproc	kvm_nextproc(3K)	read system process structures
kvm_getu	kvm_getu(3K)	get the u-area or invocation arguments for a process
kvm_nextproc	kvm_nextproc(3K)	read system process structures
kvm_nlist	kvm_nlist(3K)	get entries from kernel symbol table
kvm_open	kvm_open(3K)	specify a kernel to examine
kvm_read	kvm_read(3K)	copy data to or from a kernel image or running system
kvm_setproc	kvm_nextproc(3K)	read system process structures
kvm_write	kvm_read(3K)	copy data to or from a kernel image or running system
l64a	a64l(3C)	convert between long integer and base-64 ASCII string
labs	abs(3C)	return absolute value of integer
lckpddf	lckpddf(3C)	manipulate shadow password database lock file
lcong48	drand48(3C)	generate uniformly distributed pseudo-random numbers
ldexp	frexp(3C)	manipulate parts of floating-point numbers
ldiv	div(3C)	compute the quotient and remainder
leaveok	curs_outopts(3X)	curses terminal output option control routines
lfind	lsearch(3C)	linear search and update
lfmt	lfmt(3C)	display error message in standard format and pass to logging and monitoring services
lgamma	lgamma(3M)	log gamma function

lgamma_r	lgamma(3M)	log gamma function
link_field	form_field_new(3X)	create and destroy forms fields
link_fieldtype	form_fieldtype(3X)	forms fieldtype routines
lio_listio	lio_listio(3R)	list directed I/O
listen	listen(3N)	listen for connections on a socket
llabs	abs(3C)	return absolute value of integer
lldiv	div(3C)	compute the quotient and remainder
lltostr	strtol(3C)	conversion routines
localeconv	localeconv(3C)	get numeric formatting information
localtime	ctime(3C)	convert date and time to string
localtime_r	ctime(3C)	convert date and time to string
lockf	lockf(3C)	record locking on files
log10	exp(3M)	exponential, logarithm, power
log1p	exp(3M)	exponential, logarithm, power
log	exp(3M)	exponential, logarithm, power
logb	frexp(3C)	manipulate parts of floating-point numbers
logb	ieee_test(3M)	IEEE test functions for verifying standard compliance
longjmp	setjmp(3B)	non-local goto
longjmp	setjmp(3C)	non-local goto
longname	curs_termattrs(3X)	curses environment query routines
lrand48	drand48(3C)	generate uniformly distributed pseudo-random numbers
lsearch	lsearch(3C)	linear search and update
madvise	madvise(3)	provide advice to VM system
maillock	maillock(3X)	manage lockfile for user's mailbox
major	makedev(3C)	manage a device number
makecontext	makecontext(3C)	manipulate user contexts
makedev	makedev(3C)	manage a device number
mallinfo	malloc(3X)	memory allocator

malloc	bsdmalloc(3X)	memory allocator
malloc	malloc(3C)	memory allocator
malloc	malloc(3X)	memory allocator
mallopt	malloc(3X)	memory allocator
mapmalloc	mapmalloc(3X)	memory allocator
matherr	matherr(3M)	math library
		exception-handling function
mbchar	mbchar(3C)	multibyte character handling
mblen	mbchar(3C)	multibyte character handling
mbstowcs	mbstring(3C)	multibyte string functions
mbstring	mbstring(3C)	multibyte string functions
mbtowc	mbchar(3C)	multibyte character handling
mctl	mctl(3B)	memory management control
memalign	malloc(3C)	memory allocator
memccpy	memory(3C)	memory operations
memchr	memory(3C)	memory operations
memcmp	memory(3C)	memory operations
memcpy	memory(3C)	memory operations
memmove	memory(3C)	memory operations
memory	memory(3C)	memory operations
memset	memory(3C)	memory operations
menu_attributes	menu_attributes(3X)	control menus display attributes
menu_back	menu_attributes(3X)	control menus display attributes
menu_cursor	menu_cursor(3X)	correctly position a menu cursor
menu_driver	menu_driver(3X)	command processor for the menus subsystem
menu_fore	menu_attributes(3X)	control menus display attributes
menu_format	menu_format(3X)	set and get maximum numbers of rows and columns in menus
menu_grey	menu_attributes(3X)	control menus display attributes
menu_hook	menu_hook(3X)	assign application-specific routines for automatic invocation by menus
menu_init	menu_hook(3X)	assign application-specific routines for automatic invocation by menus
menu_item_current	menu_item_current(3X)	set and get current menu items

menu_item_name	menu_item_name(3X)	get menus item name and description
menu_item_new	menu_item_new(3X)	create and destroy menus items
menu_item_opts	menu_item_opts(3X)	menus item option routines
menu_item_userptr	menu_item_userptr(3X)	associate application data with menus items
menu_item_value	menu_item_value(3X)	set and get menus item values
menu_item_visible	menu_item_visible(3X)	tell if menus item is visible
menu_items	menu_items(3X)	connect and disconnect items to and from menus
menu_mark	menu_mark(3X)	menus mark string routines
menu_new	menu_new(3X)	create and destroy menus
menu_opts	menu_opts(3X)	menus option routines
menu_opts_off	menu_opts(3X)	menus option routines
menu_opts_on	menu_opts(3X)	menus option routines
menu_pad	menu_attributes(3X)	control menus display attributes
menu_pattern	menu_pattern(3X)	set and get menus pattern match buffer
menu_post	menu_post(3X)	write or erase menus from associated subwindows
menu_sub	menu_win(3X)	menus window and subwindow association routines
menu_term	menu_hook(3X)	assign application-specific routines for automatic invocation by menus
menu_userptr	menu_userptr(3X)	associate application data with menus
menu_win	menu_win(3X)	menus window and subwindow association routines
menus	menus(3X)	character based menus package
meta	curs_inopts(3X)	curses terminal input option control routines
minor	makedev(3C)	manage a device number
mkdirp	mkdirp(3G)	create, remove directories in a path
mkfifo	mkfifo(3C)	create a new FIFO
mktemp	mktemp(3C)	make a unique file name
mktime	mktime(3C)	converts a tm structure to a calendar time

mlock	mlock(3C)	lock (or unlock) pages in memory
mlockall	mlockall(3C)	lock or unlock address space
modf	frexp(3C)	manipulate parts of floating-point numbers
modff	frexp(3C)	manipulate parts of floating-point numbers
monitor	monitor(3C)	prepare process execution profile
move	curs_move(3X)	move curses window cursor
move_field	form_field(3X)	connect fields to forms
move_panel	panel_move(3X)	move a panels window on the virtual screen
movenextch	curs_alecompat(3X)	these functions are added to ALE curses library for moving the cursor by character.
moveprevch	curs_alecompat(3X)	these functions are added to ALE curses library for moving the cursor by character.
mq_close	mq_close(3R)	close a message queue
mq_getattr	mq_setattr(3R)	set/get message queue attributes
mq_notify	mq_notify(3R)	notify process (or thread) that a message is available on a queue
mq_open	mq_open(3R)	open a message queue
mq_receive	mq_receive(3R)	receive a message from a message queue
mq_send	mq_send(3R)	send a message to a message queue
mq_setattr	mq_setattr(3R)	set/get message queue attributes
mq_unlink	mq_unlink(3R)	remove a message queue
rand48	drand48(3C)	generate uniformly distributed pseudo-random numbers
msync	msync(3C)	synchronize memory with physical storage
munlock	mlock(3C)	lock (or unlock) pages in memory
munlockall	mlockall(3C)	lock or unlock address space
mutex	mutex(3T)	mutual exclusion locks
mutex_destroy	mutex(3T)	mutual exclusion locks

mutex_init	mutex(3T)	mutual exclusion locks
mutex_lock	mutex(3T)	mutual exclusion locks
mutex_trylock	mutex(3T)	mutual exclusion locks
mutex_unlock	mutex(3T)	mutual exclusion locks
mvaddch	curs_addch(3X)	add a character (with attributes) to a curses window and advance cursor
mvaddchnstr	curs_addchnstr(3X)	add string of characters (and attributes) to a curses window
mvaddchstr	curs_addchstr(3X)	add string of characters (and attributes) to a curses window
mvaddnstr	curs_addstr(3X)	add a string of characters to a curses window and advance cursor
mvaddnwstr	curs_addwstr(3X)	add a string of wchar_t characters to a curses window and advance cursor
mvaddstr	curs_addstr(3X)	add a string of characters to a curses window and advance cursor
mvaddwch	curs_addwch(3X)	add a wchar_t character (with attributes) to a curses window and advance cursor
mvaddwchnstr	curs_addwchnstr(3X)	add string of wchar_t characters (and attributes) to a curses window
mvaddwchstr	curs_addwchstr(3X)	add string of wchar_t characters (and attributes) to a curses window
mvaddwstr	curs_addwstr(3X)	add a string of wchar_t characters to a curses window and advance cursor
mvcur	curs_terminfo(3X)	curses interfaces to terminfo database
mvdelch	curs_delch(3X)	delete character under cursor in a curses window
mvderwin	curs_window(3X)	create curses windows
mvgetch	curs_getch(3X)	get (or push back) characters from curses terminal keyboard

mvgetnwstr	 curs_getwstr(3X)	get wchar_t character strings from curses terminal keyboard
mvgetstr	 curs_getstr(3X)	get character strings from curses terminal keyboard
mvgetwch	 curs_getwch(3X)	get (or push back) wchar_t characters from curses terminal keyboard
mvgetwstr	 curs_getwstr(3X)	get wchar_t character strings from curses terminal keyboard
mvinch	 curs_inch(3X)	get a character and its attributes from a curses window
mvinchnstr	 curs_inchstr(3X)	get a string of characters (and attributes) from a curses window
mvinchstr	 curs_inchstr(3X)	get a string of characters (and attributes) from a curses window
mvinnstr	 curs_instr(3X)	get a string of characters from a curses window
mvinnwstr	 curs_inwstr(3X)	get a string of wchar_t characters from a curses window
mvinsch	 curs_insch(3X)	insert a character before the character under the cursor in a curses window
mvinsnstr	 curs_insstr(3X)	insert string before character under the cursor in a curses window
mvinsnwstr	 curs_inswstr(3X)	insert wchar_t string before character under the cursor in a curses window
mvinsstr	 curs_insstr(3X)	insert string before character under the cursor in a curses window
mvinstr	 curs_instr(3X)	get a string of characters from a curses window
mvinswch	 curs_inswch(3X)	insert a wchar_t character before the character under the cursor in a curses window

mvinswstr	 curs_inswstr(3X)	insert wchar_t string before character under the cursor in a curses window
mvinwch	 curs_inwch(3X)	get a wchar_t character and its attributes from a curses window
mvinwchnstr	 curs_inwchstr(3X)	get a string of wchar_t characters (and attributes) from a curses window
mvinwchstr	 curs_inwchstr(3X)	get a string of wchar_t characters (and attributes) from a curses window
mvinwstr	 curs_inwstr(3X)	get a string of wchar_t characters from a curses window
mvprintw	 curs_printw(3X)	print formatted output in curses windows
mvscanw	 curs_scanw(3X)	convert formatted input from a curses window
mvwaddch	 curs_addch(3X)	add a character (with attributes) to a curses window and advance cursor
mvwaddchnstr	 curs_addchstr(3X)	add string of characters (and attributes) to a curses window
mvwaddchstr	 curs_addchstr(3X)	add string of characters (and attributes) to a curses window
mvwaddnstr	 curs_addstr(3X)	add a string of characters to a curses window and advance cursor
mvwaddnwstr	 curs_addwstr(3X)	add a string of wchar_t characters to a curses window and advance cursor
mvwaddstr	 curs_addstr(3X)	add a string of characters to a curses window and advance cursor
mvwaddwch	 curs_addwch(3X)	add a wchar_t character (with attributes) to a curses window and advance cursor
mvwaddwchnstr	 curs_addwchstr(3X)	add string of wchar_t characters (and attributes) to a curses window

mvwaddwchstr	 curs_addwchstr(3X)	add string of wchar_t characters (and attributes) to a curses window
mvwaddwstr	 curs_addwstr(3X)	add a string of wchar_t characters to a curses window and advance cursor
mvwdelch	 curs_delch(3X)	delete character under cursor in a curses window
mvwgetch	 curs_getch(3X)	get (or push back) characters from curses terminal keyboard
mvwgetnwstr	 curs_getwstr(3X)	get wchar_t character strings from curses terminal keyboard
mvwgetstr	 curs_getstr(3X)	get character strings from curses terminal keyboard
mvwgetwch	 curs_getwch(3X)	get (or push back) wchar_t characters from curses terminal keyboard
mvwgetwstr	 curs_getwstr(3X)	get wchar_t character strings from curses terminal keyboard
mvwin	 curs_window(3X)	create curses windows
mvwinch	 curs_inch(3X)	get a character and its attributes from a curses window
mvwinchnstr	 curs_inchstr(3X)	get a string of characters (and attributes) from a curses window
mvwinchstr	 curs_inchstr(3X)	get a string of characters (and attributes) from a curses window
mvwinnstr	 curs_instr(3X)	get a string of characters from a curses window
mvwinnwstr	 curs_inwstr(3X)	get a string of wchar_t characters from a curses window
mvwinsch	 curs_insch(3X)	insert a character before the character under the cursor in a curses window
mvwinsnstr	 curs_insstr(3X)	insert string before character under the cursor in a curses window

mvwinsnwstr	 curs_inswstr(3X)	insert wchar_t string before character under the cursor in a curses window
mvwinsstr	 curs_insstr(3X)	insert string before character under the cursor in a curses window
mvwinstr	 curs_instr(3X)	get a string of characters from a curses window
mvwinswch	 curs_inswch(3X)	insert a wchar_t character before the character under the cursor in a curses window
mvwinswstr	 curs_inswstr(3X)	insert wchar_t string before character under the cursor in a curses window
mvwinwch	 curs_inwch(3X)	get a wchar_t character and its attributes from a curses window
mvwinwchnstr	 curs_inwchstr(3X)	get a string of wchar_t characters (and attributes) from a curses window
mvwinwchstr	 curs_inwchstr(3X)	get a string of wchar_t characters (and attributes) from a curses window
mvwinwstr	 curs_inwstr(3X)	get a string of wchar_t characters from a curses window
mvwprintw	 curs_printw(3X)	print formatted output in curses windows
mvwscanw	 curs_scanw(3X)	convert formatted input from a curses widow
nanosleep	 nanosleep(3R)	high resolution sleep
napms	 curs_kernel(3X)	low-level curses routines
nc_perror	 getnetconfig(3N)	get network configuration database entry
nc_spperror	 getnetconfig(3N)	get network configuration database entry
ndbm	 ndbm(3)	data base subroutines
netdir	 netdir(3N)	generic transport name-to-address translation
netdir_free	 netdir(3N)	generic transport name-to-address translation
netdir_getbyaddr	 netdir(3N)	generic transport name-to-address translation

netdir_getbyname	netdir(3N)	generic transport name-to-address translation
netdir_mergeaddr	netdir(3N)	generic transport name-to-address translation
netdir_options	netdir(3N)	generic transport name-to-address translation
netdir_perror	netdir(3N)	generic transport name-to-address translation
netdir_sperror	netdir(3N)	generic transport name-to-address translation
netname2host	secure_rpc(3N)	library routines for secure remote procedure calls
netname2user	secure_rpc(3N)	library routines for secure remote procedure calls
new_field	form_field_new(3X)	create and destroy forms fields
new_fieldtype	form_fieldtype(3X)	forms fieldtype routines
new_form	form_new(3X)	create and destroy forms
new_item	menu_item_new(3X)	create and destroy menus items
new_menu	menu_new(3X)	create and destroy menus
new_page	form_new_page(3X)	forms pagination
new_panel	panel_new(3X)	create and destroy panels
newpad	curs_pad(3X)	create and display curses pads
newterm	curs_initscr(3X)	curses screen initialization and manipulation routines
newwin	curs_window(3X)	create curses windows
nextafter	frexp(3C)	manipulate parts of floating-point numbers
nextafter	ieee_functions(3M)	appendix and related miscellaneous functions for IEEE arithmetic
nextkey	dbm(3B)	data base subroutines
nftw	ftw(3C)	walk a file tree
nice	nice(3B)	change priority of a process
nis_add	nis_names(3N)	NIS+ namespace functions
nis_add_entry	nis_tables(3N)	NIS+ table functions
nis_addmember	nis_groups(3N)	NIS+ group manipulation functions
nis_checkpoint	nis_ping(3N)	misc NIS+ log administration functions
nis_clone_object	nis_subr(3N)	NIS+ subroutines

nis_creategroup	nis_groups(3N)	NIS+ group manipulation functions
nis_db	nis_db(3N)	NIS+ Database access functions
nis_destroy_object	nis_subr(3N)	NIS+ subroutines
nis_destroygroup	nis_groups(3N)	NIS+ group manipulation functions
nis_dir_cmp	nis_subr(3N)	NIS+ subroutines
nis_domain_of	nis_subr(3N)	NIS+ subroutines
nis_error	nis_error(3N)	display NIS+ error messages
nis_first_entry	nis_tables(3N)	NIS+ table functions
nis_freenames	nis_subr(3N)	NIS+ subroutines
nis_freeresult	nis_names(3N)	NIS+ namespace functions
nis_freeservlist	nis_server(3N)	miscellaneous NIS+ functions
nis_freetags	nis_server(3N)	miscellaneous NIS+ functions
nis_getnames	nis_subr(3N)	NIS+ subroutines
nis_getservlist	nis_server(3N)	miscellaneous NIS+ functions
nis_groups	nis_groups(3N)	NIS+ group manipulation functions
nis_ismember	nis_groups(3N)	NIS+ group manipulation functions
nis_leaf_of	nis_subr(3N)	NIS+ subroutines
nis_lerror	nis_error(3N)	display NIS+ error messages
nis_list	nis_tables(3N)	NIS+ table functions
nis_local_directory	nis_local_names(3N)	NIS+ local names
nis_local_group	nis_local_names(3N)	NIS+ local names
nis_local_host	nis_local_names(3N)	NIS+ local names
nis_local_names	nis_local_names(3N)	NIS+ local names
nis_local_principal	nis_local_names(3N)	NIS+ local names
nis_lookup	nis_names(3N)	NIS+ namespace functions
nis_map_group	nis_groups(3N)	NIS+ group manipulation functions
nis_mkdir	nis_server(3N)	miscellaneous NIS+ functions
nis_modify	nis_names(3N)	NIS+ namespace functions
nis_modify_entry	nis_tables(3N)	NIS+ table functions
nis_name_of	nis_subr(3N)	NIS+ subroutines
nis_names	nis_names(3N)	NIS+ namespace functions
nis_next_entry	nis_tables(3N)	NIS+ table functions
nis_objects	nis_objects(3N)	NIS+ object formats
nis_perror	nis_error(3N)	display NIS+ error messages
nis_ping	nis_ping(3N)	misc NIS+ log administration functions
nis_print_group_entry	nis_groups(3N)	NIS+ group manipulation functions
nis_print_object	nis_subr(3N)	NIS+ subroutines

nis_remove	nis_names(3N)	NIS+ namespace functions
nis_remove_entry	nis_tables(3N)	NIS+ table functions
nis_removemember	nis_groups(3N)	NIS+ group manipulation functions
nis_rmdir	nis_server(3N)	miscellaneous NIS+ functions
nis_server	nis_server(3N)	miscellaneous NIS+ functions
nis_servstate	nis_server(3N)	miscellaneous NIS+ functions
nis_sperrno	nis_error(3N)	display NIS+ error messages
nis_sperror	nis_error(3N)	display NIS+ error messages
nis_sperror_r	nis_error(3N)	display NIS+ error messages
nis_stats	nis_server(3N)	miscellaneous NIS+ functions
nis_subr	nis_subr(3N)	NIS+ subroutines
nis_tables	nis_tables(3N)	NIS+ table functions
nis_verifygroup	nis_groups(3N)	NIS+ group manipulation functions
nl	curs_outopts(3X)	curses terminal output option control routines
nl_langinfo	nl_langinfo(3C)	language information
nlist	nlist(3B)	get entries from symbol table
nlist	nlist(3E)	get entries from name list
nlsgetcall	nlsgetcall(3N)	get client's data passed via the listener
nlsprovider	nlsprovider(3N)	get name of transport provider
nlsrequest	nlsrequest(3N)	format and send listener service request message
nocbreak	curs_inopts(3X)	curses terminal input option control routines
nodelay	curs_inopts(3X)	curses terminal input option control routines
noecho	curs_inopts(3X)	curses terminal input option control routines
nonl	curs_outopts(3X)	curses terminal output option control routines
noqiflush	curs_inopts(3X)	curses terminal input option control routines
noraw	curs_inopts(3X)	curses terminal input option control routines
notimeout	curs_inopts(3X)	curses terminal input option control routines
nrand48	drand48(3C)	generate uniformly distributed pseudo-random numbers

ntohl	byteorder(3N)	convert values between host and network byte order
ntohs	byteorder(3N)	convert values between host and network byte order
offsetof	offsetof(3C)	offset of structure member
opendir	directory(3C)	directory operations
openlog	syslog(3)	control system log
overlay	curs_overlay(3X)	overlap and manipulate overlapped curses windows
overwrite	curs_overlay(3X)	overlap and manipulate overlapped curses windows
p2close	p2open(3G)	open, close pipes to and from a command
p2open	p2open(3G)	open, close pipes to and from a command
pair_content	curs_color(3X)	curses color manipulation routines
panel_above	panel_above(3X)	panels deck traversal primitives
panel_below	panel_above(3X)	panels deck traversal primitives
panel_hidden	panel_show(3X)	panels deck manipulation routines
panel_move	panel_move(3X)	move a panels window on the virtual screen
panel_new	panel_new(3X)	create and destroy panels
panel_show	panel_show(3X)	panels deck manipulation routines
panel_top	panel_top(3X)	panels deck manipulation routines
panel_update	panel_update(3X)	panels virtual screen refresh routine
panel_userptr	panel_userptr(3X)	associate application data with a panels panel
panel_window	panel_window(3X)	get or set the current window of a panels panel
panels	panels(3X)	character based panels package
pathfind	pathfind(3G)	search for named file in named directories
pclose	popen(3S)	initiate pipe to/from a process
pechochar	curs_pad(3X)	create and display curses pads

pechowchar	curs_pad(3X)	create and display curses pads
perror	perror(3C)	print system error messages
pfmt	pfmt(3C)	display error message in standard format
plock	plock(3C)	lock or unlock into memory process, text, or data
pmap_getmaps	rpc_soc(3N)	obsolete library routines for RPC
pmap_getport	rpc_soc(3N)	obsolete library routines for RPC
pmap_rmtcall	rpc_soc(3N)	obsolete library routines for RPC
pmap_set	rpc_soc(3N)	obsolete library routines for RPC
pmap_unset	rpc_soc(3N)	obsolete library routines for RPC
pnoutrefresh	curs_pad(3X)	create and display curses pads
popen	popen(3S)	initiate pipe to/from a process
pos_form_cursor	form_cursor(3X)	position forms window cursor
pos_menu_cursor	menu_cursor(3X)	correctly position a menus cursor
post_form	form_post(3X)	write or erase forms from associated subwindows
post_menu	menu_post(3X)	write or erase menus from associated subwindows
pow	exp(3M)	exponential, logarithm, power
prefresh	curs_pad(3X)	create and display curses pads
printf	printf(3B)	formatted output conversion
printf	printf(3S)	print formatted output
printw	curs_printw(3X)	print formatted output in curses windows
psiginfo	psignal(3C)	system signal messages
psignal	psignal(3B)	system signal messages
psignal	psignal(3C)	system signal messages
ptsname	ptsname(3C)	get name of the slave pseudo-terminal device
publickey	getpublickey(3N)	retrieve public or secret key

putc	putc(3S)	put character or word on a stream
putc_unlocked	putc(3S)	put character or word on a stream
putchar	putc(3S)	put character or word on a stream
putchar_unlocked	putc(3S)	put character or word on a stream
putenv	putenv(3C)	change or add value to environment
putmntent	getmntent(3C)	get mnttab file information
putp	 curs_terminfo(3X)	curses interfaces to terminfo database
putpwent	putpwent(3C)	write password file entry
puts	puts(3S)	put a string on a stream
putspent	putspent(3C)	write shadow password file entry
pututline	getutent(3C)	access utmp file entry
pututxline	getutxent(3C)	access utmpx file entry
putw	putc(3S)	put character or word on a stream
putwc	putwc(3I)	convert Process Code character to EUC and put on a stream
putwchar	putwc(3I)	convert Process Code character to EUC and put on a stream
putwin	 curs_util(3X)	curses miscellaneous utility routines
putws	putws(3I)	convert a string of Process Code characters to EUC characters and put it on a stream
qeconvert	econvert(3)	output conversion
qfconvert	econvert(3)	output conversion
qgconvert	econvert(3)	output conversion
qiflush	 curs_inopts(3X)	curses terminal input option control routines
qsort	qsort(3C)	quick sort
quadruple_to_decimal	floating_to_decimal(3)	convert floating-point value to decimal record
rac_drop	rpc_rac(3N)	remote asynchronous calls
rac_poll	rpc_rac(3N)	remote asynchronous calls
rac_recv	rpc_rac(3N)	remote asynchronous calls
rac_send	rpc_rac(3N)	remote asynchronous calls

raise	raise(3C)	send signal to program
rand	rand(3B)	simple random number generator
rand	rand(3C)	simple random-number generator
rand_r	rand(3C)	simple random-number generator
random	random(3B)	better random number generator; routines for changing generators
raw	curs_inopts(3X)	curses terminal input option control routines
rcmd	rcmd(3N)	routines for returning a stream to a remote command
re_comp	regex(3B)	regular expression handler
re_exec	regex(3B)	regular expression handler
read_vtoc	read_vtoc(3X)	read and write a disk's VTOC
readdir	directory(3C)	directory operations
readdir_r	directory(3C)	directory operations
realloc	bsdmalloc(3X)	memory allocator
realloc	malloc(3C)	memory allocator
realloc	malloc(3X)	memory allocator
realloc	mapmalloc(3X)	memory allocator
realpath	realpath(3C)	returns the real file name
reboot	reboot(3B)	reboot system or halt processor
recv	recv(3N)	receive a message from a socket
recvfrom	recv(3N)	receive a message from a socket
recvmsg	recv(3N)	receive a message from a socket
redrawwin	curs_refresh(3X)	refresh curses windows and lines
refresh	curs_refresh(3X)	refresh curses windows and lines
regcmp	regcmp(3G)	compile and execute regular expression
regex	regcmp(3G)	compile and execute regular expression
regex	regex(3B)	regular expression handler
regexpr	regexpr(3G)	regular expression compile and match routines

regterrpc	rpc_soc(3N)	obsolete library routines for RPC
remainder	ieee_functions(3M)	appendix and related miscellaneous functions for IEEE arithmetic
remove	remove(3C)	remove file
remque	insque(3C)	insert/remove element from a queue
replace_panel	panel_window(3X)	get or set the current window of a panels panel
res_init	resolver(3N)	resolver routines
res_mkquery	resolver(3N)	resolver routines
res_send	resolver(3N)	resolver routines
reset_prog_mode	curs_kernel(3X)	low-level curses routines
reset_shell_mode	curs_kernel(3X)	low-level curses routines
resetty	curs_kernel(3X)	low-level curses routines
resolver	resolver(3N)	resolver routines
restartterm	curs_terminfo(3X)	curses interfaces to terminfo database
rewind	fseek(3S)	reposition a file pointer in a stream
rewinddir	directory(3C)	directory operations
rexec	rexec(3N)	return stream to a remote command
rindex	index(3B)	string operations
rint	floor(3M)	round to integral value in floating-point format
ripoffline	curs_kernel(3X)	low-level curses routines
rmdirp	mkdirp(3G)	create, remove directories in a path
rnusers	rusers(3N)	return information about users on remote machines
rpc	rpc(3N)	library routines for remote procedure calls
rpc_broadcast	rpc_clnt_calls(3N)	library routines for client side calls
rpc_broadcast_exp	rpc_clnt_calls(3N)	library routines for client side calls
rpc_call	rpc_clnt_calls(3N)	library routines for client side calls
rpc_clnt_auth	rpc_clnt_auth(3N)	library routines for client side remote procedure call authentication

rpc_clnt_calls	rpc_clnt_calls(3N)	library routines for client side calls
rpc_clnt_create	rpc_clnt_create(3N)	library routines for dealing with creation and manipulation of CLIENT handles
rpc_control	rpc_control(3N)	library routine for manipulating global RPC attributes for client and server applications
rpc_createerr	rpc_clnt_create(3N)	library routines for dealing with creation and manipulation of CLIENT handles
rpc_rac	rpc_rac(3N)	remote asynchronous calls
rpc_reg	rpc_svc_reg(3N)	library routines for registering servers
rpc_soc	rpc_soc(3N)	obsolete library routines for RPC
rpc_svc_calls	rpc_svc_calls(3N)	library routines for RPC servers
rpc_svc_create	rpc_svc_create(3N)	library routines for the creation of server handles
rpc_svc_err	rpc_svc_err(3N)	library routines for server side remote procedure call errors
rpc_svc_reg	rpc_svc_reg(3N)	library routines for registering servers
rpc_xdr	rpc_xdr(3N)	XDR library routines for remote procedure calls
rpcb_getaddr	rpcbind(3N)	library routines for RPC bind service
rpcb_getmaps	rpcbind(3N)	library routines for RPC bind service
rpcb_gettime	rpcbind(3N)	library routines for RPC bind service
rpcb_rmtcall	rpcbind(3N)	library routines for RPC bind service
rpcb_set	rpcbind(3N)	library routines for RPC bind service
rpcb_unset	rpcbind(3N)	library routines for RPC bind service
rpcbind	rpcbind(3N)	library routines for RPC bind service

rresvport	rcmd(3N)	routines for returning a stream to a remote command
rstat	rstat(3N)	get performance data from remote kernel
ruserok	rcmd(3N)	routines for returning a stream to a remote command
rusers	rusers(3N)	return information about users on remote machines
rw_rdlock	rwlock(3T)	multiple readers, single writer locks
rw_tryrdlock	rwlock(3T)	multiple readers, single writer locks
rw_trywrlock	rwlock(3T)	multiple readers, single writer locks
rw_unlock	rwlock(3T)	multiple readers, single writer locks
rw_wrlock	rwlock(3T)	multiple readers, single writer locks
rwall	rwall(3N)	write to specified remote machines
rwlock	rwlock(3T)	multiple readers, single writer locks
rwlock_destroy	rwlock(3T)	multiple readers, single writer locks
rwlock_init	rwlock(3T)	multiple readers, single writer locks
savetty	curs_kernel(3X)	low-level curses routines
scalb	frexp(3C)	manipulate parts of floating-point numbers
scalb	ieee_test(3M)	IEEE test functions for verifying standard compliance
scalbn	ieee_functions(3M)	appendix and related miscellaneous functions for IEEE arithmetic
scale_form	form_win(3X)	forms window and subwindow association routines
scale_menu	menu_win(3X)	menus window and subwindow association routines
scandir	scandir(3B)	scan a directory
scanf	scanf(3S)	convert formatted input
scanw	curs_scanw(3X)	convert formatted input from a curses widow

sched_get_priority_max	sched_get_priority_max(3R)	get scheduling parameter limits
sched_get_priority_min	sched_get_priority_max(3R)	get scheduling parameter limits
sched_getparam	sched_setparam(3R)	set/get scheduling parameters
sched_getscheduler	sched_setscheduler(3R)	set/get scheduling policy and scheduling parameters
sched_rr_get_interval	sched_get_priority_max(3R)	get scheduling parameter limits
sched_setparam	sched_setparam(3R)	set/get scheduling parameters
sched_setscheduler	sched_setscheduler(3R)	set/get scheduling policy and scheduling parameters
sched_yield	sched_yield(3R)	yield processor
scr_dump	curs_scr_dump(3X)	read (write) a curses screen from (to) a file
scr_init	curs_scr_dump(3X)	read (write) a curses screen from (to) a file
scr_restore	curs_scr_dump(3X)	read (write) a curses screen from (to) a file
scr_set	curs_scr_dump(3X)	read (write) a curses screen from (to) a file
scrl	curs_scroll(3X)	scroll a curses window
scroll	curs_scroll(3X)	scroll a curses window
scrollok	curs_outopts(3X)	curses terminal output option control routines
seconvert	econvert(3)	output conversion
secure_rpc	secure_rpc(3N)	library routines for secure remote procedure calls
seed48	drand48(3C)	generate uniformly distributed pseudo-random numbers
seekdir	directory(3C)	directory operations
select	select(3C)	synchronous I/O multiplexing
sem_close	sem_close(3R)	close a named semaphore
sem_destroy	sem_destroy(3R)	destroy an unnamed semaphore
sem_getvalue	sem_getvalue(3R)	get the value of a semaphore
sem_init	sem_init(3R)	initialize an unnamed semaphore
sem_open	sem_open(3R)	initialize/open a named semaphore
sem_post	sem_post(3R)	increment the count of a semaphore

sem_trywait	sem_wait(3R)	acquire or wait for a semaphore
sem_unlink	sem_unlink(3R)	remove a named semaphore
sem_wait	sem_wait(3R)	acquire or wait for a semaphore
sema_destroy	semaphore(3T)	semaphores
sema_init	semaphore(3T)	semaphores
sema_post	semaphore(3T)	semaphores
sema_trywait	semaphore(3T)	semaphores
sema_wait	semaphore(3T)	semaphores
semaphore	semaphore(3T)	semaphores
send	send(3N)	send a message from a socket
sendmsg	send(3N)	send a message from a socket
sendto	send(3N)	send a message from a socket
set_current_field	form_page(3X)	set forms current page and field
set_current_item	menu_item_current(3X)	set and get current menus items
set_curterm	curs_terminfo(3X)	curses interfaces to terminfo database
set_field_back	form_field_attributes(3X)	format the general display attributes of forms
set_field_buffer	form_field_buffer(3X)	set and get forms field attributes
set_field_fore	form_field_attributes(3X)	format the general display attributes of forms
set_field_init	form_hook(3X)	assign application-specific routines for invocation by forms
set_field_just	form_field_just(3X)	format the general appearance of forms
set_field_opts	form_field_opts(3X)	forms field option routines
set_field_pad	form_field_attributes(3X)	format the general display attributes of forms
set_field_status	form_field_buffer(3X)	set and get forms field attributes
set_field_term	form_hook(3X)	assign application-specific routines for invocation by forms
set_field_type	form_field_validation(3X)	forms field data type validation
set_field_userptr	form_field_userptr(3X)	associate application data with forms
set_fieldtype_arg	form_fieldtype(3X)	forms fieldtype routines
set_fieldtype_choice	form_fieldtype(3X)	forms fieldtype routines

set_form_fields	form_field(3X)	connect fields to forms
set_form_init	form_hook(3X)	assign application-specific routines for invocation by forms
set_form_opts	form_opts(3X)	forms option routines
set_form_page	form_page(3X)	set forms current page and field
set_form_sub	form_win(3X)	forms window and subwindow association routines
set_form_term	form_hook(3X)	assign application-specific routines for invocation by forms
set_form_userptr	form_userptr(3X)	associate application data with forms
set_form_win	form_win(3X)	forms window and subwindow association routines
set_item_init	menu_hook(3X)	assign application-specific routines for automatic invocation by menus
set_item_opts	menu_item_opts(3X)	menus item option routines
set_item_term	menu_hook(3X)	assign application-specific routines for automatic invocation by menus
set_item_userptr	menu_item_userptr(3X)	associate application data with menu items
set_item_value	menu_item_value(3X)	set and get menu item values
set_max_field	form_field_buffer(3X)	set and get forms field attributes
set_menu_back	menu_attributes(3X)	control menu display attributes
set_menu_fore	menu_attributes(3X)	control menu display attributes
set_menu_format	menu_format(3X)	set and get maximum numbers of rows and columns in menus
set_menu_grey	menu_attributes(3X)	control menu display attributes
set_menu_init	menu_hook(3X)	assign application-specific routines for automatic invocation by menus
set_menu_items	menu_items(3X)	connect and disconnect items to and from menus
set_menu_mark	menu_mark(3X)	menus mark string routines
set_menu_opts	menu_opts(3X)	menus option routines

set_menu_pad	menu_attributes(3X)	control menus display attributes
set_menu_pattern	menu_pattern(3X)	set and get menus pattern match buffer
set_menu_sub	menu_win(3X)	menus window and subwindow association routines
set_menu_term	menu_hook(3X)	assign application-specific routines for automatic invocation by menus
set_menu_userptr	menu_userptr(3X)	associate application data with menus
set_menu_win	menu_win(3X)	menus window and subwindow association routines
set_new_page	form_new_page(3X)	forms pagination
set_panel_userptr	panel_userptr(3X)	associate application data with a panels panel
set_term	curs_initscr(3X)	curses screen initialization and manipulation routines
set_top_row	menu_item_current(3X)	set and get current menus items
setac	getacinfo(3)	get audit control file information
setauclass	getauclassent(3)	get audit_class entry
setauevent	getauevent(3)	get audit_user entry
setauser	getausernam(3)	get audit_user entry
setbuf	setbuf(3S)	assign buffering to a stream
setbuffer	setbuffer(3B)	assign buffering to a stream
setcat	setcat(3C)	define default catalog
setgrent	getgrnam(3C)	get group entry
sethostent	gethostbyname(3N)	get network host entry
sethostname	gethostname(3B)	get/set name of current host
setjmp	setjmp(3B)	non-local goto
setjmp	setjmp(3C)	non-local goto
setkey	crypt(3C)	generate encryption
setlabel	setlabel(3C)	define the label for pfmt() and lfmt().
setlinebuf	setbuffer(3B)	assign buffering to a stream
setlocale	setlocale(3C)	modify and query a program's locale
setlogmask	syslog(3)	control system log
setnetconfig	getnetconfig(3N)	get network configuration database entry
setnetent	getnetbyname(3N)	get network entry
setnetgrent	getnetgrent(3N)	get network group entry

setnetpath	getnetpath(3N)	get /etc/netconfig entry corresponding to NETPATH component
setpriority	getpriority(3B)	get/set scheduling priority for process, process group or user
setprotoent	getprotobyname(3N)	get protocol entry
setpwent	getpwnam(3C)	get password entry
setregid	setregid(3B)	set real and effective group IDs
setreuid	setreuid(3B)	set real and effective user IDs
setrpcent	getrpcbyname(3N)	get RPC entry
setscrreg	curs_outopts(3X)	curses terminal output option control routines
setservent	getservbyname(3N)	get service entry
setsockopt	getsockopt(3N)	get and set options on sockets
setspent	getspnam(3C)	get password entry
setstate	random(3B)	better random number generator; routines for changing generators
setsyx	curs_kernel(3X)	low-level curses routines
setterm	curs_terminfo(3X)	curses interfaces to terminfo database
settimeofday	gettimeofday(3B)	get or set the date and time
settimeofday	gettimeofday(3C)	get or set the date and time
setupterm	curs_terminfo(3X)	curses interfaces to terminfo database
setutent	getutent(3C)	access utmp file entry
setutxent	getutxent(3C)	access utmpx file entry
setvbuf	setbuf(3S)	assign buffering to a stream
sfconvert	econvert(3)	output conversion
sgconvert	econvert(3)	output conversion
shm_open	shm_open(3R)	open a shared memory object
shm_unlink	shm_unlink(3R)	remove a shared memory object
show_panel	panel_show(3X)	panels deck manipulation routines
shutdown	shutdown(3N)	shut down part of a full-duplex connection
sig2str	str2sig(3C)	translation between signal name and signal number
sigaddset	sigsetops(3C)	manipulate sets of signals
sigblock	sigblock(3B)	block signals

sigdelset	sigsetops(3C)	manipulate sets of signals
sigemptyset	sigsetops(3C)	manipulate sets of signals
sigfillset	sigsetops(3C)	manipulate sets of signals
sigfpe	sigfpe(3)	signal handling for specific SIGFPE codes
sighold	signal(3C)	simplified signal management for application processes
sigignore	signal(3C)	simplified signal management for application processes
siginterrupt	siginterrupt(3B)	allow signals to interrupt functions
sigismember	sigsetops(3C)	manipulate sets of signals
siglongjmp	setjmp(3C)	non-local goto
sigmask	sigblock(3B)	block signals
signal	signal(3B)	simplified software signal facilities
signal	signal(3C)	simplified signal management for application processes
significantd	ieee_test(3M)	IEEE test functions for verifying standard compliance
sigpause	sigblock(3B)	block signals
sigpause	signal(3C)	simplified signal management for application processes
sigqueue	sigqueue(3R)	queue a signal to a process
sigrelse	signal(3C)	simplified signal management for application processes
sigset	signal(3C)	simplified signal management for application processes
sigsetjmp	setjmp(3C)	non-local goto
sigsetmask	sigblock(3B)	block signals
sigsetops	sigsetops(3C)	manipulate sets of signals
sigstack	sigstack(3B)	set and/or get signal stack context
sigtimedwait	sigwaitinfo(3R)	wait for queued signals
sigvec	sigvec(3B)	software signal facilities
sigwaitinfo	sigwaitinfo(3R)	wait for queued signals
sin	trig(3M)	trigonometric functions
single_to_decimal	floating_to_decimal(3)	convert floating-point value to decimal record
sinh	hyperbolic(3M)	hyperbolic functions
sleep	sleep(3B)	suspend execution for interval

sleep	sleep(3C)	suspend execution for interval
slk_atroff	 curs_slk(3X)	curses soft label routines
slk_attron	 curs_slk(3X)	curses soft label routines
slk_attrset	 curs_slk(3X)	curses soft label routines
slk_clear	 curs_slk(3X)	curses soft label routines
slk_init	 curs_slk(3X)	curses soft label routines
slk_label	 curs_slk(3X)	curses soft label routines
slk_noutrefresh	 curs_slk(3X)	curses soft label routines
slk_refresh	 curs_slk(3X)	curses soft label routines
slk_restore	 curs_slk(3X)	curses soft label routines
slk_set	 curs_slk(3X)	curses soft label routines
slk_touch	 curs_slk(3X)	curses soft label routines
socket	socket(3N)	create an endpoint for communication
socketpair	socketpair(3N)	create a pair of connected sockets
spray	spray(3N)	scatter data in order to test the network
sprintf	printf(3B)	formatted output conversion
sprintf	printf(3S)	print formatted output
sqrt	sqrt(3M)	square root, cube root
rand48	drand48(3C)	generate uniformly distributed pseudo-random numbers
rand	rand(3B)	simple random number generator
rand	rand(3C)	simple random-number generator
random	random(3B)	better random number generator; routines for changing generators
sscanf	scanf(3S)	convert formatted input
ssignal	ssignal(3C)	software signals
standend	 curs_attr(3X)	curses character and window attribute control routines
standout	 curs_attr(3X)	curses character and window attribute control routines
start_color	 curs_color(3X)	curses color manipulation routines
stdio	stdio(3S)	standard buffered input/output package
stdipc	stdipc(3C)	standard interprocess communication package

step	regexpr(3G)	regular expression compile and match routines
store	dbm(3B)	data base subroutines
str2sig	str2sig(3C)	translation between signal name and signal number
str	strfind(3G)	string manipulations
strcadd	strncpy(3G)	copy strings, compressing or expanding escape codes
strcasecmp	string(3C)	string operations
strcat	string(3C)	string operations
strccpy	strncpy(3G)	copy strings, compressing or expanding escape codes
strchr	string(3C)	string operations
strcmp	string(3C)	string operations
strcoll	strcoll(3C)	string collation
strcpy	string(3C)	string operations
strcspn	string(3C)	string operations
strdup	string(3C)	string operations
streadd	strncpy(3G)	copy strings, compressing or expanding escape codes
strecpy	strncpy(3G)	copy strings, compressing or expanding escape codes
strerror	strerror(3C)	get error message string
strfind	strfind(3G)	string manipulations
strfmon	strfmon(3C)	convert monetary value to string
strftime	strftime(3C)	convert date and time to string
string	string(3C)	string operations
string_to_decimal	string_to_decimal(3)	parse characters into decimal record
strlen	string(3C)	string operations
strncasecmp	string(3C)	string operations
strncat	string(3C)	string operations
strncmp	string(3C)	string operations
strncpy	string(3C)	string operations
strpbrk	string(3C)	string operations
strptime	strptime(3C)	date and time conversion
strchr	string(3C)	string operations
strspn	strfind(3G)	string manipulations
strsignal	strsignal(3C)	get error message string
strspn	string(3C)	string operations
strstr	string(3C)	string operations

strtod	strtod(3C)	convert string to double-precision number
strtok	string(3C)	string operations
strtok_r	string(3C)	string operations
strtol	strtol(3C)	conversion routines
strtoll	strtol(3C)	conversion routines
strtoul	strtol(3C)	conversion routines
strtoull	strtol(3C)	conversion routines
strtrns	strfind(3G)	string manipulations
strxfrm	strxfrm(3C)	string transformation
subpad	curs_pad(3X)	create and display curses pads
subwin	curs_window(3X)	create curses windows
svc_auth_reg	rpc_svc_reg(3N)	library routines for registering servers
svc_control	rpc_svc_create(3N)	library routines for the creation of server handles
svc_create	rpc_svc_create(3N)	library routines for the creation of server handles
svc_destroy	rpc_svc_create(3N)	library routines for the creation of server handles
svc_dg_create	rpc_svc_create(3N)	library routines for the creation of server handles
svc_dg_enablecache	rpc_svc_calls(3N)	library routines for RPC servers
svc_done	rpc_svc_calls(3N)	library routines for RPC servers
svc_exit	rpc_svc_calls(3N)	library routines for RPC servers
svc_fd_create	rpc_svc_create(3N)	library routines for the creation of server handles
svc_fds	rpc_soc(3N)	obsolete library routines for RPC
svc_fdset	rpc_svc_calls(3N)	library routines for RPC servers
svc_freeargs	rpc_svc_calls(3N)	library routines for RPC servers
svc_getargs	rpc_svc_calls(3N)	library routines for RPC servers
svc_getcaller	rpc_soc(3N)	obsolete library routines for RPC
svc_getreq	rpc_soc(3N)	obsolete library routines for RPC

svc_getreq_common	rpc_svc_calls(3N)	library routines for RPC servers
svc_getreq_poll	rpc_svc_calls(3N)	library routines for RPC servers
svc_getreqset	rpc_svc_calls(3N)	library routines for RPC servers
svc_getrpcaller	rpc_svc_calls(3N)	library routines for RPC servers
svc_kerb_reg	kerberos_rpc(3N)	library routines for remote procedure calls using Kerberos authentication
svc_pollset	rpc_svc_calls(3N)	library routines for RPC servers
svc_raw_create	rpc_svc_create(3N)	library routines for the creation of server handles
svc_reg	rpc_svc_reg(3N)	library routines for registering servers
svc_register	rpc_soc(3N)	obsolete library routines for RPC
svc_run	rpc_svc_calls(3N)	library routines for RPC servers
svc_sendreply	rpc_svc_calls(3N)	library routines for RPC servers
svc_tli_create	rpc_svc_create(3N)	library routines for the creation of server handles
svc_tp_create	rpc_svc_create(3N)	library routines for the creation of server handles
svc_unreg	rpc_svc_reg(3N)	library routines for registering servers
svc_unregister	rpc_soc(3N)	obsolete library routines for RPC
svc_vc_create	rpc_svc_create(3N)	library routines for the creation of server handles
svcerr_auth	rpc_svc_err(3N)	library routines for server side remote procedure call errors
svcerr_decode	rpc_svc_err(3N)	library routines for server side remote procedure call errors
svcerr_noproc	rpc_svc_err(3N)	library routines for server side remote procedure call errors

svcerr_noprogram	rpc_svc_err(3N)	library routines for server side remote procedure call errors
svcerr_progvers	rpc_svc_err(3N)	library routines for server side remote procedure call errors
svcerr_systemerr	rpc_svc_err(3N)	library routines for server side remote procedure call errors
svcerr_weakauth	rpc_svc_err(3N)	library routines for server side remote procedure call errors
svcsd_create	rpc_soc(3N)	obsolete library routines for RPC
svscraw_create	rpc_soc(3N)	obsolete library routines for RPC
svctcp_create	rpc_soc(3N)	obsolete library routines for RPC
svscudp_bufcreate	rpc_soc(3N)	obsolete library routines for RPC
svscudp_create	rpc_soc(3N)	obsolete library routines for RPC
swab	swab(3C)	swap bytes
swapcontext	makecontext(3C)	manipulate user contexts
syncok	curs_window(3X)	create curses windows
sys_siglist	psignal(3B)	system signal messages
syscall	syscall(3B)	indirect system call
sysconf	sysconf(3C)	get configurable system variables
syslog	syslog(3)	control system log
system	system(3S)	issue a shell command
t_accept	t_accept(3N)	accept a connect request
t_alloc	t_alloc(3N)	allocate a library structure
t_bind	t_bind(3N)	bind an address to a transport endpoint
t_close	t_close(3N)	close a transport endpoint
t_connect	t_connect(3N)	establish a connection with another transport user
t_error	t_error(3N)	produce error message
t_free	t_free(3N)	free a library structure
t_getinfo	t_getinfo(3N)	get protocol-specific service information
t_getstate	t_getstate(3N)	get the current state

t_listen	t_listen(3N)	listen for a connect request
t_look	t_look(3N)	look at the current event on a transport endpoint
t_open	t_open(3N)	establish a transport endpoint
t_optmgmt	t_optmgmt(3N)	manage options for a transport endpoint
t_rcv	t_rcv(3N)	receive data or expedited data sent over a connection
t_rcvconnect	t_rcvconnect(3N)	receive the confirmation from a connect request
t_rcvdis	t_rcvdis(3N)	retrieve information from disconnect
t_rcvrel	t_rcvrel(3N)	acknowledge receipt of an orderly release indication
t_rcvudata	t_rcvudata(3N)	receive a data unit
t_rcvuderr	t_rcvuderr(3N)	receive a unit data error indication
t_snd	t_snd(3N)	send data or expedited data over a connection
t_snddis	t_snddis(3N)	send user-initiated disconnect request
t_sndrel	t_sndrel(3N)	initiate an orderly release
t_sndudata	t_sndudata(3N)	send a data unit
t_strerror	t_strerror(3N)	get error message string
t_sync	t_sync(3N)	synchronize transport library
t_unbind	t_unbind(3N)	disable a transport endpoint
taddr2uaddr	netdir(3N)	generic transport name-to-address translation
tan	trig(3M)	trigonometric functions
tanh	hyperbolic(3M)	hyperbolic functions
tcdrain	termios(3)	general terminal interface
tcflow	termios(3)	general terminal interface
tcflush	termios(3)	general terminal interface
tcgetattr	termios(3)	general terminal interface
tcgetpgrp	termios(3)	general terminal interface
tcgetsid	termios(3)	general terminal interface
tcsendbreak	termios(3)	general terminal interface
tcsetattr	termios(3)	general terminal interface
tcsetpgrp	tcsetpgrp(3C)	set foreground process group id of terminal
tcsetpgrp	termios(3)	general terminal interface
tdelete	tsearch(3C)	manage binary search trees
telldir	directory(3C)	directory operations

tempnam	tmpnam(3S)	create a name for a temporary file
termattrs	curs_termattrs(3X)	curses environment query routines
termios	termios(3)	general terminal interface
termname	curs_termattrs(3X)	curses environment query routines
textdomain	gettext(3I)	message handling functions
tfind	tsearch(3C)	manage binary search trees
tgetent	curs_termcap(3X)	curses interfaces (emulated) to the termcap library
tgetflag	curs_termcap(3X)	curses interfaces (emulated) to the termcap library
tgetnum	curs_termcap(3X)	curses interfaces (emulated) to the termcap library
tgetstr	curs_termcap(3X)	curses interfaces (emulated) to the termcap library
tgoto	curs_termcap(3X)	curses interfaces (emulated) to the termcap library
thr_continue	thr_suspend(3T)	suspend or continue thread execution
thr_create	thr_create(3T)	create a new thread of control
thr_exit	thr_exit(3T)	thread termination
thr_getconcurrency	thr_setconcurrency(3T)	get/set thread concurrency level
thr_getprio	thr_setprio(3T)	set/get a thread priority
thr_getspecific	thr_keycreate(3T)	thread-specific data
thr_join	thr_join(3T)	wait for thread termination
thr_keycreate	thr_keycreate(3T)	thread-specific data
thr_kill	thr_kill(3T)	send a signal to a thread
thr_min_stack	thr_min_stack(3T)	minimum stack space for a thread
thr_self	thr_self(3T)	get thread identifier
thr_setconcurrency	thr_setconcurrency(3T)	get/set thread concurrency level
thr_setprio	thr_setprio(3T)	set/get a thread priority
thr_setspecific	thr_keycreate(3T)	thread-specific data
thr_sigsetmask	thr_sigsetmask(3T)	change and/or examine calling thread's signal mask
thr_suspend	thr_suspend(3T)	suspend or continue thread execution
thr_yield	thr_yield(3T)	yield execution to another thread

tigetflag	curs_terminfo(3X)	curses interfaces to terminfo database
tigetnum	curs_terminfo(3X)	curses interfaces to terminfo database
tigetstr	curs_terminfo(3X)	curses interfaces to terminfo database
timeout	curs_inopts(3X)	curses terminal input option control routines
timer_create	timer_create(3R)	create a per-LWP timer
timer_delete	timer_delete(3R)	delete a per-LWP timer
timer_getoverrun	timer_settime(3R)	high-resolution timer operations
timer_gettime	timer_settime(3R)	high-resolution timer operations
timer_settime	timer_settime(3R)	high-resolution timer operations
times	times(3B)	get process times
tmpfile	tmpfile(3S)	create a temporary file
tmpnam	tmpnam(3S)	create a name for a temporary file
tmpnam_r	tmpnam(3S)	create a name for a temporary file
toascii	conv(3C)	translate characters
tolower	conv(3C)	translate characters
top_panel	panel_top(3X)	panels deck manipulation routines
top_row	menu_item_current(3X)	set and get current menus items
touchline	curs_touch(3X)	curses refresh control routines
touchwin	curs_touch(3X)	curses refresh control routines
toupper	conv(3C)	translate characters
towlower	wconv(3I)	Process Code character conversion macros
towupper	wconv(3I)	Process Code character conversion macros
tparm	curs_terminfo(3X)	curses interfaces to terminfo database
tputs	curs_termcap(3X)	curses interfaces (emulated) to the termcap library
tputs	curs_terminfo(3X)	curses interfaces to terminfo database
trig	trig(3M)	trigonometric functions

truncate	truncate(3C)	set a file to a specified length
tsearch	tsearch(3C)	manage binary search trees
ttyname	ttyname(3C)	find name of a terminal
ttyname_r	ttyname(3C)	find name of a terminal
ttyslot	ttyslot(3C)	find the slot in the utmp file of the current user
twalk	tsearch(3C)	manage binary search trees
typeahead	curs_inopts(3X)	control routines
tzset	ctime(3C)	convert date and time to string
tzsetwall	ctime(3C)	convert date and time to string
uaddr2taddr	netdir(3N)	generic transport name-to-address translation
ualarm	ualarm(3B)	schedule signal after interval in microseconds
ulckpwdf	lckpwdf(3C)	manipulate shadow password database lock file
ulltostr	strtol(3C)	conversion routines
unctrl	curs_util(3X)	curses miscellaneous utility routines
ungetc	ungetc(3S)	push character back onto input stream
ungetch	curs_getch(3X)	get (or push back) characters from curses terminal keyboard
ungetwc	ungetwc(3I)	push a Process Code character back into input stream
ungetwch	curs_getwch(3X)	get (or push back) wchar_t characters from curses terminal keyboard
unlockpt	unlockpt(3C)	unlock a pseudo-terminal master/slave pair
unordered	isnan(3C)	determine type of floating-point number
unpost_form	form_post(3X)	write or erase forms from associated subwindows
unpost_menu	menu_post(3X)	write or erase menus from associated subwindows
untouchwin	curs_touch(3X)	curses refresh control routines

update_panels	panel_update(3X)	panels virtual screen refresh routine
updwtmp	getutxent(3C)	access utmpx file entry
updwtmpx	getutxent(3C)	access utmpx file entry
use_env	curl_util(3X)	curses miscellaneous utility routines
user2netname	secure_rpc(3N)	library routines for secure remote procedure calls
usleep	usleep(3B)	suspend execution for interval in microseconds
utmpname	getutent(3C)	access utmp file entry
utmpxname	getutxent(3C)	access utmpx file entry
valloc	malloc(3C)	memory allocator
vfprintf	printf(3B)	formatted output conversion
vfprintf	vprintf(3S)	print formatted output of a variable argument list
vidattr	curl_terminfo(3X)	curses interfaces to terminfo database
vidputs	curl_terminfo(3X)	curses interfaces to terminfo database
vlfmt	vlfmt(3C)	display error message in standard format and pass to logging and monitoring services
vpfmt	vpfmt(3C)	display error message in standard format and pass to logging and monitoring services
vprintf	printf(3B)	formatted output conversion
vfprintf	vprintf(3S)	print formatted output of a variable argument list
vsprintf	printf(3B)	formatted output conversion
vsfprintf	vprintf(3S)	print formatted output of a variable argument list
vsyslog	vsyslog(3)	log message with a varargs argument list
vwprintw	curl_printw(3X)	print formatted output in curses windows
vwscanw	curl_scanw(3X)	convert formatted input from a curses widow
waddch	curl_addch(3X)	add a character (with attributes) to a curses window and advance cursor

waddchnstr	curs_addchstr(3X)	add string of characters (and attributes) to a curses window
waddchstr	curs_addchstr(3X)	add string of characters (and attributes) to a curses window
waddnstr	curs_addstr(3X)	add a string of characters to a curses window and advance cursor
waddnwstr	curs_addwstr(3X)	add a string of wchar_t characters to a curses window and advance cursor
waddstr	curs_addstr(3X)	add a string of characters to a curses window and advance cursor
waddwch	curs_addwch(3X)	add a wchar_t character (with attributes) to a curses window and advance cursor
waddwchnstr	curs_addwchstr(3X)	add string of wchar_t characters (and attributes) to a curses window
waddwchstr	curs_addwchstr(3X)	add string of wchar_t characters (and attributes) to a curses window
waddwstr	curs_addwstr(3X)	add a string of wchar_t characters to a curses window and advance cursor
wadjcurspos	curs_alecompat(3X)	these functions are added to ALE curses library for moving the cursor by character.
wait3	wait(3B)	wait for process to terminate or stop
wait4	wait(3B)	wait for process to terminate or stop
wait	wait(3B)	wait for process to terminate or stop
waitpid	wait(3B)	wait for process to terminate or stop
watof	wcstod(3I)	convert wide character string to double-precision number

watoi	wcstol(3I)	convert wide character string to long integer
watol	wcstol(3I)	convert wide character string to long integer
watoll	wcstol(3I)	convert wide character string to long integer
wattroff	 curs_attr(3X)	curses character and window attribute control routines
wattron	 curs_attr(3X)	curses character and window attribute control routines
wattrset	 curs_attr(3X)	curses character and window attribute control routines
wbkgd	 curs_bkgd(3X)	curses window background manipulation routines
wbkgdset	 curs_bkgd(3X)	curses window background manipulation routines
wborder	 curs_border(3X)	create curses borders, horizontal and vertical lines
wclear	 curs_clear(3X)	clear all or part of a curses window
wclrto bot	 curs_clear(3X)	clear all or part of a curses window
wclrtoeol	 curs_clear(3X)	clear all or part of a curses window
wconv	wconv(3I)	Process Code character conversion macros
wcscat	wcstring(3I)	wide character string operations
wcschr	wcstring(3I)	wide character string operations
wcscmp	wcstring(3I)	wide character string operations
wcscoll	wcscoll(3I)	wide character string comparison using collating information
wcscopy	wcstring(3I)	wide character string operations
wcscspn	wcstring(3I)	wide character string operations
wcsetno	cset(3I)	get information on EUC codesets
wcsftime	wcsftime(3I)	convert date and time to wide character string

wcslen	wcstring(3I)	wide character string operations
wcsncat	wcstring(3I)	wide character string operations
wcsncmp	wcstring(3I)	wide character string operations
wcsncpy	wcstring(3I)	wide character string operations
wcspbrk	wcstring(3I)	wide character string operations
wcsrchr	wcstring(3I)	wide character string operations
wcsspn	wcstring(3I)	wide character string operations
wctod	wctod(3I)	convert wide character string to double-precision number
wctok	wcstring(3I)	wide character string operations
wctol	wctol(3I)	convert wide character string to long integer
wctombs	mbstring(3C)	multibyte string functions
wctoul	wctoul(3I)	convert wide character string to unsigned long
wcstring	wcstring(3I)	wide character string operations
wcswcs	wcstring(3I)	wide character string operations
wcswidth	wcstring(3I)	wide character string operations
wcsxfrm	wcsxfrm(3I)	wide character string transformation
wctomb	mbchar(3C)	multibyte character handling
wctype	wctype(3I)	define character class
wcursyncup	curs_window(3X)	create curses windows
wcwidth	wcstring(3I)	wide character string operations
wdelch	curs_delch(3X)	delete character under cursor in a curses window
wdeleteln	curs_deleteln(3X)	delete and insert lines in a curses window
wechochar	curs_addch(3X)	add a character (with attributes) to a curses window and advance cursor

wchownchar	 curs_addwch(3X)	add a wchar_t character (with attributes) to a curses window and advance cursor
werase	 curs_clear(3X)	clear all or part of a curses window
wgetch	 curs_getch(3X)	get (or push back) characters from curses terminal keyboard
wgetnstr	 curs_getstr(3X)	get character strings from curses terminal keyboard
wgetnwstr	 curs_getwstr(3X)	get wchar_t character strings from curses terminal keyboard
wgetstr	 curs_getstr(3X)	get character strings from curses terminal keyboard
wgetwch	 curs_getwch(3X)	get (or push back) wchar_t characters from curses terminal keyboard
wgetwstr	 curs_getwstr(3X)	get wchar_t character strings from curses terminal keyboard
whline	 curs_border(3X)	create curses borders, horizontal and vertical lines
WIFEXITED	 wait(3B)	wait for process to terminate or stop
WIFSIGNALED	 wait(3B)	wait for process to terminate or stop
WIFSTOPPED	 wait(3B)	wait for process to terminate or stop
winch	 curs_inch(3X)	get a character and its attributes from a curses window
winchnstr	 curs_inchstr(3X)	get a string of characters (and attributes) from a curses window
winchstr	 curs_inchstr(3X)	get a string of characters (and attributes) from a curses window
windex	 wcstring(3I)	wide character string operations
winnstr	 curs_instr(3X)	get a string of characters from a curses window

winnwstr	curs_inwstr(3X)	get a string of wchar_t characters from a curses window
winsch	curs_insch(3X)	insert a character before the character under the cursor in a curses window
winsdelln	curs_deleteln(3X)	delete and insert lines in a curses window
winsertln	curs_deleteln(3X)	delete and insert lines in a curses window
winsnstr	curs_insstr(3X)	insert string before character under the cursor in a curses window
winsnwstr	curs_inswstr(3X)	insert wchar_t string before character under the cursor in a curses window
winsstr	curs_insstr(3X)	insert string before character under the cursor in a curses window
winstr	curs_instr(3X)	get a string of characters from a curses window
winswch	curs_inswch(3X)	insert a wchar_t character before the character under the cursor in a curses window
winswstr	curs_inswstr(3X)	insert wchar_t string before character under the cursor in a curses window
winwch	curs_inwch(3X)	get a wchar_t character and its attributes from a curses window
winwchnstr	curs_inwchstr(3X)	get a string of wchar_t characters (and attributes) from a curses window
winwchstr	curs_inwchstr(3X)	get a string of wchar_t characters (and attributes) from a curses window
winwstr	curs_inwstr(3X)	get a string of wchar_t characters from a curses window
wmove	curs_move(3X)	move curses window cursor

wmovenextch	curs_alecompat(3X)	these functions are added to ALE curses library for moving the cursor by character.
wmoveprevch	curs_alecompat(3X)	these functions are added to ALE curses library for moving the cursor by character.
wnoutrefresh	curs_refresh(3X)	refresh curses windows and lines
wprintw	curs_printw(3X)	print formatted output in curses windows
wredrawln	curs_refresh(3X)	refresh curses windows and lines
wrefresh	curs_refresh(3X)	refresh curses windows and lines
wrindex	wcstring(3I)	wide character string operations
write_vtoc	read_vtoc(3X)	read and write a disk's VTOC
wscanw	curs_scanw(3X)	convert formatted input from a curses widow
wscasecmp	wstring(3I)	Process Code string operations
wscat	wcstring(3I)	wide character string operations
wchr	wcstring(3I)	wide character string operations
wscmp	wcstring(3I)	wide character string operations
wscol	wstring(3I)	Process Code string operations
wscoll	wcscoll(3I)	wide character string comparison using collating information
wscopy	wcstring(3I)	wide character string operations
wscrl	curs_scroll(3X)	scroll a curses window
wscspn	wcstring(3I)	wide character string operations
wsdup	wstring(3I)	Process Code string operations
wsetscrreg	curs_outopts(3X)	curses terminal output option control routines

wslen	wcstring(3I)	wide character string operations
wncasecmp	wstring(3I)	Process Code string operations
wncat	wcstring(3I)	wide character string operations
wncmp	wcstring(3I)	wide character string operations
wncpy	wcstring(3I)	wide character string operations
wspbrk	wcstring(3I)	wide character string operations
wsprintf	wsprintf(3I)	formatted output conversion
wsrchr	wcstring(3I)	wide character string operations
wsscanf	wsscanf(3I)	formatted input conversion
wssp	wcstring(3I)	wide character string operations
wstandend	curs_attr(3X)	curses character and window attribute control routines
wstandout	curs_attr(3X)	curses character and window attribute control routines
wstod	wctod(3I)	convert wide character string to double-precision number
wstok	wcstring(3I)	wide character string operations
wstol	wctol(3I)	convert wide character string to long integer
wstring	wstring(3I)	Process Code string operations
wsxfrm	wcsxfrm(3I)	wide character string transformation
wsyncdown	curs_window(3X)	create curses windows
wsyncup	curs_window(3X)	create curses windows
wtimeout	curs_inopts(3X)	curses terminal input option control routines
wtouchln	curs_touch(3X)	curses refresh control routines
wvline	curs_border(3X)	create curses borders, horizontal and vertical lines
xdr	xdr(3N)	library routines for external data representation

xdr_accepted_reply	rpc_xdr(3N)	XDR library routines for remote procedure calls
xdr_admin	xdr_admin(3N)	library routines for external data representation
xdr_array	xdr_complex(3N)	library routines for external data representation
xdr_authsys_parms	rpc_xdr(3N)	XDR library routines for remote procedure calls
xdr_authunix_parms	rpc_soc(3N)	obsolete library routines for RPC
xdr_bool	xdr_simple(3N)	library routines for external data representation
xdr_bytes	xdr_complex(3N)	library routines for external data representation
xdr_callhdr	rpc_xdr(3N)	XDR library routines for remote procedure calls
xdr_callmsg	rpc_xdr(3N)	XDR library routines for remote procedure calls
xdr_char	xdr_simple(3N)	library routines for external data representation
xdr_complex	xdr_complex(3N)	library routines for external data representation
xdr_control	xdr_admin(3N)	library routines for external data representation
xdr_create	xdr_create(3N)	library routines for external data representation
xdr_destroy	xdr_create(3N)	stream creation library routines for external data representation
xdr_double	xdr_simple(3N)	stream creation library routines for external data representation
xdr_enum	xdr_simple(3N)	library routines for external data representation
xdr_float	xdr_simple(3N)	library routines for external data representation
xdr_free	xdr_simple(3N)	library routines for external data representation
xdr_getpos	xdr_admin(3N)	library routines for external data representation
xdr_hyper	xdr_simple(3N)	library routines for external data representation
xdr_inline	xdr_admin(3N)	library routines for external data representation

xdr_int	xdr_simple(3N)	library routines for external data representation
xdr_long	xdr_simple(3N)	library routines for external data representation
xdr_longlong_t	xdr_simple(3N)	library routines for external data representation
xdr_opaque	xdr_complex(3N)	library routines for external data representation
xdr_opaque_auth	rpc_xdr(3N)	XDR library routines for remote procedure calls
xdr_pointer	xdr_complex(3N)	library routines for external data representation
xdr_quadruple	xdr_simple(3N)	library routines for external data representation
xdr_reference	xdr_complex(3N)	library routines for external data representation
xdr_rejected_reply	rpc_xdr(3N)	XDR library routines for remote procedure calls
xdr_replymsg	rpc_xdr(3N)	XDR library routines for remote procedure calls
xdr_setpos	xdr_admin(3N)	library routines for external data representation
xdr_short	xdr_simple(3N)	library routines for external data representation
xdr_simple	xdr_simple(3N)	library routines for external data representation
xdr_sizeof	xdr_admin(3N)	library routines for external data representation
xdr_string	xdr_complex(3N)	library routines for external data representation
xdr_u_char	xdr_simple(3N)	library routines for external data representation
xdr_u_hyper	xdr_simple(3N)	library routines for external data representation
xdr_u_int	xdr_simple(3N)	library routines for external data representation
xdr_u_long	xdr_simple(3N)	library routines for external data representation
xdr_u_longlong_t	xdr_simple(3N)	library routines for external data representation
xdr_u_short	xdr_simple(3N)	library routines for external data representation
xdr_union	xdr_complex(3N)	library routines for external data representation

xdr_vector	xdr_complex(3N)	library routines for external data representation
xdr_void	xdr_simple(3N)	library routines for external data representation
xdr_wrapstring	xdr_complex(3N)	library routines for external data representation
xdrmem_create	xdr_create(3N)	library routines for external data representation stream creation
xdrrec_create	xdr_create(3N)	library routines for external data representation stream creation
xdrrec_endofrecord	xdr_admin(3N)	library routines for external data representation
xdrrec_eof	xdr_admin(3N)	library routines for external data representation
xdrrec_readbytes	xdr_admin(3N)	library routines for external data representation
xdrrec_skiprecord	xdr_admin(3N)	library routines for external data representation
xdrstdio_create	xdr_create(3N)	library routines for external data representation stream creation
xprt_register	rpc_svc_reg(3N)	library routines for registering servers
xprt_unregister	rpc_svc_reg(3N)	library routines for registering servers
y0	bessel(3M)	Bessel functions
y1	bessel(3M)	Bessel functions
yn	bessel(3M)	Bessel functions
yp_all	ypclnt(3N)	NIS Version 2 client interface
yp_bind	ypclnt(3N)	NIS Version 2 client interface
yp_first	ypclnt(3N)	NIS Version 2 client interface
yp_get_default_domain	ypclnt(3N)	NIS Version 2 client interface
yp_master	ypclnt(3N)	NIS Version 2 client interface
yp_match	ypclnt(3N)	NIS Version 2 client interface
yp_next	ypclnt(3N)	NIS Version 2 client interface

yp_order	ypclnt(3N)	NIS Version 2 client interface
yp_unbind	ypclnt(3N)	NIS Version 2 client interface
yp_update	yp_update(3N)	change NIS information
ypclnt	ypclnt(3N)	NIS Version 2 client interface
yperr_string	ypclnt(3N)	NIS Version 2 client interface
ypprot_err	ypclnt(3N)	NIS Version 2 client interface

NAME	a64l, l64a – convert between long integer and base-64 ASCII string
SYNOPSIS	<pre>#include <stdlib.h> long a64l(const char *s); char *l64a(long l);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>These functions are used to maintain numbers stored in base-64 ASCII characters. These characters 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.</p> <p>The characters used to represent “digits” are . for 0, / for 1, 0 through 9 for 2–11, A through Z for 12–37, and a through z for 38–63.</p> <p>a64l() takes a pointer to a null-terminated base-64 representation and returns a corresponding long value. If the string pointed to by <i>s</i> contains more than six characters, a64l() will use the first six.</p> <p>a64l() 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.</p> <p>l64a() 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.</p>
NOTES	<p>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.</p>

NAME	abort – terminate the process abnormally
SYNOPSIS	#include <stdlib.h> void abort(void);
MT-LEVEL	Safe
DESCRIPTION	abort() 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.
SEE ALSO	exit(2) , getrlimit(2) , kill(2) , wait(2) , waitpid(2) , fclose(3S) , raise(3C) , signal(3C)
NOTES	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.

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>
MT-LEVEL	MT-Safe
DESCRIPTION	abs() returns the absolute value of its int operand. labs() returns the absolute value of its long operand. llabs() returns the absolute value of its long long operand.
NOTES	In 2's-complement representation, the absolute value of the largest magnitude negative integral value is undefined.

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> int accept(int <i>s</i>, struct sockaddr *<i>addr</i>, int *<i>addrlen</i>);</pre>								
MT-LEVEL	Safe								
DESCRIPTION	<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). accept() 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. accept() 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><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>accept() is used with connection-based socket types, currently with SOCK_STREAM. 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	accept() 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> <tr> <td>ENODEV</td> <td>The protocol family and type corresponding to <i>s</i> could not be found in the netconfig file.</td> </tr> <tr> <td>ENOMEM</td> <td>There was insufficient user memory available to complete the operation.</td> </tr> </table>	EBADF	The descriptor is invalid.	EINTR	The accept attempt was interrupted by the delivery of a signal.	ENODEV	The protocol family and type corresponding to <i>s</i> could not be found in the netconfig file.	ENOMEM	There was insufficient user memory available to complete the operation.
EBADF	The descriptor is invalid.								
EINTR	The accept attempt was interrupted by the delivery of a signal.								
ENODEV	The protocol family and type corresponding to <i>s</i> could not be found in the netconfig 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 SOCK_STREAM .
EPROTO	A protocol error has occurred; for example, the STREAMS protocol stack has not been initialized.
EWouldBLOCK	The socket is marked as non-blocking and no connections are present to be accepted.

SEE ALSO **poll(2), bind(3N), connect(3N), listen(3N), select(3C), socket(3N), netconfig(4)**

NAME	addsev – define additional severities
SYNOPSIS	int addsev(int <i>int_val</i>, const char *<i>string</i>);
MT-LEVEL	MT-safe
DESCRIPTION	<p>The function addsev() defines additional severities for use in subsequent calls to pfmt() or lfmt(). addsev() associates an integer value <i>int_val</i> in the range [5-255] with a character <i>string</i>. It overwrites any previous string association with <i>int_val</i> and <i>string</i>.</p> <p>If <i>int_val</i> is ORed with the <i>flags</i> passed to subsequent calls pfmt() or lfmt(), <i>string</i> will be used as severity.</p> <p>Passing a NULL <i>string</i> removes the severity.</p> <p>Add-on severities are only effective within the applications defining them.</p>
RETURN VALUE	addsev() returns 0 in case of success, -1 otherwise.
USAGE	Only the standard severities are automatically displayed per the locale in effect at run-time. An application must provide the means for displaying locale-specific versions of add-on severities.
EXAMPLE	<pre>#define Panic 5 setlabel("APPL"); setcat("my_app!"); addsev(Panic, gettxt(":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: APPL: PANIC: Cannot locate database</p>
SEE ALSO	gettext(3C), lfmt(3C), pfmt(3C).

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>										
MT-LEVEL	Safe										
DESCRIPTION	<p>The <code>addseverity()</code> function builds a list of severity levels for an application to be used with the message formatting facility, <code>fmtmsg()</code>. <i>severity</i> is an integer value indicating the seriousness of the condition, and <i>string</i> is a pointer to a string describing the condition (string is not limited to a specific size).</p> <p>If <code>addseverity()</code> 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 <code>addseverity()</code> 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 <code>addseverity()</code> is called with a negative number or an integer value of 0, 1, 2, 3, or 4, the function fails and returns -1. 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"> <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 HALT.</td> </tr> <tr> <td><code>MM_ERROR</code></td> <td>Indicates that the application has detected a fault. Produces the print string ERROR.</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 WARNING.</td> </tr> <tr> <td><code>MM_INFO</code></td> <td>Provides information about a condition that is not in error. Produces the print string INFO.</td> </tr> <tr> <td><code>MM_NOSEV</code></td> <td>Indicates that no severity level is supplied for the message.</td> </tr> </table> <p>Severity levels may also be defined at run time using the <code>SEV_LEVEL</code> environment variable (see <code>fmtmsg(3C)</code>).</p>	<code>MM_HALT</code>	Indicates that the application has encountered a severe fault and is halting. Produces the print string HALT .	<code>MM_ERROR</code>	Indicates that the application has detected a fault. Produces the print string ERROR .	<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 WARNING .	<code>MM_INFO</code>	Provides information about a condition that is not in error. Produces the print string INFO .	<code>MM_NOSEV</code>	Indicates that no severity level is supplied for the message.
<code>MM_HALT</code>	Indicates that the application has encountered a severe fault and is halting. Produces the print string HALT .										
<code>MM_ERROR</code>	Indicates that the application has detected a fault. Produces the print string ERROR .										
<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 WARNING .										
<code>MM_INFO</code>	Provides information about a condition that is not in error. Produces the print string INFO .										
<code>MM_NOSEV</code>	Indicates that no severity level is supplied for the message.										
EXAMPLES	<p>When the function <code>addseverity()</code> is used as follows:</p> <pre>addseverity(7,"ALERT")</pre> <p>the following call to <code>fmtmsg()</code>:</p> <pre>fmtmsg(MM_PRINT, "UX:cat", 7, "invalid syntax", "refer to manual", "UX:cat:001")</pre> <p>produces:</p> <pre>UX:cat: ALERT: invalid syntax TO FIX: refer to manual UX:cat:001</pre>										

RETURN VALUES

addseverity() returns **MM_OK** on success or **MM_NOTOK** on failure.

SEE ALSO

fmtmsg(1), **fmtmsg(3C)**, **gettext(3C)**, **printf(3S)**

NAME	aio_cancel – cancel asynchronous I/O request
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lposix4 [<i>library</i> ...] #include <aio.h> int aio_cancel(int <i>fildes</i>, struct aiocb *<i>aiocb</i>);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>aio_cancel() attempts to cancel either one or all outstanding asynchronous I/O requests pending on the file descriptor specified by <i>fildes</i>. If <i>aiocb</i> is NULL, then all such outstanding cancelable requests are canceled; otherwise, the individual request referenced by <i>aiocb</i> references will be canceled.</p> <p>Normal completion notification occurs even for asynchronous I/O operations that are successfully canceled. If there are requests which cannot be canceled, then the normal asynchronous completion process takes place for those requests, and their associated aiocb structures are not modified.</p> <pre>struct aiocb { int aio_fildes; /* file descriptor */ volatile void *aio_buf; /* buffer location */ size_t aio_nbytes; /* length of transfer */ off_t aio_offset; /* file offset */ int aio_reqprio; /* request priority offset */ struct sigevent aio_sigevent; /* signal number and offset */ int aio_lio_opcode; /* listio operation */ }; struct sigevent { int sigev_notify; /* notification mode */ int sigev_signo; /* signal number */ union sigval sigev_value; /* signal value */ }; union sigval { int sival_int; /* integer value */ void *sival_ptr; /* pointer value */ };</pre>
RETURN VALUES	<p>If the requested operation(s) were canceled, aio_cancel() returns AIO_CANCELED. But if at least one of the requested operation(s) cannot be canceled because it is in progress, then AIO_NOTCANCELED is returned, and the application may determine the state of affairs for these operation(s) by using aio_error(3R). If all of the operation(s) had already completed, AIO_ALLDONE is returned. Otherwise, aio_cancel() returns -1, and sets errno to indicate the error condition.</p>

ERRORS**EBADF** *fdes* is not a valid file descriptor.**ENOSYS** **aio_cancel()** is not supported by this implementation.**SEE ALSO****aio_return(3R)**, **aio_read(3R)****BUGS**

In Solaris 2.4, these functions always return **-1** and set **errno** to **ENOSYS**, because this release does not support the Asynchronous Input and Output option. It is our intention to provide support for these interfaces in future releases.

NAME	aio_fsync – asynchronous file synchronization
SYNOPSIS	<pre>cc [flag ...] file ... -lposix4 [library ...] #include <aio.h> int aio_fsync(int op, aiocb *aiocb);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>aio_fsync() queues an asynchronous fsync(3C) or fdatasync(3R) request for all the currently queued I/O operations on the file referenced by <i>aiocb->aio_fildes</i>, and returns control immediately. This request is serviced concurrently with other activity of the process. If <i>op</i> is O_DSYNC, all I/O operations are completed by a call to fdatasync(3R) (synchronized I/O data integrity completion). If <i>op</i> is O_SYNC, all I/O operations are completed by a call to fsync(3C) (synchronized I/O file integrity completion). (see fcntl(5) definitions of O_DSYNC and O_SYNC.)</p> <p>When the request is queued, the error status for the operation is EINPROGRESS. When all data has been successfully transferred, the error status is reset to reflect the success or failure of the operation. aio_return(3R) and aio_error(3R) may be used with this <i>aiocb</i> value to monitor both the return and the error status of the asynchronous operation while it is proceeding.</p> <p><i>aiocb->aio_sigevent</i> defines the signal to be generated upon I/O completion. If <i>aiocb->aio_sigevent.sigev_signo</i> is non-zero, then a signal will be generated when all I/O operations have achieved synchronized I/O completion.</p> <pre>struct aiocb { int aio_fildes; /* file descriptor */ volatile void *aio_buf; /* buffer location */ size_t aio_nbytes; /* length of transfer */ off_t aio_offset; /* file offset */ int aio_reqprio; /* request priority offset */ struct sigevent aio_sigevent; /* signal number and offset */ int aio_lio_opcode; /* listio operation */ }; struct sigevent { int sigev_notify; /* notification mode */ int sigev_signo; /* signal number */ union signal sigev_value; /* signal value */ }; union signal { int sival_int; /* integer value */ void *sival_ptr; /* pointer value */ };</pre>

RETURN VALUES	If the I/O operation is successfully queued, aio_fsync() returns 0 . Otherwise, it returns -1 , and sets errno to indicate the error condition.	
ERRORS	EAGAIN	The requested asynchronous operation was not queued due to temporary resource limitations.
	EBADF	<i>aiocbp</i> -> aio_fildes is not a valid file descriptor open for writing.
	EINVAL	This implementation does not support synchronized I/O for this file. A value of <i>op</i> other than O_DSYNC or O_SYNC was specified.
	ENOSYS	aio_fsync() is not supported by this implementation.
SEE ALSO	fcntl(2) , open(2) , read(2) , write(2) , fsync(3C) , aio_error(3R) , aio_return(3R) , fdatasync(3R) , fcntl(5)	
NOTES	If aio_fsync() fails, outstanding I/O operations are not guaranteed to have been completed.	
BUGS	In Solaris 2.4, these functions always return -1 and set errno to ENOSYS , because this release does not support the Asynchronous Input and Output option. It is our intention to provide support for these interfaces in future releases.	

NAME	aio_read, aio_write – asynchronous read and write operations
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lposix4 [<i>library</i> ...] #include <aio.h> int aio_read(struct aiocb *aiocbp); int aio_write(struct aiocb *aiocbp); struct aiocb { int aio_fildes; /* file descriptor */ volatile void *aio_buf; /* buffer location */ size_t aio_nbytes; /* length of transfer */ off_t aio_offset; /* file offset */ int aio_reqprio; /* request priority offset */ struct sigevent aio_sigevent; /* signal number and offset */ int aio_lio_opcode; /* listio operation */ }; struct sigevent { int sigev_notify; /* notification mode */ int sigev_signo; /* signal number */ union sigval sigev_value; /* signal value */ }; union sigval { int sival_int; /* integer value */ void *sival_ptr; /* pointer value */ };</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>aio_read() queues an asynchronous read request, and returns control immediately. Rather than blocking until completion, the read operation continues concurrently with other activity of the process.</p> <p>Upon enqueueing the request, the calling process reads <i>aio</i><i>cbp</i>->nbytes from the file referred to by <i>aio</i><i>cbp</i>->fildes into the buffer pointed to by <i>aio</i><i>cbp</i>->aio_buf. <i>aio</i><i>cbp</i>->offset marks the absolute position from the beginning of the file (in bytes) at which the read begins.</p> <p>aio_write() queues an asynchronous write request, and returns control immediately. Rather than blocking until completion, the write operation continues concurrently with other activity of the process.</p> <p>Upon enqueueing the request, the calling process writes <i>aio</i><i>cbp</i>->nbytes from the buffer pointed to by <i>aio</i><i>cbp</i>->aio_buf into the file referred to by <i>aio</i><i>cbp</i>->fildes. If O_APPEND is set for <i>aio</i><i>cbp</i>->fildes, aio_write() operations append to the file in the same order as the calls were made.</p>

If **O_APPEND** is not set for the file descriptor, then the write operation will occur at the absolute position from the beginning of the file plus *aiocbp->offset* (in bytes).

These asynchronous operations are submitted at a priority equal to the calling process' scheduling priority minus *aiocbp->aio_reqprio*.

aiocb->aio_sigevent defines both the signal to be generated and how the calling process will be notified upon I/O completion. If *aio_sigevent.sigev_notify* 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 *aio_sigevent.sigev_notify* is **SIGEV_SIGNAL**, then the signal specified in *aio_sigevent.sigev_signo* 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 *aio_sigevent.sigev_value* will be the *si_value* component of the generated signal (see **siginfo(5)**).

RETURN VALUES

If the I/O operation is successfully queued, **aio_read()** and **aio_write()** return **0**, otherwise, they return **-1**, and set **errno** to indicate the error condition. *aiocbp* may be used as an argument to **aio_error(3R)** and **aio_return(3R)** in order to determine the error status and the return status of the asynchronous operation while it is proceeding.

ERRORS

- | | |
|------------------|--|
| EAGAIN | The requested asynchronous I/O operation was not queued due to system resource limitations. |
| ENOSYS | aio_read() or is not supported by this implementation. |
| EBADF | If the calling function is aio_read() , and <i>aiocbp->fildev</i> is not a valid file descriptor open for reading. If the calling function is aio_write() , and <i>aiocbp->fildev</i> is not a valid file descriptor open for writing. |
| EINVAL | <ul style="list-style-type: none"> • The file offset value implied by <i>aiocbp->aio_offset</i> would be invalid, • <i>aiocbp->aio_reqprio</i> is not a valid value, or • <i>aiocbp->aio_nbytes</i> is an invalid value. |
| 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 <i>aiocbp->aio_offset</i> would be invalid. |

SEE ALSO

close(2), **exec(2)**, **exit(2)**, **fork(2)**, **lseek(2)**, **read(2)**, **write(2)**, **aio_cancel(3R)**, **aio_return(3R)**, **aio_write(3R)**, **lio_listio(3R)**, **siginfo(5)**

NOTES

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

BUGS

In Solaris 2.4, these functions always return **-1** and set **errno** to **ENOSYS**, because this release does not support the Asynchronous Input and Output option. It is our intention to provide support for these interfaces in future releases.

NAME	aio_return, aio_error – retrieve return or error status of asynchronous I/O operation
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lposix4 [<i>library</i> ...] #include <aio.h> ssize_t aio_return(struct aiocb * aiocbp); int aio_error(const struct aiocb * aiocbp); struct aiocb { int aio_fildes; /* file descriptor */ volatile void *aio_buf; /* buffer location */ size_t aio_nbytes; /* length of transfer */ off_t aio_offset; /* file offset */ int aio_reqprio; /* request priority offset */ struct sigevent aio_sigevent; /* signal number and offset */ int aio_lio_opcode; /* listio operation */ }; struct sigevent { int sigev_notify; /* notification mode */ int sigev_signo; /* signal number */ union sigval sigev_value; /* signal value */ }; union sigval { int sival_int; /* integer value */ void *sival_ptr; /* pointer value */ };</pre>
MT-LEVEL	Async-Safe
DESCRIPTION	<p>aio_return() returns the return status of the asynchronous I/O request associated with the aiocb structure pointed to by <i>aiocbp</i>.</p> <p>aio_error() returns the error status of the asynchronous I/O request associated with the aiocb structure pointed to by <i>aiocbp</i>.</p> <p>aio_return() should be called only once to retrieve the valid return status of a given asynchronous operation, after aio_error() has returned a value other than EINPROGRESS.</p>
RETURN VALUES	<p>If the asynchronous I/O operation has completed successfully, aio_return() returns the return status, as described for read(2), write(2), and fsync(3C).</p> <p>If the asynchronous I/O operation has completed successfully, aio_error() returns 0. If the operation has not yet completed, then EINPROGRESS is returned. If the asynchronous I/O operation has completed unsuccessfully, then the error status, as described for read(2), write(2), and fsync(3C) is returned.</p>

If unsuccessful, `aio_return()` or `aio_error()` return `-1`, and set `errno` to indicate the error condition.

ERRORS

EINVAL `aiochp` does not reference an asynchronous operation which has completed or failed.

ENOSYS `aio_return()` or `aio_error()` is not supported by this implementation.

EXAMPLES

```
#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);
...
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 */
        }
    }
}
```

SEE ALSO

`close(2)`, `exec(2)`, `exit(2)`, `fork(2)`, `lseek(2)`, `read(2)`, `write(2)`, `fsync(3C)`, `aio_cancel(3R)`, `aio_fsync(3R)`, `aio_read(3R)`, `lio_listio(3R)`

BUGS

In Solaris 2.4, these functions always return **-1** and set **errno** to **ENOSYS**, because this release does not support the Asynchronous Input and Output option. It is our intention to provide support for these interfaces in future releases.

NAME	aio_suspend – wait for asynchronous I/O request
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lposix4 [<i>library</i> ...] #include <aio.h> int aio_suspend(const struct aiocb * const <i>list</i>[], int <i>nent</i>, const struct timespec *<i>timeout</i>);</pre>
MT-LEVEL	Async-Safe
DESCRIPTION	<p>aio_suspend() suspends the caller until at least one of the asynchronous I/O operations referenced by <i>list</i> has completed, until a signal interrupts the function, or, if <i>timeout</i> is not NULL, until the time interval specified by <i>timeout</i> has passed. If any of the aiocb structures in the list corresponds to a completed asynchronous I/O operation (i.e., the error status for the operation is not equal to EINPROGRESS), at the time of the call, the function returns without suspending the caller.</p> <p>If the time interval indicated in the timespec 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> <p><i>list</i> is an array of pointers to asynchronous I/O control blocks. <i>nent</i> indicates the number of elements in this array. Each aiocb structure pointed to must have been used in initiating an asynchronous I/O request via aio_read(3R), aio_write(3R), aio_fsync(3R), or lio_listio(3R). This array may contain NULL pointers which will be ignored.</p> <pre>struct aiocb { int aio_fildes; /* file descriptor */ volatile void *aio_buf; /* buffer location */ size_t aio_nbytes; /* length of transfer */ off_t aio_offset; /* file offset */ int aio_reqprio; /* request priority offset */ struct sigevent aio_sigevent; /* signal number and offset */ int aio_lio_opcode; /* listio operation */ }; struct sigevent { int sigev_notify; /* notification mode */ int sigev_signo; /* signal number */ union sigval sigev_value; /* signal value */ }; union sigval { int sival_int; /* integer value */ void *sival_ptr; /* pointer value */ };</pre>

```

struct timespec {
  time_t          tv_sec;          /* seconds */
  long           tv_nsec;        /* and nanoseconds */
};

```

RETURN VALUES

If **aio_suspend()** returns after one or more asynchronous I/O operations have completed, it returns **0**. Otherwise, it returns **-1**, and sets **errno** to indicate the error condition. The application may determine which asynchronous I/O had completed with both the associated error and return status of **aio_return(3R)**, and **aio_error(3R)**.

ERRORS

EAGAIN No asynchronous I/O indicated in the list referenced by *list* completed in the time interval indicated by *timeout*.

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.

ENOSYS **aio_suspend()** is not supported by this implementation.

SEE ALSO

aio_fsync(3R), **aio_read(3R)**, **aio_return(3R)**, **aio_write(3R)**, **lio_listio(3R)**

BUGS

In Solaris 2.4, these functions always return **-1** and set **errno** to **ENOSYS**, because this release does not support the Asynchronous Input and Output option. It is our intention to provide support for these interfaces in future releases.

NAME	aiocancel – cancel an asynchronous operation
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -l<code>aiocancel</code> [<i>library</i> ...] <code>#include <sys/asynch.h></code> <code>int aiocancel(aio_result_t *resultp);</code></pre>
MT-LEVEL	Unsafe
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	<p>aiocancel() returns:</p> <p>0 on success.</p> <p>-1 on failure and sets errno to indicate the error.</p>
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 below.</p> <p>EINVAL There are not any outstanding requests to cancel.</p>
SEE ALSO	aioread(3) , aiowait(3)
NOTES	Passing an illegal address as <i>resultp</i> will result in setting errno to EFAULT <i>only</i> if it is detected by the application process.

NAME	aioread, aiowrite – asynchronous I/O operations
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -laio [<i>library</i> ...] #include <sys/asynch.h> int aioread(int <i>fildev</i>, char *<i>bufp</i>, int <i>buflen</i>, off_t <i>offset</i>, int <i>whence</i>, aio_result_t *<i>resultp</i>); int aiowrite(int <i>fildev</i>, const char *<i>bufp</i>, int <i>buflen</i>, off_t <i>offset</i>, int <i>whence</i>, aio_result_t *<i>resultp</i>);</pre>
MT-LEVEL	Unsafe
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>buflen</i> bytes of data from the object referenced by the descriptor <i>fildev</i> 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>buflen</i> bytes of data from the buffer pointed to by <i>bufp</i> to the object referenced by the descriptor <i>fildev</i>.</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> <p>Upon completion of the operation both <i>aio_return</i> and <i>aio_errno</i> 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 <i>aio_return</i> to this value.</p> <p>The application supplied buffer <i>bufp</i> should not be referenced by the application until after the operation has completed. While the operation is <i>in progress</i>, this buffer is in use by the operating system.</p> <p>Notification of the completion of an asynchronous I/O operation may be obtained synchronously through the aiowait(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 aiowait(3) returns nothing (using a polling timeout) before returning from the signal handler, no asynchronous I/O notifications are lost. The aiowait(3) function is the only</p>

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

aioread() and **aiowrite()** return:

0 on success.

-1 on failure and set **errno** to indicate the error.

ERRORS

EAGAIN The number of asynchronous requests that the system can handle at any one time has been exceeded

EBADF *fdes* is not a valid file descriptor open for reading.

EFAULT At least one of *bufp* points to an address outside the address space of the requesting process. See **NOTES** below.

EINVAL The parameter *resultp* is currently being used by an outstanding asynchronous request.

ENOMEM Memory resources are unavailable to initiate request.

SEE ALSO

close(2), **exec(2)**, **exit(2)**, **lseek(2)**, **open(2)**, **read(2)**, **write(2)**, **aiocancel(3)**, **aiowait(3)**, **sigvec(3B)**

NOTES

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

NAME	aiowait – wait for completion of asynchronous I/O operation
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -laiio [<i>library</i> ...] #include <sys/asynch.h> #include <sys/time.h> aiio_result_t *aiowait(const struct timeval *<i>timeout</i>);</pre>
MT-LEVEL	Unsafe
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 <sys/time.h> and contains the following members:</p> <pre> long tv_sec; /* seconds */ long tv_usec; /* and microseconds */</pre>
RETURN VALUES	On success, aiowait() returns a pointer to the result structure used when the completed asynchronous I/O operation was requested. On failure, it returns -1 and sets errno to indicate the error. aiowait() returns 0 if the time limit expires.
ERRORS	<p>EFAULT <i>timeout</i> points to an address outside the address space of the requesting process. See NOTES below.</p> <p>EINTR A signal was delivered before an asynchronous I/O operation completed. The time limit expired.</p> <p>EINVAL There are no outstanding asynchronous I/O requests.</p>
SEE ALSO	aiocancel(3) , aioread(3)
NOTES	<p>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.</p> <p>Passing an illegal address as <i>timeout</i> will result in setting errno to EFAULT <i>only</i> if it is detected by the application process.</p>

NAME	assert – verify program assertion
SYNOPSIS	#include <assert.h> void assert(int <i>expression</i>);
MT-LEVEL	Safe
DESCRIPTION	<p>This macro is useful for putting diagnostics into programs. When it is executed, if <i>expression</i> is false (zero), assert() prints</p> <p style="padding-left: 40px;">Assertion failed: <i>expression</i>, file <i>xyz</i>, line <i>nnn</i></p> <p>on the standard error output and aborts. In the error message, <i>xyz</i> is the name of the source file and <i>nnn</i> the source line number of the assert() statement. The latter are respectively the values of the preprocessor macros __FILE__ and __LINE__.</p> <p>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.</p>
SEE ALSO	cc(1B) , abort(3C) , gettext(3I) , setlocale(3C)
NOTES	<p>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).</p> <p>Since assert() is implemented as a macro, the <i>expression</i> may not contain any string literals.</p>

NAME	atexit – add program termination routine
SYNOPSIS	#include <stdlib.h> int atexit(void (*func)(void));
MT-LEVEL	Safe
DESCRIPTION	atexit() adds the function <i>func()</i> to 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() 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	atexit() returns 0 if the registration succeeds, nonzero if it fails.
SEE ALSO	exit(3C)

NAME	au_open, au_close, au_write – construct and write audit records
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -l<code>bsm</code> -l<code>socket</code> -l<code>nsi</code> -l<code>intl</code> [<i>library</i> ...] #include <bsm/libbsm.h> int au_close(int <i>d</i>, int <i>keep</i>, short <i>event</i>); int au_open(void); int au_write(int <i>d</i>, token_t *<i>m</i>);</pre>
MT-LEVEL	Safe.
AVAILABILITY	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.
DESCRIPTION	<p>au_open() allocates an audit record descriptor to which audit tokens can be written using au_write().</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 for all calls will return a 0.</p> <p>au_open() returns -1 if a descriptor could not be allocated. au_close() and au_write() return -1 if <i>d</i> is not a valid descriptor or if audit(2) experienced an error. In the latter case, errno is set to indicate the error.</p>
SEE ALSO	bsmconv(1M) , audit(2) , auditon(2) , au_preselect(3) , au_to(3) , free(3C)

NAME	au_preselect – preselect an audit event
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -l<code>bsm</code> -l<code>socket</code> -l<code>insl</code> -l<code>intl</code> [<i>library</i> ...] #include <bsm/libbsm.h> int au_preselect(audit_event_t <i>event</i>, audit_mask_t *<i>mask_p</i>, int <i>sorf</i>, int <i>flag</i>);</pre>
MT_LEVEL	MT-Safe.
AVAILABILITY	The functionality described in this man page is available only if the Basic Security Module (BSM) has been enabled. See <code>bsmconv(1M)</code> for more information.
DESCRIPTION	<p><code>au_preselect()</code> 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 <code>getaudit(2)</code>). <code>au_preselect()</code> looks up the classes associated with <i>event</i> in <code>audit_event(4)</code> 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> <p>AU_PRS_SUCCESS Compare the event class with the success portion of the preselection mask.</p> <p>AU_PRS_FAILURE Compare the event class with the failure portion of the preselection mask.</p> <p>AU_PRS_BOTH Compare the event class with both the success and failure portions of the preselection mask.</p> <p><i>flag</i> tells <code>au_preselect()</code> how to read the <code>audit_event(4)</code> database. Upon initial invocation, <code>au_preselect()</code> reads the <code>audit_event(4)</code> database and allocates space in an internal cache for each entry with <code>malloc(3C)</code>. In subsequent invocations, the value of <i>flag</i> determines where <code>au_preselect()</code> obtains audit event information. The following are the valid values of <i>flag</i>:</p> <p>AU_PRS_REREAD Get audit event information by searching the <code>audit_event(4)</code> database.</p> <p>AU_PRS_USECACHE Get audit event information from internal cache created upon the initial invocation. This option is much faster.</p>

RETURN VALUES**au_preselect()** returns:

- 0** *event* is not preselected.
- 1** *event* 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

SEE ALSO

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

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.

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> ... -l<code>bsm</code> -l<code>socket</code> -l<code>nsi</code> -l<code>intl</code> [<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 <code>vattr</code> *<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 <code>inaddr</code> *<i>internet_addr</i>); token_t *au_to_iport(u_short <i>iport</i>); token_t *au_to_ipc(int <i>id</i>); token_t *au_to_ipc_perm(struct <code>ipc_perm</code> *<i>perm</i>); token_t *au_to_iport(u_short <i>iport</i>); token_t *au_to_me(void); token_t *au_to_newgroups(int <i>n</i>, int *<i>groups</i>); token_t *au_to_opaque(char *<i>data</i>, short <i>bytes</i>); token_t *au_to_path(char *<i>path</i>); token_t *au_to_process(au_id_t <i>auaid</i>, uid_t <i>euid</i>, gid_t <i>egid</i>, uid_t <i>ruid</i>, gid_t <i>rgid</i>, au_asid_t <i>sid</i>, au_tid_t *<i>tid</i>); token_t *au_to_return(char <i>number</i>, u_int <i>value</i>); token_t *au_to_socket(struct <code>socket</code> *<i>so</i>); token_t *au_to_subject(au_id_t <i>auaid</i>, uid_t <i>euid</i>, gid_t <i>egid</i>, uid_t <i>ruid</i>, gid_t <i>rgid</i>, pid_t <i>pid</i>, au_asid_t <i>sid</i>, au_tid_t *<i>tid</i>); token_t *au_to_text(char *<i>text</i>);</pre>
MT_LEVEL	MT-Safe.
AVAILABILITY	The functionality described in this man page is available only if the Basic Security Module (BSM) has been enabled. See <code>bsmconv(1M)</code> for more information.
DESCRIPTION	<code>au_to_arg()</code> formats the data in <i>v</i> into an “argument token.” The <i>n</i> argument indicates the argument number. The <i>text</i> 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 “new-groups token.”

au_to_subject() formats an *auid* (audit user ID), an *euid* (effective user ID), an *egid* (effective group ID), a *ruid* (real user ID), an *rgid* (real group ID), a *pid* (process ID), an *sid* (audit session ID), an *tid* (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 *auid* (audit user ID), an *euid* (effective user ID), an *egid* (effective group ID), a *ruid* (real user ID), a *rgid* (real group ID), a *pid* (process ID), an *sid* (audit session ID), and a *tid* (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.

SEE ALSO | **bsmconv(1M), au_open(3)**

NAME	au_user_mask – get user's binary preselection mask
SYNOPSIS	<pre>cc [flag ...] file ... -lbsm -lsocket -lnsl -lintl [library ...] #include <bsm/libbsm.h> int au_user_mask(char *username, au_mask_t *mask_p);</pre>
MT-LEVEL	MT-Safe.
AVAILABILITY	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.
DESCRIPTION	<p>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 <i>mask_p</i> with the combined value.</p> <p>The audit flags in the <i>flags</i> field of the audit_control(4) database and the <i>always-audit-flags</i> and <i>never-audit-flags</i> from the audit_user(4) database represent binary audit classes. These fields are combined by au_preselect(3) as follows:</p> $\text{mask} = (\text{flags} + \text{always-audit-flags}) - \text{never-audit-flags}$ <p>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.</p>
RETURN VALUES	<p>au_user_mask() returns:</p> <ul style="list-style-type: none"> 0 Success. -1 Failure. Both the audit_control(4) and the audit_user(4) database entries could not be retrieved.
FILES	<p>/etc/security/audit_control contains default parameters read by the audit daemon, auditd(1M)</p> <p>/etc/security/audit_user stores per-user audit event mask</p>
SEE ALSO	login(1) , bsmconv(1M) , getaudit(2) , setaudit(2) , au_preselect(3) , getacinfo(3) , getausernam(3) , audit_control(4) , audit_user(4)
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.

NAME	basename – return the last element of a path name								
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lgen [<i>library</i> ...] #include <libgen.h> char *basename(char *<i>path</i>);</pre>								
MT-LEVEL	MT-Safe								
DESCRIPTION	<p>Given a pointer to a null-terminated character string that contains a path name, basename() returns a pointer to the last element of <i>path</i>. Trailing “/” characters are deleted.</p> <p>If <i>path</i> or *<i>path</i> is zero, pointer to a static constant “.” is returned.</p>								
EXAMPLES	<table border="0"> <thead> <tr> <th style="text-align: left;">Input string</th> <th style="text-align: left;">Output pointer</th> </tr> </thead> <tbody> <tr> <td>/usr/lib</td> <td>lib</td> </tr> <tr> <td>/usr/</td> <td>usr</td> </tr> <tr> <td>/</td> <td>/</td> </tr> </tbody> </table>	Input string	Output pointer	/usr/lib	lib	/usr/	usr	/	/
Input string	Output pointer								
/usr/lib	lib								
/usr/	usr								
/	/								
SEE ALSO	basename(1), dirname(3G)								
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	bessel, j0, j1, jn, y0, y1, yn – Bessel functions
SYNOPSIS	<pre>cc [flag ...] file ... -lm [library ...] #include <math.h> double j0(double x); double j1(double x); double jn(int n, double x); double y0(double x); double y1(double x); double yn(int n, double x);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	These functions calculate Bessel functions of the first and second kinds for real arguments and integer orders.
RETURN VALUES	For exceptional cases, matherr(3M) tabulates the values to be returned as dictated by various Standards.
SEE ALSO	exp(3M) , matherr(3M)
DIAGNOSTICS	In IEEE 754 mode (i.e. the -xlibmieee cc compilation option), the functions y0() , y1() , and yn() have logarithmic singularities at the origin, so they treat zero and negative arguments the way log() does, as described in exp(3M) . Such arguments are unexceptional for j0() , j1() , and jn() .

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>
MT-LEVEL	MT-Safe
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 style="padding-left: 40px;">bgets(buffer, sizeof buffer, stream, "\n");</pre> <p>is identical to</p> <pre style="padding-left: 40px;">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	<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>
SEE ALSO	gets(3S)
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> int bind(int <i>s</i>, const struct sockaddr *<i>name</i>, int <i>namelen</i>);</pre>
MT-LEVEL	Safe
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 errno .
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> <p>ELOOP Too many symbolic links were encountered in translating the pathname in <i>name</i>.</p> <p>ENOENT A component of the path prefix of the pathname in <i>name</i> does not exist.</p>

ENOTDIR A component of the path prefix of the pathname in *name* is not a directory.

EROFS The inode would reside on a read-only file system.

SEE ALSO **unlink(2)**, **socket(3N)**

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	bsdmalloc, malloc, free, realloc – memory allocator
SYNOPSIS	<pre>cc [<i>flag ...</i>] <i>file ...</i> -lbsdmalloc [<i>library ...</i>] char *malloc(<i>size</i>) unsigned size; int free(<i>ptr</i>) char * ptr; char *realloc(<i>ptr, size</i>) char *ptr; unsigned size;</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 sbrk(2) 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 DIAGNOSTICS).</p> <p>malloc() returns a pointer to a block of at least <i>size</i> 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 <i>ptr</i> to <i>size</i> 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>
ERRORS	<p>malloc(), free(), and realloc() will fail if one or more of the following are true:</p> <p>EINVAL An invalid argument was specified.</p> <p> The value of <i>ptr</i> passed to free() or realloc() was not a pointer to a block previously allocated by malloc(), or realloc().</p> <p> The allocation heap is found to have been corrupted.</p> <p>ENOMEM <i>size</i> bytes of memory could not be allocated.</p>
SEE ALSO	sbrk (2), alloca (3C), calloc (3C), free (3C), malloc (3C), memalign (3C), realloc (3C), valloc (3C), malloc (3X)
WARNINGS	<p>Use of libbsdmalloc renders an application non-SCD compliant.</p> <p>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).</p>

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.

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>
MT-LEVEL	Safe
DESCRIPTION	<p>bsearch() 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>. <i>key</i> points to a datum instance to be sought in the table. <i>base</i> points to the element at the base of the table. <i>nel</i> is the number of elements in the table. <i>size</i> is the number of bytes in each element. The function pointed to by <i>compar</i> is called with two arguments that point to the elements being compared. The function must return an integer less than, equal to, or greater than 0 as accordingly the first argument is to be considered less than, equal to, or greater than the second.</p>
RETURN VALUES	A null pointer is returned if the key cannot be found in the table.
EXAMPLES	<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> struct node { 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;</pre>

```

/* 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));
}

```

SEE ALSO [hsearch\(3C\)](#), [lsearch\(3C\)](#), [qsort\(3C\)](#), [tsearch\(3C\)](#)

NOTES The pointers to the key and the element at the base of the table should be of type pointer-to-*element*.

The comparison function need not compare every byte, so arbitrary data may be contained in the elements in addition to the values being compared.

If the number of elements in the table is less than the size reserved for the table, *nel* should be the lower number.

NAME	bstring, bcopy, bcmp, bzero – bit and byte string operations
SYNOPSIS	<pre>/usr/ucb/cc [<i>flag ...</i>] <i>file ...</i> void bcopy(<i>s1, s2, len</i>) register char *<i>s1, *s2</i>; int len; int bcmp(<i>s1, s2, len</i>) register char *<i>s1, *s2</i>; int len; void bzero(<i>sp, len</i>) register char *<i>sp</i>; int len;</pre>
DESCRIPTION	<p>The functions bcopy(), bcmp(), and bzero() operate on variable length strings of bytes. They do not check for null bytes as the routines in string(3C) do.</p> <p>bcopy() copies <i>len</i> bytes from string <i>s1</i> to the string <i>s2</i>. Overlapping strings are handled correctly.</p> <p>bcmp() compares byte string <i>s1</i> against byte string <i>s2</i>, returning zero if they are identical, 1 otherwise. Both strings are assumed to be <i>len</i> bytes long. bcmp() using <i>len</i> zero bytes always returns zero.</p> <p>bzero() places <i>len</i> 0 bytes in the string <i>sp</i>.</p>
WARNINGS	The bcmp() and bcopy() routines take parameters backwards from strcmp and strcpy , respectively. See string(3C) .
SEE ALSO	string(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	bufsplit – split buffer into fields
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lgen [<i>library</i> ...] #include <libgen.h> size_t bsplit(char *buf, size_t n, char **a);</pre>
MT-LEVEL	MT-Safe
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 new-lines.</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 ':', '.', and ',' as separators along with tab and new-line:</p> <pre> bsplit(":\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	<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 multi-thread applications, the _REENTRANT flag must be defined on the compile line. This flag should only be used in multi-thread applications.</p>

NAME	byteorder, htonl, htons, ntohl, ntohs – convert values between host and network byte order
SYNOPSIS	<pre>#include <sys/types.h> #include <netinet/in.h> ulong htonl(u_long hostlong); u_short htons(u_short hostshort); u_long ntohl(u_long netlong); u_short ntohs(u_short netshort);</pre>
MT-LEVEL	Safe
DESCRIPTION	<p>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.</p> <p>These routines are most often used in conjunction with Internet addresses and ports as returned by <code>gethostent()</code> and <code>getservent()</code>. (See <code>gethostbyname(3N)</code> and <code>getservbyname(3N)</code> respectively.)</p>
SEE ALSO	<code>gethostbyname(3N)</code> , <code>getservbyname(3N)</code>

NAME	catgets – read a program message
SYNOPSIS	<pre>#include <nl_types.h> char *catgets(nl_catd catd, int set_num, int msg_num, char *s);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	catgets() attempts to read message <i>msg_num</i> , in set <i>set_num</i> , from the message catalog identified by <i>catd</i> . <i>catd</i> is a catalog descriptor returned from an earlier call to catopen() . <i>s</i> 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> .
SEE ALSO	gencat(1) , catopen(3C) , setlocale(3C) , gettext(3I) <i>Developer's Guide to Internationalization</i>
NOTES	catgets can be used safely in a multi-thread application, as long as setlocale(3C) is not being called to change the locale.

NAME	catopen, catclose – open/close a message catalog
SYNOPSIS	<pre>#include <nl_types.h> nl_catd catopen(char *name, int oflag); int catclose(nl_catd catd);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>catopen() 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. 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 C locale, catopen() will always succeed without even looking at the implementation defined 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 style="padding-left: 40px;">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/name.cat, if <i>oflag</i> is 0, or in /nlslib/{LC_MESSAGES}/name.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 <i>territory</i> element of LANG or LC_MESSAGES. %c The value of the <i>codeset</i> 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 then the value of **LC_MESSAGES** as returned by **setlocale()** is used. If this is **NULL** then the default path as defined in **nl_types()** is used.

If the value of *offlag* argument is **0**, the **LANG** environment variable is used to locate the catalogue without regard to the **LC_MESSAGES** category. If the *offlag* argument is **NL_CAT_LOCALE**, the **LC_MESSAGES** category is used to locate the message catalogue. **catclose()** closes the message catalog identified by *catd*.

RETURN VALUES

If successful, **catopen()** returns a message catalog descriptor for use on subsequent calls to **catgets()** and **catclose()**; otherwise **catopen()** returns **(nl_catd) -1**.

catclose() returns **0** if successful, otherwise **-1**.

SEE ALSO

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

NOTES

catopen and **catclose** can be used safely in a multi-thread application, as long as **setlocale(3C)** is not being called to change the locale.

NAME	clock – report CPU time used
SYNOPSIS	#include <time.h> clock_t clock(void);
MT-LEVEL	MT-Safe
DESCRIPTION	clock() 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() function, the pclose() function, or the system() function. Dividing the value returned by clock() by the constant CLOCKS_PER_SEC , defined in the <time.h> header, will give the time in seconds.
SEE ALSO	times(2) , wait(2) , popen(3S) , system(3S)
NOTES	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). If the process time used is not available or cannot be represented, clock returns the value (clock_t) -1.

NAME	clock_settime, clock_gettime, clock_getres – high-resolution clock operations
SYNOPSIS	<pre>cc [flag ...] file ... -lposix4 [library ...] #include <time.h> int clock_settime(clockid_t clock_id, const struct timespec *tp); int clock_gettime(clockid_t clock_id, struct timespec *tp); int clock_getres(clockid_t clock_id, struct timespec *res); struct timespec { time_t tv_sec; /* seconds */ long tv_nsec; /* and nanoseconds */ };</pre>
MT-LEVEL	Async-Safe
DESCRIPTION	<p>clock_settime() sets the specified clock, <i>clock_id</i>, to the value specified by <i>tp</i>. The calling process must have an effective user ID of 0.</p> <p>clock_gettime() 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(). If <i>res</i> is not NULL, the resolution of the specified clock is stored in <i>res</i>.</p> <p>The <i>clock_id</i> for the real-time clock for the system is CLOCK_REALTIME. The values returned by clock_gettime() and specified by clock_settime() represent the amount of time (in seconds and nanoseconds) since 00:00 Universal Coordinated Time, January 1, 1970.</p>
RETURN VALUES	clock_settime() , clock_gettime() , and clock_getres() return 0 upon success, otherwise they return -1 and set errno to indicate the error condition.
ERRORS	<p>EINVAL <i>clock_id</i> does not specify a known clock.</p> <p> The <i>tp</i> argument to clock_settime() is outside the range for the given clock id.</p> <p> The <i>tp</i> argument to clock_settime() specified a nanosecond value less than zero or greater than or equal to 1,000,000,000.</p> <p>ENOSYS clock_settime(), clock_gettime(), or clock_getres() is not supported by this implementation.</p> <p>EPERM The requesting process does not have the appropriate privilege to set the specified clock.</p>
SEE ALSO	time(2) , ctime(3C) , timer_gettime(3R)

NOTES

Clock resolutions are implementation defined and are not settable by a process. 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.

NAME	condition, cond_init, cond_destroy, cond_wait, cond_timedwait, cond_signal, cond_broadcast – condition variables
SYNOPSIS	<pre>#include <synch.h> int cond_init(cond_t *cvp, int type, int arg); int cond_destroy(cond_t *cvp); 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);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>A condition variable enables threads to atomically block until a condition is satisfied. The condition is tested under the protection of a mutual exclusion lock (mutex). When the condition is false, a thread typically blocks on a condition variable and atomically releases the mutex waiting for the condition to change. When another thread changes the condition, it may signal the associated condition variable to cause one or more waiting threads to wake up, reacquire the mutex, and re-evaluate the condition.</p> <p>Condition variables can be used to synchronize threads among processes if they are allocated in memory that is writable and shared by the cooperating processes (see mmap(2)) and have been initialized for this behavior.</p> <p>Condition variables must be initialized before use. cond_init() initializes the condition variable pointed to by <i>cvp</i>. A condition variable can potentially have several different types of behavior, specified by <i>type</i>. No current type uses <i>arg</i> although a future type may specify additional behavior parameters via <i>arg</i>. <i>type</i> may be one of the following:</p> <p>USYNC_PROCESS The condition variable can be used to synchronize threads in this process and other processes. Only one process should initialize the condition variable. <i>arg</i> is ignored.</p> <p>USYNC_THREAD The condition variable can be used to synchronize threads in this process, only. <i>arg</i> is ignored.</p> <p>Condition variables may also be initialized by allocation in zeroed memory. In this case a <i>type</i> of USYNC_THREAD is assumed. Multiple threads must not initialize the same condition variable simultaneously. A condition variable must not be re-initialized while other threads may be using it.</p> <p>cond_destroy() destroys any state associated with the condition variable pointed to by <i>cvp</i>. The space for storing the condition variable is not freed.</p> <p>cond_wait() atomically releases the mutex pointed to by <i>mp</i> and causes the calling thread to block on the condition variable pointed to by <i>cvp</i>. The blocked thread may be awakened by cond_signal(), cond_broadcast(), or when interrupted by delivery of a signal or a fork(). Any change in value of a condition associated with the condition variable</p>

cannot be inferred by the return of **cond_wait()** and any such condition must be re-evaluated.

cond_timedwait() is similar to **cond_wait()**, except that the calling thread will not block past the time of day specified by *abstime*. If the time of day becomes greater than *abstime* then **cond_timedwait()** returns with the error code **ETIME**.

cond_wait() and **cond_timedwait()** always return with the mutex locked and owned by the calling thread even when returning an error.

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.

RETURN VALUES

Zero is returned when successful. A non-zero value indicates an error.

ERRORS

If any of the following conditions are detected, these functions fail and return the corresponding value:

EINVAL Invalid argument. For **cond_init()**, *type* is not a recognized type. For **cond_timedwait()**, the specified number of seconds, *abstime*, is greater than *pgm_start_time* + 50,000,000, where *pgm_start_time* is the start time of the application, or the number of nanoseconds is greater than or equal to 1,000,000,000.

EFAULT *cvp*, *arg*, or *abstime* point to an illegal address.

If any of the following conditions are detected, **cond_wait()** or **cond_timedwait()** fails and returns the corresponding value:

EINTR The wait was interrupted by a signal or **fork()**.

If any of the following conditions are detected, **cond_timedwait()** fails and returns the corresponding value:

ETIME The time specified by *abstime* has passed.

EXAMPLES

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);
```

cond_timedwait() is also normally used in a loop testing some condition. 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 == ETIME) {
        /* timeout, do something */
        break;
    }
}
(void) mutex_unlock(mp);
```

This sets a bound on the total wait time even though **cond_timedwait()** may return several times due to the condition being signaled or the wait being interrupted.

SEE ALSO

mmap(2), **fork(2)**, **signal(3C)**, **mutex(3T)**

NOTES

These interfaces also available via:

#include <thread.h>

By default, there is no defined order of unblocking if multiple threads are waiting for a condition variable.

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> int connect(int <i>s</i>, struct sockaddr *<i>name</i>, int <i>namelen</i>);</pre>														
MT-LEVEL	Safe														
DESCRIPTION	<p>The parameter <i>s</i> is a socket. If it is of type SOCK_DGRAM, 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 SOCK_STREAM, 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	If the connection or binding succeeds, 0 is returned. Otherwise, -1 is returned and sets errno to indicate the error.														
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 close(2) the socket descriptor, and issue another socket(3N) 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 close(2) the socket descriptor, and issue another socket(3N) 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 close(2) the socket descriptor, and issue another socket(3N) 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.
ETIMEDOUT	Connection establishment timed out without establishing a connection.
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	<i>name</i> is not a socket.
EPROTOTYPE	The file referred to by <i>name</i> is a socket of a type other than type <i>s</i> (for example, <i>s</i> is a SOCK_DGRAM socket, while <i>name</i> refers to a SOCK_STREAM socket).

SEE ALSO **close(2)**, **accept(3N)**, **connect(3N)**, **select(3C)**, **getsockname(3N)**, **socket(3N)**

NAME	conv, toupper, tolower, _toupper, _tolower, toascii – translate characters
SYNOPSIS	<pre>#include <ctype.h> int toupper(int c); int tolower(int c); int _toupper(int c); int _tolower(int c); int toascii(int c);</pre>
MT-LEVEL	MT-Safe with exceptions
DESCRIPTION	<p>toupper() and tolower() have as their domain the range of the function getc(): all values represented in an unsigned char and the value of the macro EOF as defined in stdio.h. If the argument of toupper() represents a lower-case letter, the result is the corresponding upper-case letter. If the argument of tolower() represents an upper-case letter, the result is the corresponding lower-case letter. All other arguments in the domain are returned unchanged.</p> <p>The macros _toupper() and _tolower() accomplish the same things as toupper() and tolower(), respectively, but have restricted domains and are faster. _toupper() requires a lower-case letter as its argument; its result is the corresponding upper-case letter. _tolower() requires an upper-case letter as its argument; its result is the corresponding lower-case letter. Arguments outside the domain cause undefined results.</p> <p>toascii() yields its argument with all bits turned off that are not part of a standard 7-bit ASCII character; it is intended for compatibility with other systems.</p> <p>toupper(), tolower(), _toupper(), and _tolower() are affected by LC_CTYPE. In the “C” locale, or in a locale where shift information is not defined, these functions determine the case of characters according to the rules of the ASCII-coded character set. Characters outside the ASCII range of characters are returned unchanged.</p>
SEE ALSO	ctype(3C) , setlocale(3C) , getc(3S) , environ(5)
NOTES	toupper , tolower , _toupper , _tolower and toascii can be used safely in a multi-thread application, as long as setlocale(3C) is not being called to change the locale.

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>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>copylist() 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. 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>
EXAMPLES	<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>
SEE ALSO	malloc(3C)
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	crypt, setkey, encrypt – generate encryption
SYNOPSIS	<pre>#include <crypt.h> char *crypt(const char *key, const char *salt); void setkey(const char *key); void encrypt(char *block, int edflag);</pre>
MT-LEVEL	Safe
DESCRIPTION	<p>crypt() is the password encryption function. 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><i>key</i> is the input string to encrypt, for instance, a user's typed password. Only the first eight characters are used; the rest are ignored. <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 input string is used as the key to repeatedly encrypt a constant string. The returned value points to the encrypted input string. The first two characters of the return value are the <i>salt</i> itself.</p> <p>The setkey() and encrypt() functions provide (rather primitive) access to the actual hashing algorithm. The argument of setkey() is a character array of length 64 containing only the characters with numerical value 0 and 1. This string is divided into groups of 8, the low-order bit in each group is ignored; this gives a 56-bit key that is set into the machine. This is the key that will be used with the hashing algorithm to encrypt the string <i>block</i> with the encrypt() function.</p> <p>The <i>block</i> argument of encrypt() 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 setkey(). The argument <i>edflag</i>, indicating decryption rather than encryption, is ignored; use encrypt() in libcrypt() (see crypt(3X)) for decryption.</p>
RETURN VALUES	If <i>edflag</i> is set to anything other than zero, errno will be set to ENOSYS .
SEE ALSO	login(1) , passwd(1) , crypt(3X) , getpass(3C) , passwd(4)
NOTES	The return value for crypt() points to static data that are overwritten by each call. In the case of multithreaded applications, the return value is a pointer to thread specific data.

NAME	cset, csetlen, csetcol, csetno, wsetno – get information on EUC codesets
SYNOPSIS	<pre>#include <euc.h> int csetlen(int codeset); int csetcol(int codeset); int csetno(unsigned char c); #include <widec.h> int wsetno(wchar_t pc);</pre>
MT-LEVEL	MT-Safe with exceptions
DESCRIPTION	<p>Both csetlen() and csetcol() take a code set number <i>codeset</i>, which must be 0, 1, 2, or 3. csetlen() 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. csetcol() returns the number of columns a character in the given EUC code set would take on the display.</p> <p>csetno() is 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>wsetno() is 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<widec.h> ... x+=csetcol(wsetno(pc));</pre> <p>increments a counter “x” (such as the cursor position) by the width of the Process Code character <i>pc</i>.</p>
SEE ALSO	setlocale (3C), euclen (3I)
NOTES	cset , csetlen , csetcol , csetno and wsetno can be used safely in a multi-thread application, as long as setlocale (3C) is not being called to change the locale.

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>
MT-LEVEL	See the NOTES section of this page.
DESCRIPTION	<p>ctermid() 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, the contents of which are overwritten at the next call to ctermid(), and the address of which is returned. Otherwise, <i>s</i> is assumed to point to a character array of at least L_ctermid elements; the path name is placed in this array and the value of <i>s</i> is returned. The constant L_ctermid is defined in the header <stdio.h>.</p> <p>ctermid_r() has the same functionality as ctermid() except that if <i>s</i> is a NULL pointer, the function returns NULL.</p>
SEE ALSO	ttyname(3C)
NOTES	<p>The ctermid_r() 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> <p>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.</p> <p>The difference between ctermid() and ttyname(3C) is that ttyname() must be handed a file descriptor and returns the actual name of the terminal associated with that file descriptor, while ctermid() returns a string (/dev/tty) that will refer to the terminal if used as a file name. Thus ttyname() is useful only if the process already has at least one file open to a terminal.</p> <p>ctermid() is unsafe in multithreaded applications. ctermid_r() is MT-Safe, and should be used instead.</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); char *ctime_r(const time_t *clock, char *buf, int buflen); struct tm *localtime(const time_t *clock); struct tm *localtime_r(const time_t *clock, struct tm *res); struct tm *gmtime(const time_t *clock); struct tm *gmtime_r(const time_t *clock, struct tm *res); char *asctime(const struct tm *tm); char *asctime_r(const struct tm *tm, char *buf, int buflen); extern time_t timezone, altzone; extern int daylight; extern char *tzname[2]; void tzset(void); void tzsetwall(void);</pre>
MT-LEVEL	See the NOTES section of this page.
DESCRIPTION	<p>ctime(), localtime(), and gmtime() accept arguments of type time_t, pointed to by clock(), representing the time in seconds since 00:00:00 UTC, January 1, 1970. ctime() returns a pointer to a 26-character string as shown below. Time zone and daylight savings corrections are made before the string is generated. The fields are constant in width:</p> <pre style="padding-left: 40px;">Fri Sep 13 00:00:00 1986\n\0</pre> <p>ctime_r() has the same functionality as ctime() except that a buffer <i>buf</i> with length <i>buflen</i> must be supplied by the caller to store the result. <i>buf</i> must be at least 26 bytes.</p> <p>localtime() and gmtime() return pointers to tm structures, described below. localtime() corrects for the main time zone and possible alternate (“daylight savings”) time zone; gmtime() converts directly to Coordinated Universal Time (UTC), which is the time the UNIX system uses internally.</p> <p>localtime_r() and gmtime_r() have the same functionality as localtime() and gmtime() respectively, except that a buffer <i>res</i> must be supplied by the caller for storing the result.</p> <p>asctime() converts a tm structure to a 26-character string, as shown in the above example, and returns a pointer to the string.</p> <p>asctime_r() has the same functionality as asctime() except that the caller must supply a buffer <i>buf</i> with length <i>buflen</i> for the result to be stored. <i>buf</i> must be at least 26 bytes.</p> <p>asctime_r() returns a pointer to <i>buf</i> upon success. In case of failure, NULL is returned and errno is set.</p>

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] */
                          /* 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 time 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.

tzset() uses the contents of the environment variable **TZ** to override the value of the different external variables. The function **tzset()** 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)**).

tzsetwall() sets things up so that **localtime()** returns the best available approximation of local wall clock time.

LC_TIME determines how these functions handle date and time formats. In the "C" locale, date and time handling follow the U.S. rules.

ERRORS	ctime_r() and asctime_r() will fail if the following is true: ERANGE length of the buffer supplied by caller is not large enough to store the result.
EXAMPLES	<p>tzset() 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</p> <p style="text-align: center;">EST5EDT4,116/2:00:00,298/2:00:00</p> <p>or simply</p> <p style="text-align: center;">EST5EDT</p> <p>An example of a southern hemisphere setting such as the Cook Islands could be</p> <p style="text-align: center;">KDT9:30KST10:00,63/5:00,302/20:00</p> <p>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. ctime(), localtime(), mktime(), and strftime() 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.</p>
FILES	/usr/lib/locale/locale/LC_TIME/time file containing locale specific date and time information
SEE ALSO	time(2) , getenv(3C) , mktime(3C) , printf(3S) , putenv(3C) , setlocale(3C) , strftime(3C) , TIMEZONE(4) , environ(5)
NOTES	<p>All interfaces on this page with an _r suffix are as proposed in the POSIX.4a Draft #6 document, and are subject to change to be compliant to the standard when it is accepted. 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.</p> <p>The return values for ctime(), localtime(), and gmtime() point to static data whose content is overwritten by each call.</p> <p>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.</p>

asctime(), **ctime()**, **gmtime()** and **localtime()** are unsafe in multi-thread applications. **asctime_r()**, **ctime_r()**, **gmtime_r()** and **localtime_r()** are MT-Safe, and should be used instead. **tzset()** and **tzsetwall()** are unsafe in multi-thread applications.

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>
MT-LEVEL	MT-Safe with exceptions
DESCRIPTION	<p>These macros classify character-coded integer values. Each is a predicate returning non-zero for true, zero for false. The behavior of these macros, except isascii(), is affected by the current locale (see setlocale(3C)). To modify the behavior, change the LC_TYPE category in setlocale(), that is, setlocale(LC_CTYPE, newlocale). 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 int, the value of which is representable as an unsigned char, or EOF, which is defined by the stdio.h header and represents end-of-file.</p> <p>isalpha() tests for any character for which isupper() or islower() is true, or any character that is one of an implementation-defined set of characters for which none of iscntrl(), isdigit(), ispunct(), or isspace() is true. In the C locale, isalpha() returns true only for the characters for which isupper() or islower() is true.</p> <p>isupper() tests for any character that is an upper-case letter or is one of an implementation-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.</p>

islower()	tests for any character that is a lower-case letter or is one of an implementation-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]).
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 an implementation-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() is true.
isprint()	tests for any printing character, including space (" ").
isgraph()	tests for any printing character, except space.
iscntrl()	tests for any "control character" as defined by the character set.
isascii()	tests for any ASCII character, code between 0 and 0177 inclusive.

All the character classification macros and the conversion functions and macros use a table lookup.

Functions exist for all the above-defined macros. To get the function form, the macro name must be undefined (for example, **#undef isdigit**).

RETURN VALUES

If the argument to any of the character handling macros is not in the domain of the function, the result is undefined.

FILES

/usr/lib/locale/locale/LC_CTYPE

SEE ALSO

chrtbl(1M), **setlocale(3C)**, **stdio(3S)**, **ascii(5)**, **environ(5)**

NOTES

isdigit(), **isxdigit()**, **islower()**, **isupper()**, **isalpha()**, **isalnum()**, **isspace()**, **iscntrl()**, **ispunct()**, **isprint()**, **isgraph()** and **isascii()** can be used safely in a multi-thread application, as long as **setlocale(3C)** is not being called to change the locale.

NAME	curs_addch, addch, waddch, mvaddch, mvwaddch, echochar, wechochar – add a character (with attributes) to a curses window and advance cursor
SYNOPSIS	<pre>#include <curses.h> int addch(chtype ch); int waddch(WINDOW *win, chtype ch); int mvaddch(int y, int x, chtype ch); int mvwaddch(WINDOW *win, int y, int x, chtype ch); int echochar(chtype ch); int wechochar(WINDOW *win, chtype ch);</pre>
MT-LEVEL	Unsafe
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 scroll() 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 $\^X$ 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 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.</p>
Line Graphics	<p>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.</p>

<i>Name</i>	<i>Default</i>	<i>Glyph Description</i>
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 VALUES All routines return the integer **ERR** upon failure and an integer value other than **ERR** upon successful completion.

SEE ALSO **curs_attr(3X)**, **curs_clear(3X)**, **curs_inch(3X)**, **curs_outopts(3X)**, **curs_refresh(3X)**, **curses(3X)**, **putc(3S)**

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>#include <curses.h> int addchstr(chtype *chstr); int addchnstr(chtype *chstr, int n); int waddchstr(WINDOW *win, chtype *chstr); int waddchnstr(WINDOW *win, chtype *chstr, int n); int mvaddchstr(int y, int x, chtype *chstr); int mvaddchnstr(int y, int x, chtype *chstr, int n); int mvwaddchstr(WINDOW *win, int y, int x, chtype *chstr); int mvwaddchnstr(WINDOW *win, int y, int x, chtype *chstr, int n);</pre>
MT-LEVEL	Unsafe
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 n=-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.
SEE ALSO	curs_addstr(3X) , curses(3X)
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>#include <curses.h> int addstr(char *str); int addnstr(char *str, int n); int waddstr(WINDOW *win, char *str); int waddnstr(WINDOW *win, char *str, int n); int mvaddstr(int y, int x, char *str); int mvaddnstr(int y, int x, char *str, int n); int mvwaddstr(WINDOW *win, int y, int x, char *str); int mvwaddnstr(WINDOW *win, int y, int x, char *str, int n);</pre>
MT-LEVEL	Unsafe
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 ERR upon failure and an integer value other than ERR upon successful completion.
SEE ALSO	curs_addch(3X) , curses(3X)
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 <code>wchar_t</code> character (with attributes) to a curses window and advance cursor
SYNOPSIS	<pre>#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>
MT-LEVEL	Unsafe
DESCRIPTION	<p>The <code>addwch()</code>, <code>waddwch()</code>, <code>mvaddwch()</code>, and <code>mvwaddwch()</code> routines put the character <code>wch</code>, 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 <code>putwchar(3I)</code> 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 <code>wch</code> is a tab, newline, 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 <code>wch</code> is another control character, it is drawn in the <code>^X</code> notation. Calling <code>winwch(3X)</code> 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 <code>inwch()</code> and <code>addwch()</code>.) See <code>standout(3X)</code>, predefined video attribute constants.</p> <p>The <code>echowchar()</code> and <code>wechowchar()</code> routines are functionally equivalent to a call to <code>addwch()</code> followed by a call to <code>refresh(3X)</code>, or a call to <code>waddwch()</code> followed by a call to <code>wrefresh(3X)</code>. 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	<p>The following variables may be used to add line drawing characters to the screen with routines of the <code>addwch()</code> family. When variables are defined for the terminal, the <code>A_ALTCHARSET</code> bit is turned on. (See <code>curs_attr(3X)</code>). Otherwise, the default character listed below is stored in the variable. The names chosen are consistent with the VT100 nomenclature.</p>

<i>Name</i>	<i>Default</i>	<i>Glyph Description</i>
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	•	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.

SEE ALSO **putwchar(3I)**, **clrtoeol(3X)**, **curses(3X)**, **curs_attr(3X)**, **curs_inwch(3X)**, **curs_outopts(3X)**, **refresh(3X)**, **standout(3X)**, **winwch(3X)**, **wrefresh(3X)**

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 <code>wchar_t</code> characters (and attributes) to a curses window
SYNOPSIS	<pre>#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);</pre>
MT-LEVEL	Unsafe
DESCRIPTION	<p>All of these routines copy <code>wchstr</code>, which points to a string of <code>wchar_t</code> characters, directly into the window image structure starting at the current cursor position. The four routines with <code>n</code> as the last argument copy at most <code>n</code> elements, but no more than will fit on the line. If <code>n=-1</code> 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 work faster than <code>waddnwstr(3X)</code> because they merely copy <code>wchstr</code> 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.</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, unless otherwise noted in the preceding routine descriptions.
SEE ALSO	<code>curses(3X)</code> , <code>waddnwstr(3X)</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 all routines except <code>waddwchnstr()</code> may be macros.</p> <p>None of these routines can use the color attribute in <code>chtype</code>.</p>

NAME	<code>curs_addwstr</code> , <code>addwstr</code> , <code>addnwstr</code> , <code>waddwstr</code> , <code>waddnwstr</code> , <code>mvaddwstr</code> , <code>mvaddnwstr</code> , <code>mvwaddwstr</code> , <code>mvwaddnwstr</code> – add a string of <code>wchar_t</code> characters to a curses window and advance cursor
SYNOPSIS	<pre>#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);</pre>
MT-LEVEL	Unsafe
DESCRIPTION	All of these routines write all the characters of the null-terminated <code>wchar_t</code> character string <code>wstr</code> on the given window. The effect is similar to calling <code>waddwch(3X)</code> once for each <code>wchar_t</code> character in the string. The four routines with <code>n</code> as the last argument write at most <code>n</code> <code>wchar_t</code> characters. If <code>n</code> is negative, then the entire string will be added.
RETURN VALUE	All routines return the integer <code>ERR</code> upon failure and an integer value other than <code>ERR</code> upon successful completion.
SEE ALSO	<code>curses(3X)</code> , <code>waddwch(3X)</code>
NOTES	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> . Note that all of these routines except <code>waddwstr()</code> and <code>waddnwstr()</code> 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>#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>
MT-LEVEL	Unsafe
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 (++x) 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 ERR upon failure and an integer value other than ERR upon successful completion.
SEE ALSO	curses(3X) , getyx(3X)
NOTES	<p>The header file <curses.h> automatically includes the header files <stdio.h>, <unctrl.h> and <widec.h>.</p> <p>Note that movenextch(), moveprevch(), and adjcurspos() may be macros.</p>

NAME	curs_attr, attroff, wattroff, attron, wattron, attrset, wattrset, standend, wstandend, standout, wstandout – curses character and window attribute control routines																		
SYNOPSIS	<pre>#include <curses.h> int attroff(int attrs); int wattroff(WINDOW *win, int attrs); int attron(int attrs); int wattron(WINDOW *win, int attrs); int attrset(int attrs); int wattrset(WINDOW *win, int attrs); int standend(void); int wstandend(WINDOW *win); int standout (void); int wstandout(WINDOW *win);</pre>																		
MT-LEVEL	Unsafe																		
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 attron(A_STANDOUT). The routine standend() is the same as attroff(), that is, it turns off all attributes.</p>																		
Attributes	<p>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().</p> <table border="0" style="margin-left: 20px;"> <tr> <td style="padding-right: 20px;">A_STANDOUT</td> <td>Best highlighting mode of the terminal.</td> </tr> <tr> <td>A_UNDERLINE</td> <td>Underlining</td> </tr> <tr> <td>A_REVERSE</td> <td>Reverse video</td> </tr> <tr> <td>A_BLINK</td> <td>Blinking</td> </tr> <tr> <td>A_DIM</td> <td>Half bright</td> </tr> <tr> <td>A_BOLD</td> <td>Extra bright or bold</td> </tr> <tr> <td>A_ALTCHARSET</td> <td>Alternate character set</td> </tr> <tr> <td>A_CHARTEXT</td> <td>Bit-mask to extract a character</td> </tr> <tr> <td>COLOR_PAIR(<i>n</i>)</td> <td>Color-pair number <i>n</i></td> </tr> </table>	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	Bit-mask to extract a character	COLOR_PAIR(<i>n</i>)	Color-pair number <i>n</i>
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	Bit-mask to extract a character																		
COLOR_PAIR(<i>n</i>)	Color-pair number <i>n</i>																		

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.

SEE ALSO

curs_addch(3X), **curs_addstr(3X)**, **curs_printw(3X)**, **curses(3X)**

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>#include <curses.h> int beep(void); int flash(void);</pre>
MT-LEVEL	Unsafe
DESCRIPTION	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.
RETURN VALUES	These routines always return OK .
SEE ALSO	curses(3X)
NOTES	The header <curses.h> automatically includes the headers <stdio.h> and <unctrl.h> .

NAME	curs_bkgd, bkgd, bkgdset, wbkgdset, wbkgd – curses window background manipulation routines
SYNOPSIS	<pre>#include <curses.h> int bkgd(chtype ch); void bkgdset(chtype ch); void wbkgdset(WINDOW *win, chtype ch); int wbkgd(WINDOW *win, chtype ch);</pre>
MT-LEVEL	Unsafe
DESCRIPTION	<p>The bkgdset() and wbkgdset() routines manipulate the background of the named window. Background is a chtype 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 OK , or a non-negative integer, if immedok() is set. See curs_outopts(3X) .
SEE ALSO	curs_addch(3X) , curs_outopts(3X) , curses(3X)
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	<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 <i>tr</i>, chtype <i>bl</i>, chtype <i>br</i>); int box(WINDOW *<i>win</i>, chtype <i>verch</i>, chtype <i>horch</i>); int hline(chtype <i>ch</i>, int <i>n</i>); int whline(WINDOW *<i>win</i>, chtype <i>ch</i>, int <i>n</i>); int vline(chtype <i>ch</i>, int <i>n</i>); int wvline(WINDOW *<i>win</i>, chtype <i>ch</i>, int <i>n</i>);</pre>
MT-LEVEL	Unsafe
DESCRIPTION	<p>With the border(), wborder(), and box() routines, a border is drawn around the edges of the window. The arguments and attributes are:</p> <ul style="list-style-type: none"> <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 <p>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.</p> <p>box(<i>win</i>, <i>verch</i>, <i>horch</i>) is a shorthand for the following call:</p> <pre style="padding-left: 2em;">wborder(<i>win</i>, <i>verch</i>, <i>verch</i>, <i>horch</i>, <i>horch</i> , 0, 0, 0, 0)</pre> <p>hline() and whline() draw a horizontal (left to right) line using <i>ch</i> starting at the current cursor position in the window. The current cursor position is not changed. The line is at most <i>n</i> characters long, or as many as fit into the window.</p> <p>vline() and wvline() draw a vertical (top to bottom) line using <i>ch</i> starting at the current cursor position in the window. The current cursor position is not changed. The line is at most <i>n</i> characters long, or as many as fit into the window.</p>
RETURN VALUES	All routines return the integer OK , or a non-negative integer if immedok() is set. See curs_outopts(3X) .

SEE ALSO `curs_outopts(3X)`, `curses(3X)`

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, clrtoebol, wclrtoebol, clreol, wclreol – clear all or part of a curses window
SYNOPSIS	<pre>#include <curses.h> int erase(void); int werase(WINDOW *win); int clear(void); int wclear(WINDOW *win); int clrtoebol(void); int wclrtoebol(WINDOW *win); int clreol(void); int wclreol(WINDOW *win);</pre>
MT-LEVEL	Unsafe
DESCRIPTION	<p>The erase() and werase() routines copy blanks to every position in the window. 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 clrtoebol() and wclrtoebol() 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 clreol() and wclreol() 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_ouptops(3X) .
SEE ALSO	curs_ouptops(3X) , curs_refresh(3X) , curses(3X)
NOTES	The header <curses.h> automatically includes the headers <stdio.h> and <unctrl.h> . Note that erase() , werase() , clear() , wclear() , clrtoebol() , and clreol() 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

```
#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 *fgp, short *bgp);
```

MT-LEVEL Unsafe

DESCRIPTION
Overview

curses provides routines that manipulate color on color alphanumeric terminals. To use these routines **start_color()** must be called, usually right after **initscr()**. See **curs_initscr(3X)**. 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 **init_pair**. After it has been initialized, **COLOR_PAIR(n)**, a macro defined in **<curses.h>**, 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 **init_color()** to change the definition of a color. The routines **has_colors()** and **can_change_color()** return **TRUE** or **FALSE**, depending on whether the terminal has color capabilities and whether the programmer can change the colors. The routine **color_content()** allows a programmer to identify the amounts of red, green, and blue components in an initialized color. The routine **pair_content()** allows a programmer to find out how a given color-pair is currently defined.

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 **shorts** 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 **shorts** 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.

SEE ALSO

curs_attr(3X), **curs_initscr(3X)**, **curses(3X)**

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>#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>
MT-LEVEL	Unsafe
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 ERR upon failure and an integer value other than ERR upon successful completion.
SEE ALSO	curses(3X)
NOTES	The header <curses.h> automatically includes the headers <stdio.h> and <unctrl.h> . Note that delch() , mvdelch() , and mvwdelch() may be macros.

NAME	curs_deleteln, deleteln, wdeleteln, insdelln, winsdelln, insertln, winsertln – delete and insert lines in a curses window
SYNOPSIS	<pre>#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>
MT-LEVEL	Unsafe
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 ERR upon failure and an integer value other than ERR upon successful completion.
SEE ALSO	curses(3X)
NOTES	The header <curses.h> automatically includes the headers <stdio.h> and <unctrl.h> . Note that all but winsdelln() may be macros.

NAME	curs_getch, getch, wgetch, mvgetch, mvwgetch, ungetch – get (or push back) characters from curses terminal keyboard
SYNOPSIS	<pre>#include <curses.h> int getch(void); int wgetch(WINDOW *win); int mvgetch(int y, int x); int mvwgetch(WINDOW *win, int y, int x); int ungetch(int ch);</pre>
MT-LEVEL	Unsafe
DESCRIPTION	<p>With the getch(), wgetch(), mvgetch(), and mvwgetch() routines a character is read from the terminal associated with the window. In no-delay mode, if no input is waiting, the value ERR 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 TRUE, 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 <curses.h> with integers beginning with 0401, whose names begin with KEY_. If a character that could be the beginning of a function key (such as escape) is received, curses 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> <p>The ungetch() routine places <i>ch</i> back onto the input queue to be returned by the next call to wgetch().</p>
Function Keys	<p>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 <i>terminfo</i> database.</p>

<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(n)	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
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
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
KEY_SREDO	Shifted redo key
KEY_SREPLACE	Shifted replace key
KEY_SRIGHT	Shifted right arrow
KEY_SRESUME	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.

SEE ALSO

curs_inopts(3X), **curs_move(3X)**, **curs_refresh(3X)**, **curses(3X)**

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	<code>curs_getstr</code> , <code>getstr</code> , <code>wgetstr</code> , <code>mvgetstr</code> , <code>mvwgetstr</code> , <code>wgetnstr</code> – get character strings from curses terminal keyboard
SYNOPSIS	<pre>#include <curses.h> int getstr(char *str); int wgetstr(WINDOW *win, char *str); int mvgetstr(int y, int x, char *str); int mvwgetstr(WINDOW *win, int y, int x, char *str); int wgetnstr(WINDOW *win, char *str, int n);</pre>
MT-LEVEL	Unsafe
DESCRIPTION	The effect of <code>getstr()</code> is as though a series of calls to <code>getch()</code> were made, until a newline or carriage return is received. The resulting value is placed in the area pointed to by the character pointer <code>str</code> . <code>wgetnstr()</code> reads at most <code>n</code> 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 VALUES	All routines return the integer <code>ERR</code> upon failure and an integer value other than <code>ERR</code> upon successful completion.
SEE ALSO	<code>curs_getch(3X)</code> , <code>curses(3X)</code>
NOTES	The header <code><curses.h></code> automatically includes the headers <code><stdio.h></code> and <code><unctrl.h></code> . Note that <code>getstr()</code> , <code>mvgetstr()</code> , and <code>mvwgetstr()</code> may be macros.

NAME	curs_getwch, getwch, wgetwch, mvgetwch, mvwgetwch, ungetwch – get (or push back) wchar_t characters from curses terminal keyboard
SYNOPSIS	<pre>#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>
MT-LEVEL	Unsafe
DESCRIPTION	<p>The getwch(), wgetwch(), mvgetwch(), and mvwgetwch() routines read an EUC character from the terminal associated with the window, transform it into a wchar_t character, and return a wchar_t character. In no-delay mode, if no input is waiting, the value ERR 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(3X), wrefresh will be called before another character is read.</p> <p>If keypad is TRUE, 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 <curses.h> with integers beginning with 0401, whose names begin with KEY_. If a character that could be the beginning of a function key (such as escape) is received, curses(3X) 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> <p>The ungetwch() routine places wch back onto the input queue to be returned by the next call to wgetwch().</p>
Function Keys	<p>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.</p>

<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
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
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
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.
SEE ALSO	curses(3X) , curs_inopts(3X) , curs_move(3X) , wrefresh(3X) , terminfo(4)
NOTES	<p>The header file <curses.h> automatically includes the header files <stdio.h>, <unctrl.h> and <widec.h>.</p> <p>Use of the escape key by a programmer for a single character function is discouraged.</p> <p>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.</p> <p>Note that getwch(), mvgetwch(), and mvwgetwch() may be macros.</p>

NAME	curs_getwstr, getwstr, getnwstr, wgetwstr, wgetnwstr, mvgetwstr, mvgetnwstr, mvwgetwstr, mvwgetnwstr – get wchar_t character strings from curses terminal keyboard
SYNOPSIS	<pre>#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);</pre>
MT-LEVEL	Unsafe
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 <i>wstr</i> . getnwstr() reads at most <i>n</i> 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.
SEE ALSO	curses(3X) , getwch(3X)
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	<pre>#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);</pre>
MT-LEVEL	Unsafe
DESCRIPTION	<p>With the <code>getyx()</code> macro, the cursor position of the window is placed in the two integer variables <code>y</code> and <code>x</code>.</p> <p>With the <code>getparyx()</code> macro, if <code>win</code> is a subwindow, the beginning coordinates of the subwindow relative to the parent window are placed into two integer variables, <code>y</code> and <code>x</code>. Otherwise, <code>-1</code> is placed into <code>y</code> and <code>x</code>.</p> <p>Like <code>getyx()</code>, the <code>getbegyx()</code> and <code>getmaxyx()</code> macros store the current beginning coordinates and size of the specified window.</p>
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).
SEE ALSO	<code>curses(3X)</code>
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 all of these interfaces are macros and that “&” is not necessary before the variables <code>y</code> and <code>x</code>.</p>

NAME	curs_inch, inch, winch, mvinch, mvwinch – get a character and its attributes from a curses window						
SYNOPSIS	<pre>#include <curses.h> ctype inch(void); ctype winch(WINDOW *win); ctype mvinch(int y, int x); ctype mvwinch(WINDOW *win, int y, int x);</pre>						
MT-LEVEL	Unsafe						
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() . <table><tr><td>A_CHARTEXT</td><td>Bit-mask to extract character</td></tr><tr><td>A_ATTRIBUTES</td><td>Bit-mask to extract attributes</td></tr><tr><td>A_COLOR</td><td>Bit-mask to extract color-pair field information</td></tr></table>	A_CHARTEXT	Bit-mask to extract character	A_ATTRIBUTES	Bit-mask to extract attributes	A_COLOR	Bit-mask to extract color-pair field information
A_CHARTEXT	Bit-mask to extract character						
A_ATTRIBUTES	Bit-mask to extract attributes						
A_COLOR	Bit-mask to extract color-pair field information						
SEE ALSO	curses(3X)						
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	<pre>#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);</pre>
MT-LEVEL	Unsafe
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 <i>n</i> as the last argument, return the string at most <i>n</i> 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 <i>chstr</i> (see curs_inch(3X)).
RETURN VALUES	All routines return the integer ERR upon failure and an integer value other than ERR upon successful completion.
SEE ALSO	curs_inch(3X) , curses(3X)
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>#include <curses.h> WINDOW *initscr(void); int endwin(void); int isendwin(void); SCREEN *newterm(char *type, FILE *outfd, FILE *infd); SCREEN *set_term(SCREEN *new); void delscreen(SCREEN* sp);</pre>
MT-LEVEL	Unsafe
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 curses 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 SCREEN * 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 \$TERM, a file pointer for output to the terminal, and another file pointer for input from the terminal (if <i>type</i> is NULL, \$TERM will be used). The program must also call endwin() for each terminal being used before exiting from curses. 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> <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.</p> <p>The isendwin() routine returns TRUE if endwin() has been called without any subsequent calls to wrefresh(), and FALSE otherwise.</p> <p>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.</p>

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.

SEE ALSO

curs_kernel(3X), **curs_refresh(3X)**, **curs_slk(3X)**, **curs_util(3X)**, **curses(3X)**

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>#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>
MT-LEVEL	Unsafe
DESCRIPTION	<p>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.</p> <p>Note that cbreak() overrides raw(). (See curs_getch(3X) for a discussion of how these routines interact with echo() and noecho().)</p> <p>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</p>

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(7)**). 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(7)**). 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.

SEE ALSO

curs_getch(3X), **curs_initscr(3X)**, **curses(3X)**, **termio(7)**

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, mvwünsch, mvwünsch – insert a character before the character under the cursor in a curses window
SYNOPSIS	<pre>#include <curses.h> int insch(chtype ch); int wünsch(WINDOW *win, chtype ch); int mvwünsch(int y, int x, chtype ch); int mvwünsch(WINDOW *win, int y, int x, chtype ch);</pre>
MT-LEVEL	Unsafe
DESCRIPTION	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.)
RETURN VALUES	All routines return the integer ERR upon failure and an integer value other than ERR upon successful completion.
SEE ALSO	curses(3X)
NOTES	The header <curses.h> automatically includes the headers <stdio.h> and <unctrl.h> . Note that insch() , mvwünsch() , and mvwünsch() may be macros.

NAME	curs_insstr, insstr, insnstr, winsstr, winsnstr, mvinsstr, mvinsnstr, mvwinsstr, mvwinsnstr – insert string before character under the cursor in a curses window
SYNOPSIS	<pre>#include <curses.h> int insstr(char *str); int insnstr(char *str, int n); int winsstr(WINDOW *win, char *str); int winsnstr(WINDOW *win, char *str, int n); int mvinsstr(int y, int x, char *str); int mvinsnstr(int y, int x, char *str, int n); int mvwinsstr(WINDOW *win, int y, int x, char *str); int mvwinsnstr(WINDOW *win, int y, int x, char *str, int n);</pre>
MT-LEVEL	Unsafe
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 <i>^X</i> 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.
SEE ALSO	curs_clear(3X) , curs_inch(3X) , curses(3X)
NOTES	The header <curses.h> automatically includes the headers <stdio.h> and <unctrl.h> . Note that all but winsnstr() may be macros.

NAME	<code>curs_instr</code> , <code>instr</code> , <code>innstr</code> , <code>winstr</code> , <code>winnstr</code> , <code>mvinstr</code> , <code>mvinnstr</code> , <code>mvwinstr</code> , <code>mvwinnstr</code> – get a string of characters from a curses window
SYNOPSIS	<pre>#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);</pre>
MT-LEVEL	Unsafe
DESCRIPTION	These routines return a string of characters in <i>str</i> , 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 <i>n</i> as the last argument return the string at most <i>n</i> characters long.
RETURN VALUES	All routines return the integer ERR upon failure and an integer value other than ERR upon successful completion.
SEE ALSO	<code>curses(3X)</code>
NOTES	The header <code><curses.h></code> automatically includes the headers <code><stdio.h></code> and <code><unctrl.h></code> . Note that all routines except <code>winnstr()</code> may be macros.

NAME	<code>curs_inswch</code> , <code>inswch</code> , <code>winswch</code> , <code>mvinswch</code> , <code>mvwinswch</code> – insert a <code>wchar_t</code> character before the character under the cursor in a curses window
SYNOPSIS	<pre>#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>
MT-LEVEL	Unsafe
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.
SEE ALSO	<code>curses(3X)</code>
NOTES	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> . Note that <code>inswch()</code> , <code>mvinswch()</code> , and <code>mvwinswch()</code> may be macros. None of these routines can use the color attribute in <code>chtype</code> .

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>#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>
MT-LEVEL	Unsafe
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, 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 <code>ERR</code> upon failure and an integer value other than <code>ERR</code> upon successful completion.
SEE ALSO	<code>clrtoeol(3X)</code> , <code>curses(3X)</code> , <code>winwch(3X)</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 all but <code>winsnwstr()</code> may be macros.</p>

NAME	curs_inwch, inwch, winwch, mvinwch, mvwinwch – get a <code>wchar_t</code> character and its attributes from a curses window
SYNOPSIS	<pre>#include <curses.h> chtype inwch(void); chtype winwch(WINDOW *win); chtype mvinwch(int y, int x); chtype mvwinwch(WINDOW *win, int y, int x);</pre>
MT-LEVEL	Unsafe
DESCRIPTION	These routines return the <code>wchar_t</code> character, of type <code>chtype</code> , 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 <code><curses.h></code> can be used with the logical AND (<code>&</code>) operator to extract the character or attributes alone.
Attributes	<p>The following bit-masks may be AND-ed with characters returned by <code>winwch()</code>.</p> <pre> A_WCHARTEXT Bit-mask to extract character A_WATTRIBUTES Bit-mask to extract attributes</pre>
SEE ALSO	<code>curses(3X)</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 all of these routines may be macros.</p> <p>None of these routines can use the color attribute in <code>chtype</code>.</p>

NAME	curs_inwchstr, inwchstr, inwchnstr, winwchstr, winwchnstr, mvinwchstr, mvinwchnstr, mvwinwchstr, mvwinwchnstr – get a string of <code>wchar_t</code> characters (and attributes) from a curses window
SYNOPSIS	<pre>#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);</pre>
MT-LEVEL	Unsafe
DESCRIPTION	These routines return a string of type <code>chtype</code> , holding <code>wchar_t</code> characters, starting at the current cursor position in the named window and ending at the right margin of the window. The four functions with <code>n</code> as the last argument, return the string at most <code>n wchar_t</code> characters long. Constants defined in <code><curses.h></code> can be used with the logical AND (<code>&</code>) operator to extract the <code>wchar_t</code> character or the attribute alone from any position in the <code>wchstr</code> (see <code>curs_inwch(3X)</code>).
RETURN VALUE	All routines return the integer <code>ERR</code> upon failure and an integer value other than <code>ERR</code> upon successful completion.
SEE ALSO	<code>curses(3X)</code> , <code>curs_inwch(3X)</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 all routines except <code>winwchnstr()</code> may be macros.</p> <p>None of these routines can use the color attribute in <code>chtype</code>.</p>

NAME	<code>curs_inwstr</code> , <code>inwstr</code> , <code>innwstr</code> , <code>winwstr</code> , <code>winnwstr</code> , <code>mvinwstr</code> , <code>mvinnwstr</code> , <code>mvwinwstr</code> , <code>mvwinnwstr</code> – get a string of <code>wchar_t</code> characters from a curses window
SYNOPSIS	<pre>#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);</pre>
MT-LEVEL	Unsafe
DESCRIPTION	These routines return the string of <code>wchar_t</code> characters in <code>wstr</code> 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 <code>n</code> as the last argument return the string at most <code>n</code> <code>wchar_t</code> characters long.
RETURN VALUES	All routines return the integer <code>ERR</code> upon failure and an integer value other than <code>ERR</code> upon successful completion.
SEE ALSO	<code>curses(3X)</code>
NOTES	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> . Note that all routines except <code>winnwstr()</code> 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>#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 line, int (*init)(WINDOW *, int)); int curs_set(int visibility); int napms(int ms);</pre>
MT-LEVEL	Unsafe
DESCRIPTION	<p>The following routines give low-level access to various curses functionality. These routines typically are used inside library routines.</p> <p>The def_prog_mode() and def_shell_mode() routines save the current terminal modes as the “program” (in curses) or “shell” (not in curses) state for use by the reset_prog_mode() and reset_shell_mode() routines. This is done automatically by initscr().</p> <p>The reset_prog_mode() and reset_shell_mode() routines restore the terminal to “program” (in curses) or “shell” (out of curses) state. These are done automatically by endwin() and, after an endwin(), by doupdate(), so they normally are not called.</p> <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().</p> <p>With the getsyx() routine, the current coordinates of the virtual screen cursor are returned in <i>y</i> and <i>x</i>. If leaveok() is currently TRUE, then -1,-1 is returned. If lines have been removed from the top of the screen, using ripoffline(), <i>y</i> and <i>x</i> include these lines; therefore, <i>y</i> and <i>x</i> should be used only as arguments for setsyx().</p> <p>With the setsyx() routine, the virtual screen cursor is set to <i>y</i>, <i>x</i>. If <i>y</i> and <i>x</i> 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().</p>

The **ripoffline()** routine provides access to the same facility that **slk_init()** (see **curs_slk(3X)**) uses to reduce the size of the screen. **ripoffline()** 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.

ripoffline() 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.

SEE ALSO

curs_initscr(3X), **curs_outopts(3X)**, **curs_refresh(3X)**, **curs_scr_dump(3X)**, **curs_slk(3X)**, **curses(3X)**

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	#include <curses.h> int move(int y, int x); int wmove(WINDOW *win, int y, int x);
MT-LEVEL	Unsafe
DESCRIPTION	With these routines, the cursor associated with the window is moved to line <i>y</i> and column <i>x</i> . 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.
SEE ALSO	curs_refresh(3X) , curses(3X)
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>#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>
MT-LEVEL	Unsafe
DESCRIPTION	<p>These routines set options that deal with output within curses. All options are initially FALSE, unless otherwise stated. It is not necessary to turn these options off before calling endwin().</p> <p>With the clearok() routine, if enabled (<i>bf</i> is TRUE), the next call to wrefresh() 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 <i>win</i> argument to clearok() is the global variable curscr(), the next call to wrefresh() with any window causes the screen to be cleared and repainted from scratch.</p> <p>With the idlok() routine, if enabled (<i>bf</i> is TRUE), curses considers using the hardware insert/delete line feature of terminals so equipped. If disabled (<i>bf</i> is FALSE), curses very seldom uses this feature. (The insert/delete character feature is always considered.) This option should be enabled only if the 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.</p> <p>With the idcok() routine, if enabled (<i>bf</i> is TRUE), curses considers using the hardware insert/delete character feature of terminals so equipped. This is enabled by default.</p> <p>With the immedok() routine, if enabled (<i>bf</i> is TRUE), any change in the window image, such as the ones caused by waddch(), wclrtoeol(), 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</p>

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**.

SEE ALSO

curs_addch(3X), **curs_clear(3X)**, **curs_initscr(3X)**, **curs_refresh(3X)**, **curs_scroll(3X)**, **curses(3X)**

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	<pre>#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);</pre>
MT-LEVEL	Unsafe
DESCRIPTION	<p>The overlay() and overwrite() routines overlay <i>srcwin</i> on top of <i>dstwin</i>. <i>srcwin</i> and <i>dstwin</i> 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.</p> <p>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, (<i>dminrow</i>, <i>dmincol</i>) and (<i>dmaxrow</i>, <i>dmaxcol</i>), and the upper-left-corner coordinates of the source window, (<i>sminrow</i>, <i>smincol</i>). If the argument <i>overlay</i> is true, then copying is non-destructive, as in overlay().</p>
RETURN VALUES	Routines that return an integer return ERR upon failure and an integer value other than ERR upon successful completion.
SEE ALSO	curs_pad(3X) , curs_refresh(3X) , curses(3X)
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>#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>
MT-LEVEL	Unsafe
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> <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. <i>pminrow</i> and <i>pmincol</i> specify the upper left-hand corner of the rectangle to be displayed in the pad. <i>sminrow</i>, <i>smincol</i>, <i>smaxrow</i>, and <i>smaxcol</i> 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 <i>pminrow</i>, <i>pmincol</i>, <i>sminrow</i>, or <i>smincol</i> are treated as if they were zero.</p> <p>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</p>

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.

SEE ALSO **addch(3X)**, **curses(3X)**, **refresh(3X)**, **subwin(3X)**, **touchline(3X)**, **touchwin(3X)**, **waddch(3X)**, **wnoutrefresh(3X)**, **wrefresh(3X)**

NOTES The header **<curses.h>** automatically includes the headers **<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>#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>
MT-LEVEL	Unsafe
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 va_list, a pointer to a list of arguments, as defined in <varargs.h>.</p>
RETURN VALUES	All routines return the integer ERR upon failure and an integer value other than ERR upon successful completion.
SEE ALSO	curses(3X) , printf(3S) , vprintf(3S)
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>#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>
MT-LEVEL	Unsafe
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 stdscr 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, curses 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 one burst of output, with fewer total characters transmitted and less CPU time used. If the <i>win</i> argument to wrefresh() is the global variable curscr, the screen is immediately cleared and repainted from scratch.</p> <p>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 wredrawln() 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.</p>

RETURN VALUES All routines return the integer **ERR** upon failure and an integer value other than **ERR** upon successful completion.

SEE ALSO **curs_outopts(3X)**, **curses(3X)**

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>#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 varglist);</pre>
MT-LEVEL	Unsafe
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 <i>fmt</i> 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 <i>va_list</i>, a pointer to a list of arguments, as defined in <varargs.h>.</p>
RETURN VALUES	<p>vwscanw() returns ERR 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>
SEE ALSO	curs_getstr(3X) , curs_printw(3X) , curses(3X) , scanf(3S)
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>#include <curses.h> int scr_dump(char *filename); int scr_restore(char *filename); int scr_init(char *filename); int scr_set(char *filename);</pre>
MT-LEVEL	Unsafe
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 curses data structures about what the terminal currently has on its screen. If the data is determined to be valid, curses 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.
SEE ALSO	curs_initscr(3X) , curs_refresh(3X) , curs_util(3X) , curses(3X) , system(3S)
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, wscrl – scroll a curses window
SYNOPSIS	<pre>#include <curses.h> int scroll(WINDOW *win); int scl(int n); int wscrl(WINDOW *win, int n);</pre>
MT-LEVEL	Unsafe
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 wscrl() 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 ERR upon failure and an integer value other than ERR upon successful completion.
SEE ALSO	curs_outopts(3X) , curses(3X)
NOTES	The header <curses.h> automatically includes the headers <stdio.h> and <unctrl.h> . Note that scl() and scroll() may be macros.

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>#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>
MT-LEVEL	Unsafe
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 stdscr, reducing the size of stdscr and the variable LINES. curses standardizes on eight labels of up to eight characters each.</p> <p>To use soft labels, the slk_init() routine must be called before initscr() or newterm() is called. If initscr() eventually uses a line from stdscr to emulate the soft labels, then <i>fmt</i> determines how the labels are arranged on the screen. Setting <i>fmt</i> to 0 indicates a 3-2-3 arrangement of the labels; 1 indicates a 4-4 arrangement.</p> <p>With the slk_set() routine, <i>labnum</i> is the label number, from 1 to 8. <i>label</i> is the string to be put on the label, up to eight characters in length. A null string or a null pointer sets up a blank label. <i>fmt</i> is either 0, 1, or 2, indicating whether the label is to be left-justified, centered, or right-justified, respectively, within the label.</p> <p>The slk_refresh() and slk_noutrefresh() routines correspond to the wrefresh() and wnoutrefresh() routines.</p> <p>With the slk_label() routine, the current label for label number <i>labnum</i> is returned with leading and trailing blanks stripped.</p> <p>With the slk_clear() routine, the soft labels are cleared from the screen.</p> <p>With the slk_restore() routine, the soft labels are restored to the screen after a slk_clear() is performed.</p>

With the **slk_touch()** routine, all the soft labels are forced to be output the next time a **slk_noutrefresh()** is performed.

The **slk_atron()**, **slk_attrset()**, and **slk_attroff()** routines correspond to **atron()**, **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.

SEE ALSO

curs_attr(3X), **curs_initscr(3X)**, **curs_refresh(3X)**, **curses(3X)**

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>#include <curses.h> int baudrate(void); char erasechar(void); int has_ic(void); int has_il(void); char killchar(void); char *longname(void); chtype termattrs(void); char *termname(void);</pre>
MT-LEVEL	Unsafe
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 restored by set_term(), so the value should be saved between calls to newterm() if longname() is going to be used with multiple terminals.</p> <p>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.</p> <p>The termname() routine returns the value of the environment variable TERM (truncated to 14 characters).</p>
RETURN VALUES	<p>longname() and termname() return NULL on error.</p> <p>Routines that return an integer return ERR upon failure and an integer value other than ERR upon successful completion.</p>

SEE ALSO `curs_initscr(3X)`, `curs_outopts(3X)`, `curses(3X)`

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>#include <curses.h> #include <term.h> int tgetent(char *bp, char *name); int tgetflag(char id[2]); int tgetnum(char id[2]); char *tgetstr(char id[2], char **area); char *tgoto(char *cap, int col, int row); int tputs(char *str, int affcnt, int (*putc)(void));</pre>
MT-LEVEL	Unsafe
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	<p>Routines that return an integer return ERR upon failure and an integer value other than ERR upon successful completion.</p> <p>Routines that return pointers return NULL on error.</p>
SEE ALSO	curs_terminfo(3X) , curses(3X) , putc(3S)
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> #include <curses.h> #include <term.h> int setupterm(char *term, int fildes, int *errret); int setterm(char *term); int set_curterm(TERMINAL *nterm); int del_curterm(TERMINAL *oterm); int restartterm(char *term, int fildes, int *errret); char *tparm(char *str, long int p1, long int p2, long int p3, long int p4, long int p5, long int p6, long int p7, long int p8, long int p9); int tputs(char *str, int affcnt, int (*putc)(char)); int putp(char *str); int vidputs(chtype attrs, int (*putc)(char)); int vidattr(chtype attrs); int mvcur(int oldrow, int oldcol, int newrow, int newcol); int tigetflag(char *capname); int tigetnum(char *capname); int tigetstr(char *capname); </pre>
MT-LEVEL	Unsafe
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, curses routines are more suitable and their use is recommended.</p> <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 <i>terminfo</i> variables lines and columns are initialized by setupterm() as follows: If use_env(FALSE) has been called, values for lines and columns specified in <i>terminfo</i> 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 <i>terminfo</i> database are used.</p> <p>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 <i>terminfo</i> strings (including the output of tparm()) should be printed with tputs() or putp(). Call the reset_shell_mode() routine</p>

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 *fildev* 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 **curses(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.

SEE ALSO

curs_initscr(3X), **curs_kernel(3X)**, **curs_termcap(3X)**, **curses(3X)**, **putc(3S)**, **terminfo(4)**

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 – cursers refresh control routines
SYNOPSIS	<pre>#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>
MT-LEVEL	Unsafe
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=1</i>) or have not (<i>changed=0</i>) 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.
SEE ALSO	curs_refresh(3X) , curses(3X)
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>#include <curses.h> char *unctrl(chtype c); char *keyname(int c); int filter(void); void use_env(char bool); int putwin(WINDOW *win, FILE *filep); WINDOW *getwin(FILE *filep); int delay_output(int ms); int flushinp(void);</pre>
MT-LEVEL	Unsafe
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 curses think that there is a one-line screen. curses 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 lines and columns specified in the <i>terminfo</i> database will be used, even if environment variables LINES and COLUMNS (used by default) are set, or if curses is running in a window (in which case default behavior would be to use the window size if LINES and COLUMNS are not set).</p> <p>With the putwin() routine, all data associated with window <i>win</i> is written into the file to which <i>filep</i> points. This information can be later retrieved using the getwin() function.</p> <p>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.</p> <p>The delay_output() routine inserts an <i>ms</i> millisecond pause in output. This routine should not be used extensively because padding characters are used rather than a CPU pause.</p> <p>The flushinp() routine throws away any typeahead that has been typed by the user and has not yet been read by the program.</p>

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.

SEE ALSO **curs_initscr(3X)**, **curs_scr_dump(3X)**, **curses(3X)**

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>#include <curses.h> WINDOW *newwin(int nlines, int ncols, int begin_y, int begin_x); int delwin(WINDOW *win); int mvwin(WINDOW *win, int y, int x); WINDOW *subwin(WINDOW *orig, int nlines, int ncols, int begin_y, int begin_x); WINDOW *derwin(WINDOW *orig, int nlines, int ncols, int begin_y, int begin_x); int mvderwin(WINDOW *win, int par_y, int par_x); WINDOW *dupwin(WINDOW *win); void wsyncup(WINDOW *win); int syncok(WINDOW *win, bool bf); void wcursyncup(WINDOW *win); void wsyncdown(WINDOW *win);</pre>
MT-LEVEL	Unsafe
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 newwin(0,0,0,0).</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> <p>The subwin() 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 window is at position (<i>begin_y</i>, <i>begin_x</i>) on the screen. (This position is relative to the screen, and not to the window <i>orig</i>.) The window is made in the middle of the window <i>orig</i>, so that changes made to one window will affect both windows. The subwindow shares memory with the window <i>orig</i>. When using this routine, it is necessary to call touchwin() or touchline() on <i>orig</i> before calling wrefresh() on the subwindow.</p> <p>The derwin() routine is the same as subwin(), except that <i>begin_y</i> and <i>begin_x</i> are relative to the origin of the window <i>orig</i> rather than the screen. There is no difference between the subwindows and the derived windows.</p>

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.

SEE ALSO

curs_refresh(3X), **curs_touch(3X)**, **curses(3X)**

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	curses – CRT screen handling and optimization package
SYNOPSIS	<pre>cc [flag ...] file ... -lcurses [library ...] #include <curses.h></pre>
MT-LEVEL	Unsafe
DESCRIPTION	<p>The curses library routines give the user a terminal-independent method of updating character screens with reasonable optimization.</p> <p>The curses package allows: overall screen, window and pad manipulation; output to windows and pads; reading terminal input; control over terminal and curses input and output options; environment query routines; color manipulation; use of soft label keys; terminfo access; and access to low-level curses routines.</p> <p>To initialize the routines, the routine initscr() or newterm() must be called before any of the other routines that deal with windows and screens are used. The routine endwin() 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 style="padding-left: 40px;">initscr,cbreak,noecho;</pre> <p>Most programs would additionally use the sequence:</p> <pre style="padding-left: 40px;">nonl,intrflush(stdscr,FALSE),keypad(stdscr,TRUE);</pre> <p>Before a curses 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 tput init command after the shell environment variable TERM has been exported. (See terminfo(4) for further details.)</p> <p>The curses 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 stdscr, which is the size of the terminal screen, is supplied. Others may be created with newwin(3X).</p> <p>Windows are referred to by variables declared as WINDOW *. These data structures are manipulated with routines described on 3X pages (whose names begin "curs_"). Among which the most basic routines are move(3X) and addch(3X). More general versions of these routines are included with names beginning with w, allowing the user to specify a window. The routines not beginning with w affect stdscr.</p> <p>After using routines to manipulate a window, refresh(3X) is called, telling curses to make the user's CRT screen look like stdscr. The characters in a window are actually of type chtype, (character and attribute data) so that other information about the character may also be stored with each character.</p> <p>Special windows called <i>pads</i> 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.</p>

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.

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.

International Functions

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	 curs_addch(3X)
addchnstr	 curs_addchstr(3X)
addchstr	 curs_addchstr(3X)
addnstr	 curs_addstr(3X)
addnwstr	 curs_addwstr(3X)
addstr	 curs_addstr(3X)
addwch	 curs_addwch(3X)
addwchnstr	 curs_addwchstr(3X)
addwchstr	 curs_addwchstr(3X)
addwstr	 curs_addwstr(3X)
adjcurspos	 curs_alecompat(3X)
attroff	 curs_attr(3X)
attron	 curs_attr(3X)
attrset	 curs_attr(3X)
baudrate	 curs_termattrs(3X)
beep	 curs_beep(3X)
bkgd	 curs_bkgd(3X)
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	curs_termattrs(3X)
filter	curs_util(3X)
flash	curs_beep(3X)
flushinp	curs_util(3X)
getbegyx	curs_getyx(3X)
getch	curs_getch(3X)
getmaxyx	curs_getyx(3X)
getnwstr	curs_getwstr(3X)
getparyx	curs_getyx(3X)
getstr	curs_getstr(3X)
getsyx	curs_kernel(3X)
getwch	curs_getwch(3X)
getwin	curs_util(3X)
getwstr	curs_getwstr(3X)
getyx	curs_getyx(3X)
halfdelay	curs_inopts(3X)
has_colors	curs_color(3X)
has_ic	curs_termattrs(3X)
has_il	curs_termattrs(3X)
idcok	curs_outopts(3X)
idlok	curs_outopts(3X)
immedok	curs_outopts(3X)
inch	curs_inch(3X)
inchstr	curs_inchstr(3X)
inchstr	curs_inchstr(3X)
init_color	curs_color(3X)
init_pair	curs_color(3X)
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_insstr(3X)
insnwstr	curs_inswstr(3X)
insstr	curs_insstr(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)
mvwadddch	 curs_addch(3X)
mvwadddchnstr	 curs_addchstr(3X)
mvwadddchstr	 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)
mvprintw	 curs_printw(3X)
mvscanw	 curs_scanw(3X)
napms	 curs_kernel(3X)
newpad	 curs_pad(3X)
newterm	 curs_initscr(3X)
newwin	 curs_window(3X)

nl	 curs_ouptopts(3X)
nocbreak	 curs_inoptts(3X)
nodelay	 curs_inoptts(3X)
noecho	 curs_inoptts(3X)
nonl	 curs_ouptopts(3X)
noqiflush	 curs_inoptts(3X)
noraw	 curs_inoptts(3X)
notimeout	 curs_inoptts(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_inoptts(3X)
raw	 curs_inoptts(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)
riporffline	 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	 curs_scroll(3X)
scrollok	 curs_ouptopts(3X)
set_curterm	 curs_terminfo(3X)
set_term	 curs_initscr(3X)
setscreg	 curs_ouptopts(3X)
setsyx	 curs_kernel(3X)
setterm	 curs_terminfo(3X)
setupterm	 curs_terminfo(3X)
slk_attroff	 curs_slk(3X)
slk_attron	 curs_slk(3X)
slk_attrset	 curs_slk(3X)
slk_clear	 curs_slk(3X)

slk_init	 curs_slk(3X)
slk_label	 curs_slk(3X)
slk_noutrefresh	 curs_slk(3X)
slk_refresh	 curs_slk(3X)
slk_restore	 curs_slk(3X)
slk_set	 curs_slk(3X)
slk_touch	 curs_slk(3X)
srcl	 curs_scroll(3X)
standend	 curs_attr(3X)
standout	 curs_attr(3X)
start_color	 curs_color(3X)
subpad	 curs_pad(3X)
subwin	 curs_window(3X)
syncok	 curs_window(3X)
termattrs	 curs_termattrs(3X)
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)
tparm	 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	 curs_addchstr(3X)
waddnstr	 curs_addstr(3X)
waddnwstr	 curs_addwstr(3X)
waddstr	 curs_addstr(3X)

waddwch	curs_addwch(3X)
waddwchnstr	curs_addwchstr(3X)
waddwchstr	curs_addwchstr(3X)
waddwstr	curs_addwstr(3X)
wadjcurspos	curs_alecompat(3X)
wattroff	curs_attr(3X)
wattron	curs_attr(3X)
wattrset	curs_attr(3X)
wbkgd	curs_bkgd(3X)
wbkgdset	curs_bkgd(3X)
wborder	curs_border(3X)
wclear	curs_clear(3X)
wclrtoobot	curs_clear(3X)
wclrtoeol	curs_clear(3X)
wcursyncup	curs_window(3X)
wdelch	curs_delch(3X)
wdeleteln	curs_deleteln(3X)
wechochar	curs_addch(3X)
wechowchar	curs_addwch(3X)
werase	curs_clear(3X)
wgetch	curs_getch(3X)
wgetnstr	curs_getstr(3X)
wgetnwstr	curs_getwstr(3X)
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	 curs_refresh(3X)
wrefresh	 curs_refresh(3X)
wscanw	 curs_scanw(3X)
wscrl	 curs_scroll(3X)
wsetscreg	 curs_outopts(3X)
wstandend	 curs_attr(3X)
wstandout	 curs_attr(3X)
wsyncdown	 curs_window(3X)
wsyncup	 curs_window(3X)
wtimeout	 curs_inopts(3X)
wtouchln	 curs_touch(3X)
wvline	 curs_border(3X)

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.

SEE ALSO

terminfo(4) 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	cuserid – get character login name of the user
SYNOPSIS	#include <stdio.h> char *cuserid(char *s);
MT-LEVEL	MT-Safe
DESCRIPTION	cuserid() generates a character-string representation of the login name that the owner of the current process is logged in under. If <i>s</i> is a NULL pointer, this representation is generated in an internal static area, the address of which is returned. Otherwise, <i>s</i> is assumed to point to an array of at least L_cuserid characters; the representation is left in this array. The constant L_cuserid is defined in the <stdio.h> header.
RETURN VALUES	If the login name cannot be found, cuserid() returns a NULL pointer; if <i>s</i> is not a NULL pointer, a null character <code>`\0'</code> will be placed at <i>s</i> [0].
SEE ALSO	getlogin(3C) , getpwnam(3C)
NOTES	In multi-thread applications, the caller must always supply an array <i>s</i> for the return value.

NAME	dbm, dbmopen, dbmclose, fetch, store, delete, firstkey, nextkey – data base subroutines
SYNOPSIS	<pre> /usr/ucb/cc [flag ...] file ... -ldb #include <dbm.h> typedef struct { char *dptr; int dsize; } datum; int dbmopen(file) char *file; int dbmclose() datum fetch(key) datum key; int store(key, dat) datum key, dat; int delete(key) datum key; datum firstkey() datum nextkey(key) datum key; </pre>
DESCRIPTION	<p>The dbm() library has been superseded by ndbm(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 datum typedef. A datum 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 .dir as its suffix. The second file contains all data and has .pag 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 <i>file.dir</i> and <i>file.pag</i> must exist. An empty database is created by creating zero-length .dir and .pag 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>

for (key = firstkey; key.dptr != NULL; key = nextkey(key))

RETURN VALUES

All functions that return an **int** indicate errors with negative values. A zero return indicates no error. Routines that return a **datum** indicate errors with a NULL (0) *dptr*.

SEE ALSO

ar(1), **cat(1)**, **cp(1)**, **tar(1)**, **ndbm(3)**

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.

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.

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	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>
MT-LEVEL	MT-Safe
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> $(\text{pd->sign}) * (\text{pd->ds}) * 10^{(\text{pd->exponent})}$ <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> <p>strtod(3C), scanf(3S), fscanf(3S), and sscanf(3S) all use decimal_to_double().</p>
SEE ALSO	fscanf(3S) , scanf(3S) , sscanf(3S) , strtod(3C)

NAME	dial – establish an outgoing terminal line connection
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lnsl [<i>library</i> ...] #include <dial.h> int dial(CALL <i>call</i>); void undial(int <i>fd</i>);</pre>
MT-LEVEL	Unsafe
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 <dial.h>).</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 <dial.h> 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 speed 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 baud 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 line element in the CALL structure. Legal values for such terminal device names are kept in the Devices file. In this case, the value of the baud element should be set to -1. This value will cause dial to determine the correct value from the <Devices> file.</p> <p>The telno 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> <pre>0-9 dial 0-9 * dial * # dial # = wait for secondary dial tone - delay for approximately 4 seconds</pre>

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	<i>/* interrupt occurred */</i>
D_HUNG	-2	<i>/* dialer hung (no return from write) */</i>
NO_ANS	-3	<i>/* no answer within 10 seconds */</i>
ILL_BD	-4	<i>/* illegal baud-rate */</i>
A_PROB	-5	<i>/* acu problem (open() failure) */</i>
L_PROB	-6	<i>/* line problem (open() failure) */</i>
NO_Ldv	-7	<i>/* can't open Devices file */</i>
DV_NT_A	-8	<i>/* requested device not available */</i>
DV_NT_K	-9	<i>/* requested device not known */</i>
NO_BD_A	-10	<i>/* no device available at requested baud */</i>
NO_BD_K	-11	<i>/* no device known at requested baud */</i>
DV_NT_E	-12	<i>/* requested speed does not match */</i>
BAD_SYS	-13	<i>/* system not in Systems file*/</i>

FILES

`/etc/uucp/Devices`
`/etc/uucp/Systems`
`/var/spool/uucp/LCK..tty-device`

SEE ALSO

uucp(1C), **alarm(2)**, **read(2)**, **write(2)**, **termio(7)**

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	<code>difftime</code> – computes the difference between two calendar times
SYNOPSIS	<pre>#include <time.h> double difftime(time_t time1, time_t time0);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<code>difftime()</code> computes the difference between two calendar times. <code>difftime()</code> returns the difference ($time1-time0$) expressed in seconds as a double . This function is provided because there are no general arithmetic properties defined for type time_t .
SEE ALSO	<code>ctime(3C)</code>

NAME	directory, opendir, readdir, readdir_r, telldir, seekdir, rewinddir, closedir – directory operations
SYNOPSIS	<pre>#include <dirent.h> DIR *opendir(const char *filename); struct dirent *readdir(DIR *dirp); struct dirent *readdir_r(DIR *dirp, struct dirent *res); long telldir(DIR *dirp); void seekdir(DIR *dirp, long loc); void rewinddir(DIR *dirp); int closedir(DIR *dirp);</pre>
MT-LEVEL	See the NOTES section of this page.
DESCRIPTION	<p>opendir() opens the directory named by <i>filename</i> and associates a directory stream with it. opendir() returns a pointer to be used to identify the directory stream in subsequent operations. The directory stream is positioned at the first entry. A null pointer is returned if <i>filename</i> cannot be accessed or is not a directory, or if it cannot malloc(3C) enough memory to hold a DIR structure or a buffer for the directory entries.</p> <p>readdir() 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 filesystems. It returns a NULL pointer upon reaching the end of the directory stream, or upon detecting an invalid location in the directory.</p> <p>readdir_r() shall not return directory entries containing empty names. It is unspecified whether entries are returned for dot or dot-dot. The pointer returned by readdir_r() points to data that may be overwritten by another call to readdir_r() on the same directory stream. This data shall not be overwritten by another call to readdir_r() on a different directory stream. readdir_r() may buffer several directory entries per actual read operation; readdir_r() marks for update the <i>st_atime</i> field of the directory each time the directory is actually read.</p> <p>readdir_r_r() has the equivalent functionality as readdir_r() except that a buffer <i>res</i> must be supplied by the caller to store the result. To allocate <i>res</i> correctly a <i>struct dirent res</i> is not sufficient, thus the size should be <code>sizeof(struct dirent) + _POSIX_PATH_MAX</code> (defined in <code><limits.h></code>).</p> <p>telldir() returns the current location associated with the named directory stream.</p> <p>seekdir() sets the position of the next readdir_r() operation on the directory stream. The new position reverts to the position associated with the directory stream at the time the telldir() operation that provides <i>loc</i> was performed. Values returned by telldir() are good only for the lifetime of the DIR pointer from which they are derived. If the directory is closed and then reopened, the telldir() value may be invalidated due to undetected directory compaction. It is safe to use a previous telldir() value immediately after a call to opendir() and before any calls to readdir_r().</p>

rewinddir() resets the position of the named directory stream 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()** would.

closedir() closes the named directory stream and frees the **DIR** structure.

RETURN VALUES

opendir(), **readdir()**, and **readdir_r()** return **NULL** on failure and set **errno** to indicate the error. **telldir()**, **seekdir()**, and **closedir()** return **-1** on failure and set **errno** to indicate the error.

ERRORS

opendir() will fail if one or more of the following are true:

EACCES	Read permission is denied on the specified directory.
EFAULT	<i>filename</i> points outside the allocated address space.
ELOOP	Too many symbolic links were encountered in translating <i>filename</i> .
ENOTDIR	A component of <i>filename</i> is not a directory.
EMFILE	The maximum number of file descriptors are currently open.
ENFILE	The system file table is full.
ENAMETOOLONG	The length of the <i>filename</i> argument exceeds {PATH_MAX} , or the length of a <i>filename</i> component exceeds {NAME_MAX} while {_POSIX_NO_TRUNC} is in effect.
ENOENT	A component of <i>filename</i> does not exist or is a null pathname.

A component of *filename* denies search permission.

readdir() and **readdir_r()** will fail if one or more of the following are true:

EAGAIN	Mandatory file/record locking was set, O_NDELAY or O_NONBLOCK was set, and there was a blocking record lock.
EAGAIN	Total amount of system memory available when reading using raw I/O is temporarily insufficient.
EAGAIN	No data is waiting to be read on a file associated with a tty device and O_NONBLOCK was set.
EAGAIN	No message is waiting to be read on a stream and O_NDELAY or O_NONBLOCK was set.
EBADF	The file descriptor determined by the DIR stream is no longer valid. This results if the DIR stream has been closed.
EBADMSG	Message waiting to be read on a stream is not a data message.
EDEADLK	The read() was going to go to sleep and cause a deadlock to occur.
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** *fildev* is on a remote machine and the link to that machine is no longer active.
- ENXIO** The device associated with *fildev* is a block special or character special file and the value of the file pointer is out of range.

telldir(), **seekdir()**, and **closedir()** return 0 on success and will fail if one or more of the following are true:

- EBADF** The file descriptor determined by the **DIR** stream is no longer valid. This results if the **DIR** stream has been closed.

EXAMPLES

Here is a sample program that prints the names of all the files in the current directory:

```
#include <stdio.h>
#include <dirent.h>

main()
{
    DIR *dirp;
    struct dirent *direntp;

    dirp = opendir( "." );
    while ( (direntp = readdir( dirp )) != NULL )
        (void)printf( "%s\n", direntp->d_name );
    (void)closedir( dirp );
    return (0);
}
```

SEE ALSO

getdents(2), **dirent(4)**

NOTES

The **readdir_r()** 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.

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.

readdir() is unsafe in multi-thread applications. **readdir_r()** is safe, and should be used instead. **closedir()**, **directory()**, **opendir()**, **rewinddir()**, **seekdir()**, and **telldir()** are safe in multi-thread applications.

NAME	dirname – report the parent directory name of a file path name														
SYNOPSIS	<pre>cc [flag ...] file ... -lgen [library ...] #include <libgen.h> char *dirname(char *path);</pre>														
MT-LEVEL	MT-Safe														
DESCRIPTION	<p>Given a pointer to a null-terminated character string that contains a file system path name, dirname() returns a string that is the parent directory of that file. In doing this, it may place a null byte in the path name after the next to last element, so the content of <i>path</i> must be disposable. The returned string should not be deallocated by the caller. Trailing “/” characters in the path are not counted as part of the path.</p> <p>If <i>path</i> or <i>*path</i> is zero, a pointer to a static constant “.” is returned.</p> <p>dirname() and basename() together yield a complete path name. dirname (path) is the directory where basename (path) is found.</p>														
EXAMPLES	<p>A simple file name and the strings “.” and “..” all have “.” as their return value.</p> <table border="0" style="margin-left: 40px;"> <thead> <tr> <th style="text-align: left;">Input string</th> <th style="text-align: left;">Output pointer</th> </tr> </thead> <tbody> <tr> <td>/usr/lib</td> <td>/usr</td> </tr> <tr> <td>/usr/</td> <td>/</td> </tr> <tr> <td>usr</td> <td>.</td> </tr> <tr> <td>/</td> <td>/</td> </tr> <tr> <td>.</td> <td>.</td> </tr> <tr> <td>..</td> <td>.</td> </tr> </tbody> </table> <p>The following code 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 pointer	/usr/lib	/usr	/usr/	/	usr	.	/	/
Input string	Output pointer														
/usr/lib	/usr														
/usr/	/														
usr	.														
/	/														
.	.														
..	.														
SEE ALSO	basename(1) , chdir(2) , basename(3G)														
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	div, ldiv, lldiv – compute the quotient and remainder
SYNOPSIS	<pre>#include <stdlib.h> <div_t denom);="" denom);<="" div(int="" int="" ldiv(long="" ldiv_t="" lldiv(long="" lldiv_t="" long="" numer,="" pre=""> </div_t></pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>div() computes the quotient and remainder of the division of the numerator <i>numer</i> by the denominator <i>denom</i>. This function 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>ldiv() and lldiv() are similar to div(), except that the arguments and the members of the returned structure are different. ldiv() returns a structure of type ldiv_t and has type long int. lldiv() returns a structure of type lldiv_t and has type long long.</p>
RETURN VALUES	<p>div() returns a structure of type div_t, comprising both the quotient and remainder:</p> <pre>int quot; /*quotient*/ int rem; /*remainder*/</pre> <p>ldiv() returns a structure of type ldiv_t and lldiv() returns a structure of type lldiv_t, comprising both the quotient and remainder:</p> <pre>long int quot; /*quotient*/ long int rem; /*remainder*/</pre>

NAME	dladdr – translate address to symbolic information.
SYNOPSIS	<pre>cc [flag ...] file ... -ldl [library ...] #include <dlfcn.h> int dladdr(void * address, DL_info * dli);</pre>
MT-LEVEL	MT-Safe
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 -ldl is passed to the link-editor.</p> <p>These routines are available to dynamically linked processes ONLY.</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 <i>_end</i> 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 DL_info structure must be preallocated by the user. The structure members are filled in by dladdr() based on the specified <i>address</i>. The DL_info 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>dli_fname Contains a pointer to the filename of the containing object.</p> <p>dli_fbase Contains the base address of the containing object.</p> <p>dli_sname Contains a pointer to the nearest symbol name to the specified address. This symbol either has the same address, or is the nearest symbol with a lower address.</p> <p>dli_saddr Contains the actual address of the above symbol.</p>
RETURN VALUES	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 DL_info elements are filled.
SEE ALSO	ld(1), dlclose(3X), dlerror(3X), dlopen(3X), dlsym(3X) <i>Linker and Libraries Guide</i>
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 [<i>flag</i> ...] <i>file</i> ... -ldl [<i>library</i> ...] #include <dlfcn.h> int dlclose(void *<i>handle</i>);</pre>
MT-LEVEL	MT-Safe
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 -ldl is passed to the link-editor.</p> <p>These routines are available to dynamically linked processes ONLY.</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	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-0 value. More detailed diagnostic information will be available through dlderror() .
SEE ALSO	ld(1), dladdr(3X), dlderror(3X), dlopen(3X), dlsym(3X) <i>Linker and Libraries Guide</i>
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	dlerror – get diagnostic information
SYNOPSIS	<pre>cc [flag ...] file ... -ldl [library ...] #include <dlfcn.h> char *dlerror(void);</pre>
MT-LEVEL	MT-Safe
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 -ldl is passed to the link-editor.</p> <p>These routines are available to dynamically linked processes ONLY.</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 NULL. Thus, invoking dlerror() a second time, immediately following a prior invocation, will result in NULL being returned.</p>
SEE ALSO	ld(1), dladdr(3X), dlclose(3X), dlopen(3X), dlsym(3X) <i>Linker and Libraries Guide</i>
NOTES	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.

NAME	dlopen – open a shared object
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -ldl [<i>library</i> ...] #include <dlfcn.h> void *dlopen(const char *pathname, int mode);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>dlopen() 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 -ldl is passed to the link-editor.</p> <p>These routines are available to dynamically linked processes ONLY.</p> <p>dlopen() makes a shared object available to a running process. dlopen() returns to the process a "handle" which the process may use on subsequent calls to dlsym() and dlclose(). This value 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 '/' will be interpreted as an absolute path or relative to the current directory, otherwise the set of search paths currently in effect by the run-time linker will be used to locate the specified file (SEE <i>NOTES</i> section below). If the value of <i>pathname</i> is 0, dlopen() will make the symbols contained in the original a.out, and all of the objects that were loaded at program startup with the a.out, available through dlsym().</p> <p>When a shared object is brought into the address space of a process, it may contain references to symbols whose addresses are not known until the object is loaded. These references must be relocated before the symbols can be accessed. The <i>mode</i> parameter governs when these relocations take place and may have the following values:</p> <p>RTLD_LAZY Under this <i>mode</i>, only references to data symbols are relocated when the object is loaded. References to functions are not relocated until a given function is invoked for the first time. This <i>mode</i> should result in better performance, since a process may not reference all of the functions in any given shared object. This behavior mimics the normal loading of shared object dependencies by a dynamic executable during process initialization.</p> <p>RTLD_NOW Under this <i>mode</i>, all necessary relocations are performed when the object is first loaded. This may result in some wasted effort, if relocations are performed for functions that are never referenced, but is useful for applications that need to know as soon as an object is loaded that all symbols referenced during execution will be available.</p>

Objects loaded by a single invocation of **dlopen()** may import symbols from one another or from any object loaded automatically during program startup, but objects loaded by one **dlopen()** invocation may not directly reference symbols from objects loaded by a different **dlopen()** invocation. Those symbols may, however, be referenced indirectly using **dlsym()**. However, the *mode* parameter may also be **ored** with the following value to effect the scope of symbol availability:

RTLD_GLOBAL Under this *mode*, the objects symbols will be made available for all to access.

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()**.

SEE ALSO

ld(1), **dladdr(3X)**, **dlclose(3X)**, **dlderror(3X)**, **dlsym(3X)**

Linker and Libraries Guide

NOTES

If other shared objects were link edited with *pathname* when *pathname* was built (ie. the *pathname* has dependencies on other shared objects), those objects will automatically be loaded by **dlopen()**. The directory search path that will be used to find both *pathname* and the other *needed* objects may be effected by setting the environment variable **LD_LIBRARY_PATH**, or from a run-path setting within the application or the shared object from which the **dlopen()** originated. These search rules will only be applied to pathnames 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.

Users who wish to gain access to the symbol table of the **a.out** itself using **dlopen(0, mode)** should be aware that some symbols defined in the **a.out** 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 **a.out**: specifically those referenced by the shared objects with which the **a.out** is linked.

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 *handle, const char *name);</pre>
MT-LEVEL	MT-Safe
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 -ldl is passed to the link-editor.</p> <p>These routines are available to dynamically linked processes ONLY.</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 the special flag RTLD_NEXT. <i>name</i> is the symbol's name as a character string.</p> <p>In the former case 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()).</p> <p>In the latter case dlsym() will search for the named symbol in the objects that were loaded following the object from which the dlsym() call is being made. If these subsequent objects were loaded from dlopen() calls, dlsym() will search the object only if the caller is part of the same dlopen() dependency hierarchy, or the object was given global search access (see dlopen() with reference to the mode RTLD_GLOBAL).</p>
RETURN VALUES	If <i>handle</i> does not refer to a valid object opened by dlopen() , is not the special flag RTLD_NEXT , or if the named symbol cannot be found within any of the objects associated with <i>handle</i> , dlsym() will return NULL . More detailed diagnostic information will be available through dlerror() .
EXAMPLES	<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> <pre>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);</pre>

SEE ALSO

ld(1), dladdr(3X), dlclose(3X), dlerror(3X), dlopen(3X)

Linker and Libraries Guide

NAME	doconfig – execute a configuration script
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lnsl [<i>library</i> ...] # include <sac.h> int doconfig(int <i>fildev</i>, char *<i>script</i>, long <i>rflag</i>);</pre>
MT-LEVEL	Unsafe
DESCRIPTION	<p>doconfig() is a Service Access Facility library function that interprets the configuration scripts contained in the files <code></etc/saf/pmtag/_config></code>, <code></etc/saf/_sysconfig></code>, and <code></etc/saf/pmtag/svctag></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 pmadm(1M) and sacadm(1M).</p> <p><i>script</i> is the name of the configuration script; <i>fildev</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 assign command is considered illegal and will generate an error return. If <i>rflag</i> has the NORUN bit set, the run and runwait 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: assign, push, pop, runwait, and run. The comment character is #; when a # 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>assign <i>variable=value</i> 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. assign will fail if space cannot be allocated for the new variable or if any part of the specification is invalid.</p> <p>push <i>module1</i> [, <i>module2</i>, <i>module3</i>, ...] Used to push STREAMS modules onto the stream designated by <i>fildev</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> <p>pop [<i>module</i>] 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,</p>

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 top-most 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.

SEE ALSO

sh(1), **pmadm(1M)**, **sacadm(1M)**

NOTES

This interface is unsafe in multithreaded applications. Unsafe interfaces should be called only from the main thread.

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 xsubi[3]); long lrand48(void); long nrand48(unsigned short xsubi[3]); long mrand48(void); long jrand48(unsigned short xsubi[3]); void srand48(long seedval); unsigned short *seed48(unsigned short seed16v[3]); void lcong48(unsigned short param[7]);</pre>
MT-LEVEL	Safe
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 lrnd48() 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(), lrnd48(), or mrnd48() is called. (Although it is not recommended practice, constant default initializer values will be supplied automatically if drand48(), lrnd48(), 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> $X_{n+1} = (aX_n + c)_{\text{mod } m} \quad n \geq 0.$ <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()**, **mrnd48()**, 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 **mrnd48()** 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 **srnd48()** 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 **lcng48()** 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 **lcng48()** has been called, a subsequent call to either **srnd48()** or **seed48()** will restore the “standard” multiplier and addend values, a and c , specified above.

SEE ALSO **rand(3C)**

NAME	dup2 – duplicate an open file descriptor
SYNOPSIS	<pre>#include <unistd.h> int dup2(int <i>filde</i>, int <i>filde2</i>);</pre>
MT-LEVEL	Unsafe
DESCRIPTION	dup2() causes the file descriptor <i>filde2</i> to refer to the same file as <i>filde</i> . <i>filde</i> is a file descriptor referring to an open file, and <i>filde2</i> 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 <i>filde2</i> already referred to an open file, not <i>filde</i> , it is closed first. If <i>filde2</i> refers to <i>filde</i> , or if <i>filde</i> is not a valid open file descriptor, <i>filde2</i> will not be closed first.
RETURN VALUES	Upon successful completion a non-negative integer, namely, the file descriptor, is returned. Otherwise, a value of -1 is returned and errno is set to indicate the error.
ERRORS	dup2() will fail if one or more of the following are true: EBADF <i>filde</i> is not a valid open file descriptor. EINTR a signal was caught during the dup2() call. EMFILE the process has too many open files (see fcntl(2)).
SEE ALSO	close(2) , creat(2) , exec(2) , fcntl(2) , getrlimit(2) , open(2) , pipe(2) , lockf(3C)

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>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>econvert() 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>fconvert works much like econvert, except that the correct digit has been rounded as if for sprintf(%w.nf) output with <i>n=ndigit</i> 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>gconvert() 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 sprintf(%w.nf), if possible, and otherwise in floating-decimal format, like sprintf(%w.ne); in either case <i>buf</i> is ready for printing, with sign and exponent. The result corresponds to that obtained by</p> <pre>(void) sprintf(buf, "%w.ng", value);</pre> <p>If <i>trailing</i> = 0, trailing zeros and a trailing point are suppressed, as in sprintf(%g). If <i>trailing</i> != 0, trailing zeros and a trailing point are retained, as in sprintf(%#g).</p>

seconvert, **sfconvert**, and **sgconvert()** 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.

qeconvert, **qfconvert**, and **qgconvert()** are quadruple-precision versions of these functions. **qfconvert()** can overflow the *decimal_record* field *ds* if *value* is too large. In that case, *buf[0]* is set to zero.

ecvt() and **fcvt()** are obsolete 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.

gcvf() is an obsolete 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* \geq 8.

SEE ALSO**sprintf(3S)**

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>
MT-LEVEL	Unsafe
DESCRIPTION	<p>ecvt() converts <i>value</i> to a null-terminated string of <i>ndigit</i> digits and returns a pointer thereto. The high-order digit is non-zero, unless the value is zero. The low-order digit is rounded. The position of the decimal point relative to the beginning of the string is stored indirectly through <i>decpt</i> (negative means to the left of the returned digits). The decimal point is not included in the returned string. If the sign of the result is negative, the word pointed to by <i>sign</i> is non-zero, otherwise it is zero.</p> <p>fcvt() is identical to ecvt(), except that the correct digit has been rounded for printf %f output of the number of digits specified by <i>ndigit</i>.</p> <p>gcvt() converts the <i>value</i> to a null-terminated string in the array pointed to by <i>buf</i> and returns <i>buf</i>. It attempts to produce <i>ndigit</i> significant digits in %f format if possible, otherwise %e format (scientific notation), ready for printing. A minus sign, if there is one, or a decimal point will be included as part of the returned string. Trailing zeros are suppressed.</p>
SEE ALSO	printf(3S)
NOTES	The values returned by ecvt() and fcvt() point to a single static data array whose content is overwritten by each call.

NAME	elf – object file access library
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lelf [<i>library</i> ...] #include <libelf.h></pre>
MT-LEVEL	Unsafe
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, elf_begin(3E) 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 elf_getscn(3E)). 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	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, etc., as in the following.

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

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, etc.) 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.

elf_name	These class-independent names perform some service, <i>name</i> , for the program.
elf32_name	Service names with an embedded class, 32 here, indicate they work only for the designated class of files.
Elf_Type	Data types can be class-independent as well, distinguished by <i>Type</i> .
Elf32_Type	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 Elf_Cmd enumeration; they range from zero through ELF_C_NUM-1 .
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 elf32_fsize() for more information.
ELF_K_KIND	The function elf_kind() identifies the <i>KIND</i> of file associated with an ELF descriptor. These values are members of the Elf_Kind

enumeration; they range from zero through `ELF_K_NUM-1`.

ELF_T_TYPE When a service function, such as `elf32_xlatetom()` or `elf32_xlatetof()`, deals with multiple types, names of this form specify the desired *TYPE*. Thus, for example, `ELF_T_EHDR` is directly related to `Elf32_Ehdr`. These values are members of the `Elf_Type` enumeration; they range from zero through `ELF_T_NUM-1`.

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_rawfile(3E)`, `elf_strptr(3E)`, `elf_update(3E)`, `elf_version(3E)`, `ar(4)`

ANSI C Programmer's Guide

SPARC only
x86 only

`a.out(4)`

`cof2elf(1)`

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
486	Intel 80486
860	Intel 80860
68K	Motorola 68000
88K	Motorola 88000

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.

```
#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.

NAME	elf32_fsize – return the size of an object file type																					
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lelf [<i>library</i> ...] #include <libelf.h> size_t elf32_fsize(Elf_Type <i>type</i>, size_t <i>count</i>, unsigned <i>ver</i>);</pre>																					
MT-LEVEL	Unsafe																					
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> <table border="0" style="margin-left: 40px;"> <thead> <tr> <th style="text-align: left;">Elf_Type</th> <th style="text-align: left;">File Size</th> <th style="text-align: left;">Memory Size</th> </tr> </thead> <tbody> <tr> <td>ELF_T_ADDR</td> <td>ELF32_FSZ_ADDR</td> <td>sizeof(Elf32_Addr)</td> </tr> <tr> <td>ELF_T_BYTE</td> <td>1</td> <td>sizeof(unsigned char)</td> </tr> <tr> <td>ELF_T_HALF</td> <td>ELF32_FSZ_HALF</td> <td>sizeof(Elf32_Half)</td> </tr> <tr> <td>ELF_T_OFF</td> <td>ELF32_FSZ_OFF</td> <td>sizeof(Elf32_Off)</td> </tr> <tr> <td>ELF_T_SWORD</td> <td>ELF32_FSZ_SWORD</td> <td>sizeof(Elf32_Sword)</td> </tr> <tr> <td>ELF_T_WORD</td> <td>ELF32_FSZ_WORD</td> <td>sizeof(Elf32_Word)</td> </tr> </tbody> </table> <p>elf32_fsize() returns zero 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>	Elf_Type	File Size	Memory Size	ELF_T_ADDR	ELF32_FSZ_ADDR	sizeof(Elf32_Addr)	ELF_T_BYTE	1	sizeof(unsigned char)	ELF_T_HALF	ELF32_FSZ_HALF	sizeof(Elf32_Half)	ELF_T_OFF	ELF32_FSZ_OFF	sizeof(Elf32_Off)	ELF_T_SWORD	ELF32_FSZ_SWORD	sizeof(Elf32_Sword)	ELF_T_WORD	ELF32_FSZ_WORD	sizeof(Elf32_Word)
Elf_Type	File Size	Memory Size																				
ELF_T_ADDR	ELF32_FSZ_ADDR	sizeof(Elf32_Addr)																				
ELF_T_BYTE	1	sizeof(unsigned char)																				
ELF_T_HALF	ELF32_FSZ_HALF	sizeof(Elf32_Half)																				
ELF_T_OFF	ELF32_FSZ_OFF	sizeof(Elf32_Off)																				
ELF_T_SWORD	ELF32_FSZ_SWORD	sizeof(Elf32_Sword)																				
ELF_T_WORD	ELF32_FSZ_WORD	sizeof(Elf32_Word)																				
SEE ALSO	elf(3E), elf32_xlatetof(3E), elf_version(3E)																					

NAME	elf32_getehdr, elf32_newehdr – retrieve class-dependent object file header
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lelf [<i>library</i> ...] #include <libelf.h> Elf32_Ehdr *elf32_getehdr(Elf *<i>elf</i>); Elf32_Ehdr *elf32_newehdr(Elf *<i>elf</i>);</pre>
MT-LEVEL	Unsafe
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>The header includes the following members.</p> <pre> unsigned char e_ident[EI_NIDENT]; Elf32_Half e_type; Elf32_Half e_machine; Elf32_Word e_version; Elf32_Addr e_entry; Elf32_Off e_phoff; Elf32_Off e_shoff; Elf32_Word e_flags; Elf32_Half e_ehsize; Elf32_Half e_phentsize; Elf32_Half e_phnum; Elf32_Half e_shentsize; Elf32_Half e_shnum; Elf32_Half e_shstrndx;</pre> <p>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.</p>
SEE ALSO	elf(3E) , elf_begin(3E) , elf_flagdata(3E) , elf_getident(3E)

NAME	elf32_getphdr, elf32_newphdr – retrieve class-dependent program header table																
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lelf [<i>library</i> ...] #include <libelf.h> Elf32_Phdr *elf32_getphdr(Elf *elf); Elf32_Phdr *elf32_newphdr(Elf *elf, size_t count);</pre>																
MT-LEVEL	Unsafe																
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 ELF_F_DIRTY 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() (see elf32_getehdr(3E)), allowing a program to replace or delete the program header table, changing its size if necessary.</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 null, both functions return a null pointer. Additionally, elf32_newphdr() returns a null pointer if <i>count</i> is zero.</p> <p>The table is an array of Elf32_Phdr structures, each of which includes the following members.</p> <table border="0" style="margin-left: 40px;"> <tr><td>Elf32_Word</td><td>p_type;</td></tr> <tr><td>Elf32_Off</td><td>p_offset;</td></tr> <tr><td>Elf32_Addr</td><td>p_vaddr;</td></tr> <tr><td>Elf32_Addr</td><td>p_paddr;</td></tr> <tr><td>Elf32_Word</td><td>p_filesz;</td></tr> <tr><td>Elf32_Word</td><td>p_memsz;</td></tr> <tr><td>Elf32_Word</td><td>p_flags;</td></tr> <tr><td>Elf32_Word</td><td>p_align;</td></tr> </table> <p>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 <i>count</i>. If the program is building a new file, it is responsible for creating the file's ELF header before creating the program header table.</p>	Elf32_Word	p_type;	Elf32_Off	p_offset;	Elf32_Addr	p_vaddr;	Elf32_Addr	p_paddr;	Elf32_Word	p_filesz;	Elf32_Word	p_memsz;	Elf32_Word	p_flags;	Elf32_Word	p_align;
Elf32_Word	p_type;																
Elf32_Off	p_offset;																
Elf32_Addr	p_vaddr;																
Elf32_Addr	p_paddr;																
Elf32_Word	p_filesz;																
Elf32_Word	p_memsz;																
Elf32_Word	p_flags;																
Elf32_Word	p_align;																
SEE ALSO	elf(3E) , elf32_getehdr(3E) , elf_begin(3E) , elf_flagdata(3E)																

NAME	elf32_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>);</pre>																				
MT-LEVEL	Unsafe																				
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 header includes the following members.</p> <table border="0" style="margin-left: 40px;"> <tr><td>Elf32_Word</td><td>sh_name;</td></tr> <tr><td>Elf32_Word</td><td>sh_type;</td></tr> <tr><td>Elf32_Word</td><td>sh_flags;</td></tr> <tr><td>Elf32_Addr</td><td>sh_addr;</td></tr> <tr><td>Elf32_Off</td><td>sh_offset;</td></tr> <tr><td>Elf32_Word</td><td>sh_size;</td></tr> <tr><td>Elf32_Word</td><td>sh_link;</td></tr> <tr><td>Elf32_Word</td><td>sh_info;</td></tr> <tr><td>Elf32_Word</td><td>sh_addralign;</td></tr> <tr><td>Elf32_Word</td><td>sh_entsize;</td></tr> </table> <p>If the program is building a new file, it is responsible for creating the file's ELF header before creating sections.</p>	Elf32_Word	sh_name;	Elf32_Word	sh_type;	Elf32_Word	sh_flags;	Elf32_Addr	sh_addr;	Elf32_Off	sh_offset;	Elf32_Word	sh_size;	Elf32_Word	sh_link;	Elf32_Word	sh_info;	Elf32_Word	sh_addralign;	Elf32_Word	sh_entsize;
Elf32_Word	sh_name;																				
Elf32_Word	sh_type;																				
Elf32_Word	sh_flags;																				
Elf32_Addr	sh_addr;																				
Elf32_Off	sh_offset;																				
Elf32_Word	sh_size;																				
Elf32_Word	sh_link;																				
Elf32_Word	sh_info;																				
Elf32_Word	sh_addralign;																				
Elf32_Word	sh_entsize;																				
SEE ALSO	elf(3E) , elf_flagdata(3E) , elf_getscn(3E) , elf_strptr(3E)																				

NAME	elf32_xlatetof, elf32_xlatetom – class-dependent data translation								
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lelf [<i>library</i> ...] #include <libelf.h> Elf_Data *elf32_xlatetof(Elf_Data *<i>dst</i>, const Elf_Data *<i>src</i>, unsigned <i>encode</i>); Elf_Data *elf32_xlatetom(Elf_Data *<i>dst</i>, const Elf_Data *<i>src</i>, unsigned <i>encode</i>);</pre>								
MT-LEVEL	Unsafe								
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 e_ident[EI_DATA] 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 Elf_Data descriptor, which the translation routines use as follows.</p> <table border="0"> <tr> <td style="vertical-align: top;">d_buf</td> <td>Both the source and destination must have valid buffer pointers.</td> </tr> <tr> <td style="vertical-align: top;">d_type</td> <td>This member's value specifies the type of the data to which d_buf points and the type of data to be created in the destination. The program supplies a d_type value in the source; the library sets the destination's d_type to the same value. These values are summarized below.</td> </tr> <tr> <td style="vertical-align: top;">d_size</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 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.</td> </tr> <tr> <td style="vertical-align: top;">d_version</td> <td>This member holds version number of the objects (desired) in the buffer. The source and destination versions are independent.</td> </tr> </table> <p>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.</p>	d_buf	Both the source and destination must have valid buffer pointers.	d_type	This member's value specifies the type of the data to which d_buf points and the type of data to be created in the destination. The program supplies a d_type value in the source; the library sets the destination's d_type to the same value. These values are summarized below.	d_size	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 version number of the objects (desired) in the buffer. The source and destination versions are independent.
d_buf	Both the source and destination must have valid buffer pointers.								
d_type	This member's value specifies the type of the data to which d_buf points and the type of data to be created in the destination. The program supplies a d_type value in the source; the library sets the destination's d_type to the same value. These values are summarized below.								
d_size	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 version number of the objects (desired) in the buffer. The source and destination versions are independent.								

Elf_Type	32-Bit Memory Type
ELF_T_ADDR	Elf32_Addr
ELF_T_BYTE	unsigned char
ELF_T_DYN	Elf32_Dyn
ELF_T_EHDR	Elf32_Ehdr
ELF_T_HALF	Elf32_Half
ELF_T_OFF	Elf32_Off
ELF_T_PHDR	Elf32_Phdr
ELF_T_REL	Elf32_Rel
ELF_T_RELA	Elf32_Rela
ELF_T_SHDR	Elf32_Shdr
ELF_T_SWORD	Elf32_Sword
ELF_T_SYM	Elf32_Sym
ELF_T_WORD	Elf32_Word

Translating buffers of type **ELF_T_BYTE** does not change the byte order.

SEE ALSO [elf\(3E\)](#), [elf32_fsize\(3E\)](#), [elf_getdata\(3E\)](#), [elf_getident\(3E\)](#)

NAME	elf_begin, elf_end, 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_Cmd elf_next(Elf *elf); size_t elf_rand(Elf *elf, size_t offset);</pre>
MT-LEVEL	Unsafe
DESCRIPTION	<p>elf_begin(), elf_end(), 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(), 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> <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.</p> <p>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. 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 the next call to elf_begin(), as if the program had used elf_next() or elf_rand() to “move” to the initial member.</p> <p>Second, if <i>ref</i> 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 <i>ref</i> appropriately (except for the initial member, which elf_begin() prepares; see the example below). In this case, <i>fildes</i> should be the same file descriptor used for the parent archive.</p>

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 zero. Consequently, a 0 return value indicates the ELF descriptor is no longer valid.

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

A prototype for reading a file appears below. 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 two, 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)
    {
        /* library out of date */
        /* recover from error */
    }
    cmd = ELF_C_READ;
    arf = elf_begin(fildes, cmd, (Elf *)0);
    while ((elf = elf_begin(fildes, cmd, arf)) != 0)
    {
        if ((ehdr = elf32_getehdr(elf)) != 0)
        {
            /* process the file ... */
        }
        cmd = elf_next(elf);
        elf_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)
{
    /* not an archive */
}
/* initial processing */
/* set offset = ... for desired member header */
while (elf_rand(arf, offset) == offset)
{
    if ((elf = elf_begin(fildes, ELF_C_READ, arf)) == 0)
        break;
    if ((ehdr = elf32_getehdr(elf)) != 0)
    {
        /* process archive member ... */
    }
    /* 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)
    return;
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 ... */

close(creat("path/name", 0666));
elf_update(elf, ELF_C_WRITE);
elf_end(elf);
```

In the example above, the call to **creat** truncates the file, thus ensuring the resulting file will have the “right” size. Without truncation, the updated file might be as big as the original, even if information were deleted. The library truncates the file, if it can, with **truncate** (see **truncate**(3C)). Some systems, however, do not support **truncate**, and the call to **creat** protects against this.

Notice that both file creation examples open the file with write *and* read permissions. On systems that support **mmap**, the library uses it to enhance performance, and **mmap** 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**.

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), **truncate**(3C), **ar**(4)

NAME	elf_cntl – control an elf file descriptor
SYNOPSIS	<pre>cc [flag ...] file ... -lelf [library ...] #include <libelf.h> int elf_cntl(Elf *elf, Elf_Cmd cmd);</pre>
MT-LEVEL	Unsafe
DESCRIPTION	<p>elf_cntl() instructs the library to modify its behavior with respect to an ELF descriptor, <i>elf</i>. 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 <i>elf</i>. 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.</p> <p>The <i>cmd</i> argument tells what actions to take and may have the following values.</p> <p>ELF_C_FDDONE This value tells the library not to use the file descriptor associated with <i>elf</i>. 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.</p> <p>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.</p> <p>If elf_cntl() succeeds, it returns zero. Otherwise <i>elf</i> was null or an error occurred, and the function returns -1.</p>
SEE ALSO	elf(3E) , elf_begin(3E) , elf_getdata(3E) , elf_rawfile(3E)
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 [<i>flag</i> ...] <i>file</i> ... -l<code>elf</code> [<i>library</i> ...] #include <libelf.h> const char *elf_errmsg (int <i>err</i>); int elf_errno(void);</pre>
MT-LEVEL	Unsafe
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 zero, 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>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 zero.</p> <pre>(void)elf_errno(); /* processing ... */ while (more_to_do) { if ((err = elf_errno()) != 0) { /* print msg */ msg = elf_errmsg(err); } }</pre>
SEE ALSO	elf(3E)

NAME	elf_fill – set fill byte
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lelf [<i>library</i> ...] #include <libelf.h> void elf_fill(int <i>fill</i>);</pre>
MT-LEVEL	Unsafe
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 <i>fill</i> character. The library uses zero bytes unless the application supplies a value. See elf_getdata(3E) for more information about these holes.
SEE ALSO	elf(3E) , elf_flagdata(3E) , elf_getdata(3E) , elf_update(3E)
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 <i>not</i> fill holes.

NAME	elf_flagdata, elf_flagehdr, elf_flagelf, elf_flagphdr, elf_flagscn, elf_flagshdr – manipulate flags
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lelf [<i>library</i> ...] #include <libelf.h> unsigned elf_flagdata(Elf_Data *<i>data</i>, Elf_Cmd <i>cmd</i>, unsigned <i>flags</i>); unsigned elf_flagehdr(Elf *<i>elf</i>, Elf_Cmd <i>cmd</i>, unsigned <i>flags</i>); unsigned elf_flagelf(Elf *<i>elf</i>, Elf_Cmd <i>cmd</i>, unsigned <i>flags</i>); unsigned elf_flagphdr(Elf *<i>elf</i>, Elf_Cmd <i>cmd</i>, unsigned <i>flags</i>); unsigned elf_flagscn(Elf_Scn *<i>scn</i>, Elf_Cmd <i>cmd</i>, unsigned <i>flags</i>); unsigned elf_flagshdr(Elf_Scn *<i>scn</i>, Elf_Cmd <i>cmd</i>, unsigned <i>flags</i>);</pre>
MT-LEVEL	Unsafe
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 zero 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> <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.</p>

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

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);
```

SEE ALSO

`elf(3E)`, `elf32_getehdr(3E)`, `elf_getdata(3E)`, `elf_update(3E)`

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>
MT-LEVEL	Unsafe
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; long ar_uid; long ar_gid; unsigned long ar_mode; off_t ar_size; char *ar_rawname;</pre> <p>An archive member name, available through ar_name, is a null-terminated string, with the ar format control characters removed. The ar_rawname 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 (ar_name) defined below.</p> <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> <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 ar_name 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 ar_name to a null pointer, the ar_rawname member will be set as usual.</p>
SEE ALSO	elf(3E) , elf_begin(3E) , elf_getarsym(3E) , ar(4)

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>
MT-LEVEL	Unsafe
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 zero 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>
SEE ALSO	elf(3E) , elf_begin(3E) , elf_getarhdr(3E) , elf_hash(3E) , ar(4)

NAME	elf_getbase – get the base offset for an object file
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lelf [<i>library</i> ...] #include <libelf.h> off_t elf_getbase(Elf *<i>elf</i>);</pre>
MT-LEVEL	Unsafe
DESCRIPTION	elf_getbase() returns the file offset of the first byte of the file or archive member associated with <i>elf</i> , if it is known or obtainable, and -1 otherwise. A null <i>elf</i> 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, <i>not</i> the beginning of the archive member header.
SEE ALSO	elf(3E) , elf_begin(3E) , ar(4)

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>
MT-LEVEL	Unsafe
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 (d_type below) is set to ELF_T_BYTE, and the version (d_version 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 ELF_F_DIRTY 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> <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 <i>elf</i> 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 <i>scn</i> is null or an error occurs, elf_rawdata() returns a null pointer.</p>

The **Elf_Data** structure includes the following members:

```

void          *d_buf;
Elf_Type      d_type;
size_t        d_size;
off_t         d_off;
size_t        d_align;
unsigned      d_version;

```

These members are available for direct manipulation by the program. Descriptions appear below.

d_buf	A pointer to the data buffer resides here. A data element with no data has a null pointer.
d_type	This member's value specifies the type of the data to which d_buf points. A section's type determines how to interpret the section contents, as summarized below.
d_size	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 SHT_NOBITS below for more information.)
d_off	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.
d_align	This member holds the buffer's required alignment, from the beginning of the section. That is, d_off will be a multiple of this member's value. For example, if this member's value is four, the beginning of the buffer will be four-byte aligned within 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.
d_version	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.

DATA ALIGNMENT

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.

First, the program can use **elf_fill()** 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 zero, and **elf_fill()** lets the application change that.

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.

**SECTION AND
MEMORY TYPES**

elf_getdata() interprets sections' data according to the section type, as noted in the section header available through **elf32_getshdr()**. 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 **elf32_xlatetof(3E)**.

Section Type	Elf_Type	32-Bit Type
SHT_DYNAMIC	ELF_T_DYN	Elf32_Dyn
SHT_DYNSYM	ELF_T_SYM	Elf32_Sym
SHT_HASH	ELF_T_WORD	Elf32_Word
SHT_NOBITS	ELF_T_BYTE	unsigned char
SHT_NOTE	ELF_T_BYTE	unsigned char
SHT_NULL	<i>none</i>	<i>none</i>
SHT_PROGBITS	ELF_T_BYTE	unsigned char
SHT_REL	ELF_T_REL	Elf32_Rel
SHT_RELA	ELF_T_RELA	Elf32_Rela
SHT_STRTAB	ELF_T_BYTE	unsigned char
SHT_SYMTAB	ELF_T_SYM	Elf32_Sym
<i>other</i>	ELF_T_BYTE	unsigned 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

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)
{
    /* not a string table */
}
data = 0;
if ((data = elf_getdata(scn, data)) == 0 || data->d_size == 0)
{
    /* 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.

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)

NAME elf_getident – retrieve file identification data

SYNOPSIS `cc [flag ...] file ... -lelf [library ...]`
`#include <libelf.h>`
`char *elf_getident(Elf *elf, size_t *ptr);`

MT-LEVEL Unsafe

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.

<code>e_ident</code> Index	Value	Purpose
<code>EI_MAG0</code>	<code>ELFMAG0</code>	File identification
<code>EI_MAG1</code>	<code>ELFMAG1</code>	
<code>EI_MAG2</code>	<code>ELFMAG2</code>	
<code>EI_MAG3</code>	<code>ELFMAG3</code>	
<code>EI_CLASS</code>	<code>ELFCLASSNONE</code> <code>ELFCLASS32</code> <code>ELFCLASS64</code>	File class
<code>EI_DATA</code>	<code>ELFDATANONE</code> <code>ELFDATA2LSB</code> <code>ELFDATA2MSB</code>	Data encoding
<code>EI_VERSION</code>	<code>EV_CURRENT</code>	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 zero stored through `ptr`, if `ptr` is non-null.

SEE ALSO

elf(3E), elf32_getehdr(3E), elf_begin(3E), elf_kind(3E), elf_rawfile(3E)

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>
MT-LEVEL	Unsafe
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 the first "real" section has index 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 e_shnum 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 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.</p> <p>elf_ndxscn() takes an existing section descriptor, <i>scn</i>, and returns its section table index. If <i>scn</i> is null or an error occurs, elf_ndxscn() returns SHN_UNDEF.</p>

EXAMPLES

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)
    {
        /* process section */
    }
```

SEE ALSO

elf(3E), elf32_getehdr(3E), elf32_getshdr(3E), elf_begin(3E), elf_flagdata(3E), elf_getdata(3E)

NAME	elf_hash – compute hash value
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lelf [<i>library</i> ...] #include <libelf.h> unsigned long elf_hash(const char *<i>name</i>);</pre>
MT-LEVEL	Unsafe
DESCRIPTION	<p>elf_hash() computes a hash value, given a null terminated string, <i>name</i>. The returned hash value, <i>h</i>, can be used as a bucket index, typically after computing $h \bmod x$ to ensure appropriate bounds.</p> <p>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 <i>name</i> 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).</p> <p>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).</p>
SEE ALSO	elf(3E) , elf_getdata(3E) , elf32_xlatetof(3E)

NAME	elf_kind – determine file type
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lelf [<i>library</i> ...] #include <libelf.h> Elf_Kind elf_kind(Elf *elf);</pre>
MT-LEVEL	Unsafe
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.</p> <p>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>
SEE ALSO	elf(3E) , elf32_getehdr(3E) , elf_begin(3E) , elf_getident(3E) , ar(4)

NAME	elf_rawfile – retrieve uninterpreted file contents
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lelf [<i>library</i> ...] #include <libelf.h> char *elf_rawfile(Elf *elf, size_t *ptr);</pre>
MT-LEVEL	Unsafe
DESCRIPTION	<p>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.</p> <p>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.</p> <p>elf_rawdata() (see elf_getdata(3E)) is a related function, providing access to sections within a file.</p> <p>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 zero stored through ptr, if ptr is non-null.</p>
SEE ALSO	elf(3E), elf32_getehdr(3E), elf_begin(3E), elf_cntl(3E), elf_getdata(3E), elf_getident(3E), elf_kind(3E)
NOTES	<p>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.</p> <p>Multiple calls to elf_rawfile() with the same ELF descriptor return the same value; the library does not create duplicate copies of the file.</p>

NAME	elf_strptr – make a string pointer
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lelf [<i>library</i> ...] #include <libelf.h> char *elf_strptr(Elf *elf, size_t section, size_t offset);</pre>
MT-LEVEL	Unsafe
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>A prototype for retrieving section names appears below. The file header specifies the section name string table in the e_shstrndx 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>
SEE ALSO	elf(3E), elf32_getshdr(3E), elf32_xlatetof(3E), elf_getdata(3E)
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 *<i>elf</i>, Elf_Cmd <i>cmd</i>);</pre>
MT-LEVEL	Unsafe
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. <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.</p> <p>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>

The following table shows ELF Header members:

Member	Notes
e_ident[EI_DATA]	Library controls other e_ident values
e_type	
e_machine	
e_version	
e_entry	
e_phoff	Only when ELF_F_LAYOUT asserted
e_shoff	Only when ELF_F_LAYOUT asserted
e_flags	
e_shstrndx	

The following table shows the Program Header members:

Member	Notes
p_type	The application controls all program header entries
p_offset	
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 ELF_F_LAYOUT asserted
sh_size	Only when ELF_F_LAYOUT asserted
sh_link	
sh_info	
sh_addralign	Only when ELF_F_LAYOUT asserted
sh_entsize	

The following table shows the Data Descriptor members:

Member	Notes
d_buf	
d_type	
d_size	
d_off	Only when ELF_F_LAYOUT 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 zero.

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.

SEE ALSO [elf\(3E\)](#), [elf32_fsize\(3E\)](#), [elf32_getehdr\(3E\)](#), [elf32_getshdr\(3E\)](#), [elf32_xlatetof\(3E\)](#), [elf_begin\(3E\)](#), [elf_flagdata\(3E\)](#), [elf_getdata\(3E\)](#)

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>
MT-LEVEL	Unsafe
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>, <i>ver</i>, 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 <i>ver</i> equal to <code>EV_NONE</code> causes elf_version() to return the library's internal version, without altering the working version. If <i>ver</i> 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>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>
SEE ALSO	elf(3E), elf32_xlatetof(3E), elf_begin(3E)
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	end, etext, 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>etext The address of etext is the first address above the program text.</p> <p>edata The address of edata is the first address above the initialized data region.</p> <p>end The address of end is the first address above the uninitialized data region.</p>
SEE ALSO	cc(1B) , brk(2) , malloc(3C) , stdio(3S)
NOTE	When execution begins, the program break (the first location beyond the data) coincides with end , but the program break may be reset by the routines brk() , malloc() , the standard input/output library (see stdio(3S)), by the profile (-p) option of cc(1B) , and so on. Thus, the current value of the program break should be determined by sbrk ((char *)0) (see brk(2)).

NAME	erf, erfc – error functions
SYNOPSIS	<pre>cc [flag ...] file ... -lm [library ...] #include <math.h> double erf(double x); double erfc(double x);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>erf(x) returns the error function of x; where $\text{erf}(x) := (2/\sqrt{\pi}) \int_0^x \exp(-t^2) dt$.</p> <p>erfc(x) returns $1.0 - \text{erf}(x)$, computed however by other methods that avoid cancellation for large x.</p>

NAME	ethers – 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>
MT-LEVEL	MT-Safe
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 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).</p> <p>The function ether_line() scans a line (pointed to by <i>l</i>) and sets the hostname and the Ethernet number (pointed to by <i>e</i>). 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. The format of the scanned line is described by ethers(4).</p>

FILES /etc/ethers
/etc/nsswitch.conf

SEE ALSO ethers(4), nsswitch.conf(4)

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>
MT-LEVEL	Safe
DESCRIPTION	<p>euclen() 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>euccol() returns the screen column width of the EUC character pointed to by <i>s</i>.</p> <p>eucscol() returns the screen column width of the EUC string pointed to by <i>str</i>.</p> <p>For the euclen() and euccol(), routines, <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>
FILES	<i>/usr/lib/locale/locale/LC_CTYPE</i>
SEE ALSO	<i>getwidth(3I)</i>

NAME	exit – terminate process
SYNOPSIS	#include <stdlib.h> void exit(int status);
MT-LEVEL	Safe
DESCRIPTION	<p>The C function exit() first calls any functions registered through the atexit(3C) function in the reverse order of their registration. exit() then calls _exit, which circumvents all such functions and cleanup.</p> <p>The symbols EXIT_SUCCESS and EXIT_FAILURE are defined in the header <stdlib.h> and may be used as the value of <i>status</i> to indicate successful or unsuccessful termination, respectively.</p>
SEE ALSO	exit(2), atexit(3C)

NAME	exp, expm1, log, log1p, log10, pow – exponential, logarithm, power
SYNOPSIS	<pre>cc [flag ...] file ... -lm [library ...] #include <math.h> double exp(double x); double expm1(double x); double log(double x); double log1p(double x); double log10(double x); double pow(double x, double y);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>exp(x) computes the exponential function e^{**x}.</p> <p>expm1(x) computes $(e^{**x})-1$ accurately even for tiny x.</p> <p>log(x) computes the natural logarithm of x.</p> <p>log1p(x) computes $\log(1+x)$ accurately even for tiny x.</p> <p>log10(x) computes the base-10 logarithm of x.</p> <p>pow(x, y) computes x raised to the power y.</p>
RETURN VALUES	For exceptional cases, matherr(3M) tabulates the values to be returned as dictated by various Standards.
SEE ALSO	matherr(3M)
DIAGNOSTICS	In IEEE 754 mode (i.e. the -xlibmieee cc compilation option), log(±0) returns $-\infty$ and raises the division by zero exception; if $x < 0$, log(x) returns a NaN and raises the invalid operation exception; if $x == +\infty$ or a quiet NaN, log(x) returns x and raises no exception; if x is a signaling NaN, log(x) returns a quiet NaN and raises the invalid operation exception; log(1) returns 0 and raises no exception; for all other positive x , log(x) returns a normalized number and raises the inexact exception.

NAME	fattach – attach a STREAMS-based file descriptor to an object in the file system name space
SYNOPSIS	int fattach(int <i>fildes</i>, const char *<i>path</i>);
MT-LEVEL	MT-Safe
DESCRIPTION	<p>The fattach() routine attaches a STREAMS-based file descriptor to an object in the file system name space, effectively associating a name with <i>fildes</i>. <i>fildes</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>fildes</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>fildes</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	If successful, fattach() returns 0; otherwise it returns -1 and sets errno to indicate an error.
ERRORS	<p>Under the following conditions, the function fattach() fails and sets errno to:</p> <p>EACCES The user is the owner of <i>path</i> but does not have write permissions on <i>path</i> or <i>fildes</i> is locked.</p> <p>EBADF <i>fildes</i> is not a valid open file descriptor.</p> <p>EBUSY <i>path</i> is currently a mount point or has a STREAMS file descriptor attached it.</p> <p>EINVAL <i>path</i> is a file in a remotely mounted directory.</p> <p>EINVAL <i>fildes</i> does not represent a STREAMS file.</p> <p>ELOOP Too many symbolic links were encountered in translating <i>path</i>.</p> <p>ENAMETOOLONG The size of <i>path</i> exceeds {PATH_MAX}, or the component of a path name is longer than {NAME_MAX} while {_POSIX_NO_TRUNC} is in effect.</p> <p>ENOENT <i>path</i> does not exist.</p>

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.

SEE ALSO

fdetach(1M), chmod(2), stat(2), fdetach(3C), isastream(3C), streamio(7)

STREAMS Programmer's Guide

NAME	fclose, fflush – close or flush a stream
SYNOPSIS	#include <stdio.h> int fclose(FILE *stream); int fflush(FILE *stream);
MT-LEVEL	MT-Safe
DESCRIPTION	<p>fclose() causes any buffered data waiting to be written for the named <i>stream</i> (see intro(3)) to be written out, and the <i>stream</i> to be closed. If the underlying file pointer is not already at end of file, and the file is one capable of seeking, the file pointer is adjusted so that the next operation on the open file pointer deals with the byte after the last one read from or written to the file being closed.</p> <p>fclose() is performed automatically for all open files upon calling exit().</p> <p>If <i>stream</i> points to an output stream or an update stream on which the most recent operation was not input, fflush() causes any buffered data waiting to be written for the named <i>stream</i> to be written to that file. Any unread data buffered in <i>stream</i> is discarded. The <i>stream</i> remains open. If <i>stream</i> is open for reading, the underlying file pointer is not already at end of file, and the file is one capable of seeking, the file pointer is adjusted so that the next operation on the open file pointer deals with the byte after the last one read from or written to the stream.</p> <p>When calling fflush(), if <i>stream</i> is a null pointer, all files open for writing are flushed.</p>
RETURN VALUES	Upon successful completion these functions return a value of zero. Otherwise EOF is returned.
SEE ALSO	close(2) , exit(2) , intro(3) , fopen(3S) , setbuf(3S) , stdio(3S)

NAME	fdatasync – synchronize a file's data
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lposix4 [<i>library</i> ...] #include <unistd.h> int fdatasync(int <i>fildev</i>);</pre>
MT-LEVEL	Async-Safe
DESCRIPTION	fdatasync() forces all currently queued I/O operations associated with the file descriptor <i>fildev</i> to synchronized I/O data integrity completion. See fcntl(5) definition of O_DSYNC .
RETURN VALUES	fdatasync() returns 0 upon success; otherwise, it returns -1 and sets errno to indicate the error condition.
ERRORS	EBADF <i>fildev</i> is not a valid file descriptor. EINVAL This implementation does not support synchronized I/O for this file. ENOSYS fdatasync() is not supported by this implementation. In the event that any of the queued I/O operations fail, fdatasync() returns the error conditions defined for read(2) and write(2) .
SEE ALSO	fcntl(2) , open(2) , read(2) , write(2) , fsync(3C) , aio_fsync(3R) , fcntl(5)
NOTES	If fdatasync() fails, outstanding I/O operations are not guaranteed to have been completed.

NAME	fdetach – detach a name from a STREAMS-based file descriptor
SYNOPSIS	int fdetach(const char *path);
MT-LEVEL	Unsafe
DESCRIPTION	The fdetach() routine detaches a STREAMS-based file descriptor from a name in the file system. <i>path</i> is the path name of the object in the file system name space, which was previously attached (see fattach(3C)). The user must be the owner of the file or a user with the appropriate privileges. All subsequent operations on <i>path</i> will operate on the file system node and not on the STREAMS file. The permissions and status of the node are restored to the state the node was in before the STREAMS file was attached to it.
RETURN VALUES	If successful, fdetach() returns 0; otherwise it returns -1 and sets errno to indicate an error.
ERRORS	Under the following conditions, the function fdetach() fails and sets errno to: EINVAL <i>path</i> is not attached to a STREAMS file. ELOOP Too many symbolic links were encountered in translating <i>path</i> . ENAMETOOLONG The size of <i>path</i> exceeds {PATH_MAX} , or a path name component is longer than {NAME_MAX} while {_POSIX_NO_TRUNC} is in effect. ENOENT <i>path</i> does not exist. ENOTDIR A component of the path prefix is not a directory. EPERM The effective user ID is not the owner of <i>path</i> or is not a user with appropriate permissions.
SEE ALSO	fdetach(1M) , fattach(3C) , streamio(7) <i>STREAMS Programmer's Guide</i>

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>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>ferror() returns non-zero when an error has previously occurred reading from or writing to the named <i>stream</i> (see intro(3)), otherwise zero.</p> <p>feof() returns non-zero when EOF has previously been detected reading the named input <i>stream</i>, otherwise zero.</p> <p>clearerr() resets the error indicator and EOF indicator to zero on the named <i>stream</i>.</p> <p>fileno() returns the integer file descriptor associated with the named <i>stream</i>; see open(2).</p>
SEE ALSO	open(2) , intro(3) , fopen(3S) , stdio(3S)

NAME	ffs – find first set bit
SYNOPSIS	#include <string.h> int ffs(const int i);
MT-LEVEL	MT-Safe
DESCRIPTION	ffs() finds the first bit set in the argument passed it and returns the index of that bit. Bits are numbered starting at 1 from the low order bit. A return value of zero indicates that the value passed is zero.

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>
MT-LEVEL	MT-Safe
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> $(\text{pd->sign}) * (\text{pd->ds}) * 10^{(\text{pd->exponent})}$ <p>is a correctly rounded approximation to <i>*px</i>. <i>pd->ds</i> has at least one and no more than DECIMAL_STRING_LENGTH-1 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 DECIMAL_STRING_LENGTH.</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 >= DECIMAL_STRING_LENGTH, then <i>ds</i> is undefined. <i>pd->exponent</i> always gets -<i>pm->ndigits</i>. Thus if <i>*px</i> == 12.34 and <i>pm->ndigits</i> == 1, then <i>pd->ds</i> gets 123, <i>pd->exponent</i> gets -1, and <i>pd->ndigits</i> gets 3.</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> trailing zeros; in other words, rounding occurs -<i>pm->ndigits</i> to the left of the decimal point, but the digits rounded away are retained as zeros. The total number of digits required is in <i>pd->ndigits</i>. <i>pd->exponent</i> always gets 0. Thus if <i>*px</i> == 12.34 and</p>

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()**.

SEE ALSO**econvert(3)**, **fconvert(3)**, **gconvert(3)**, **printf(3S)**, **sprintf(3S)**

NAME	flock – apply or remove an advisory lock on an open file
SYNOPSIS	<pre> /usr/ucb/cc [flag ...] file ... #include <sys/file.h> int flock(fd, operation) int fd, operation; </pre>
DESCRIPTION	<p>flock() applies or removes an <i>advisory</i> lock on the file associated with the file descriptor <i>fd</i>. A lock is applied by specifying an <i>operation</i> parameter LOCK_SH or LOCK_EX for a shared or exclusive lock. Additionally, this 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>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: <i>shared</i> locks and <i>exclusive</i> 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 shared lock may be <i>upgraded</i> to an exclusive lock, and vice versa, simply by specifying the appropriate lock type; the previous lock will be released and the new lock applied (possibly after other processes have gained and released the lock).</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 cmd is not a valid argument.</p> <p>EOPNOTSUPP The argument <i>fd</i> refers to an object other than a file.</p> <p>EWOULDBLOCK The file is locked and the LOCK_NB option was specified.</p>
SEE ALSO	lockd(1M), close(2), dup(2), exec(2), fcntl(2), fork(2), open(2), lockf(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>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.</p>

Processes blocked awaiting a lock may be awakened by signals.

BUGS

Locks obtained through the **flock()** mechanism are known only within the system on which they were placed. Thus, multiple clients may successfully acquire exclusive locks on the same remote file. If this behavior is not explicitly desired, the **fcntl(2)** or **lockf(3C)** functions should be used instead; these make use of the services of the **network lock manager** (see **lockd(1M)**).

NAME	flockfile, funlockfile – acquire and release stream lock
SYNOPSIS	<pre>#include <stdio.h> void flockfile(FILE *stream); void funlockfile(FILE *stream);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>flockfile() 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 as long as it is the same thread.</p> <p>funlockfile() 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>
EXAMPLES	<p>For example:</p> <pre>FILE iop; . . . flockfile(iop); fprintf(iop, "hello "); fprintf(iop, "world0); fputc(iop, 'a'); funlockfile(iop);</pre> <p>will print everything out together, blocking other threads that might want to write to the same file between fprintf's.</p> <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>
SEE ALSO	intro(3), ferror(3S), getc(3S), putc(3S), ungetc(3S), stdio(3S)
NOTES	The interfaces on this page are as proposed in the POSIX.4a Draft #6 document, and are subject to change to be compliant to the standard when it is accepted.

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	floor, ceil, rint – round to integral value in floating-point format
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lm [<i>library</i> ...] #include <math.h> double ceil(double <i>x</i>); double floor(double <i>x</i>); double rint(double <i>x</i>);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>ceil(), floor() and rint() convert a double value into an integral value in double format. They vary in how they choose the result when the argument is not already an integral value. Here an “integral value” means a value of a mathematical integer, which however might be too large to fit in a particular computer’s int format. All sufficiently large values in a particular floating-point format are already integral; in IEEE 754 double-precision format, that means all values $\geq 2^{*52}$. Zeros, infinities, and quiet NaNs are treated as integral values by these functions, which always preserve their argument’s sign.</p> <p>ceil() returns the least integral value greater than or equal to <i>x</i>. This corresponds to IEEE 754 rounding toward positive infinity.</p> <p>floor() returns the greatest integral value less than or equal to <i>x</i>. This corresponds to IEEE 754 rounding toward negative infinity.</p> <p>rint() rounds <i>x</i> to an integral value according to the current IEEE 754 rounding direction.</p>

NAME	fmtmsg – display a message on stderr or system console
SYNOPSIS	<pre>#include <fmtmsg.h> int fmtmsg(long classification, const char *label, int severity, const char *text, const char *action, const char *tag);</pre>
MT-LEVEL	Safe
DESCRIPTION	<p>Based on a message's classification component, fmtmsg() writes a formatted message to stderr, to the console, or to both.</p> <p>fmtmsg() can be used instead of the traditional printf(3S) interface to display messages to stderr. fmtmsg(), in conjunction with gettext(), provides a simple interface for producing language-independent applications.</p> <p>A formatted message consists of up to five standard components as defined below. The component, <i>classification</i>, 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><i>classification</i></p> <p>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 stderr and the system console).</p> <ul style="list-style-type: none"> • “Major classifications” identify the source of the condition. Identifiers are: MM_HARD (hardware), MM_SOFT (software), and MM_FIRM (firmware). • “Message source subclassifications” identify the type of software in which the problem is spotted. Identifiers are: MM_APPL (application), MM_UTIL (utility), and MM_OPSYS (operating system). • “Display subclassifications” indicate where the message is to be displayed. Identifiers are: MM_PRINT to display the message on the standard error stream, MM_CONSOLE 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: MM_RECOVER (recoverable) and MM_NRECOV (non-recoverable). • An additional identifier, MM_NULLMC, indicates that no classification component is supplied for the message. <p><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</p>

label **UX:cat** indicates the UNIX System V package and the **cat** application.

severity

Indicates the seriousness of the condition. Identifiers for the standard levels of *severity* are:

- **MM_HALT** indicates that the application has encountered a severe fault and is halting. Produces the print string **HALT**.
- **MM_ERROR** indicates that the application has detected a fault. Produces the print string **ERROR**.
- **MM_WARNING** indicates a condition out of the ordinary that might be a problem and should be watched. Produces the print string **WARNING**.
- **MM_INFO** provides information about a condition that is not in error. Produces the print string **INFO**.
- **MM_NOSEV** indicates that no severity level is supplied for the message.

Other severity levels may be added by using the **addseverity()** routine.

text Describes the condition that produced the message. The *text* string is not limited to a specific size.

action Describes the first step to be taken in the error recovery process. **fmtmsg()** precedes each action string with the prefix: **TOFIX:.** The *action* string is not limited to a specific size.

tag An identifier which references on-line documentation for the message. Suggested usage is that *tag* includes the *label* and a unique identifying number. A sample *tag* is **UX:cat:146**.

Environment Variables

There are two environment variables that control the behavior of **fmtmsg()**: **MSGVERB** and **SEV_LEVEL**.

MSGVERB tells **fmtmsg()** which message components it is to select when writing messages to **stderr**. The value of **MSGVERB** is a colon-separated list of optional keywords. **MSGVERB** 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 the **MSGVERB** environment variable to see which message components it is to select when generating a message to write to the standard error stream, **stderr**. The values accepted on the initial call are saved for future calls.

MSGVERB affects only which components are selected for display to the standard error stream. All message components are included in console messages.

SEV_LEVEL defines severity levels and associates print strings with them for use by **fmtmsg()**. The standard severity levels shown 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()** is controlling.

```

0   (no severity is used)
1   HALT
2   ERROR
3   WARNING
4   INFO

```

SEV_LEVEL can be set as follows:

```

SEV_LEVEL=[description[:description[:...]]]
export SEV_LEVEL

```

description is a comma-separated list containing three fields:

```

description=severity_keyword,level,printstring

```

severity_keyword is a character string that is used as the keyword on the **-s severity** option to the **fmtmsg** command. (This field is not used by the **fmtmsg()** function.)

level 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.

printstring 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 see 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).

RETURN VALUES

The exit codes for **fmtmsg()** are the following:

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
```

SEE ALSO

fmtmsg(1), addseverity(3C), gettext(3C), printf(3S)

NAME	fopen, freopen – open a stream
SYNOPSIS	<pre> /usr/ucb/cc [flag ...] file ... #include <stdio.h> FILE *fopen(file, mode) const char *file, *mode; FILE *freopen(file, mode, iop) const char *file, *mode; register FILE *iop; </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. <i>file</i> points to a character string that contains the name of the file to be opened. <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 <p>freopen() opens the file named by <i>file</i> and associates the stream pointed to by <i>iop</i> with it. The <i>mode</i> 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 <i>iop</i>.</p> <p>freopen() is typically used to attach the preopened streams associated with stdin, stdout, and stderr to other files.</p> <p>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.</p>
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, freopen, fdopen – open a stream												
SYNOPSIS	<pre>#include <stdio.h> FILE *fopen(const char *filename, const char *type); FILE *freopen(const char *filename, const char *type, FILE *stream); FILE *fdopen(int fildes, const char *type);</pre>												
MT-LEVEL	MT-Safe												
DESCRIPTION	<p>fopen() opens the file named by <i>filename</i> and associates a <i>stream</i> with it. fopen() returns a pointer to the FILE structure associated with the <i>stream</i>.</p> <p><i>filename</i> points to a character string that contains the name of the file to be opened.</p> <p><i>type</i> is a character string beginning with one of the following sequences:</p> <table border="0" style="margin-left: 2em;"> <tr> <td>“r” or “rb”</td> <td>open for reading</td> </tr> <tr> <td>“w” or “wb”</td> <td>truncate to zero length or create for writing</td> </tr> <tr> <td>“a” or “ab”</td> <td>append; open for writing at end of file, or create for writing</td> </tr> <tr> <td>“r+”, “r+b” or “rb+”</td> <td>open for update (reading and writing)</td> </tr> <tr> <td>“w+”, “w+b” or “wb+”</td> <td>truncate or create for update</td> </tr> <tr> <td>“a+”, “a+b” or “ab+”</td> <td>append; open or create for update at end-of-file</td> </tr> </table> <p>The “b” is ignored in the above <i>types</i>. The “b” exists to distinguish binary files from text files. However, there is no distinction between these types of files on a UNIX system.</p> <p>freopen() substitutes the named file in place of the open <i>stream</i>. A flush is first attempted, and then the original <i>stream</i> is closed, regardless of whether the open ultimately succeeds. Failure to flush or close <i>stream</i> successfully is ignored. freopen() returns a pointer to the FILE structure associated with <i>stream</i>.</p> <p>freopen() is typically used to attach the preopened <i>streams</i> associated with stdin, stdout, and stderr to other files. stderr is by default unbuffered, but the use of freopen() will cause it to become buffered or line-buffered.</p> <p>fdopen() associates a <i>stream</i> with a file descriptor. File descriptors are obtained from open(2), dup(2), creat(2), or pipe(2), which open files but do not return pointers to a FILE structure <i>stream</i>. Streams are necessary input for almost all of the Section 3S library routines. The <i>type</i> of <i>stream</i> must agree with the mode of the open file. The file position indicator associated with <i>stream</i> is set to the position indicated by the file offset associated with <i>fildes</i>.</p> <p>When a file is opened for update, both input and output may be done on the resulting <i>stream</i>. However, output may not be directly followed by input without an intervening fflush(), fseek(), fsetpos(), or rewind(), and input may not be directly followed by output without an intervening fseek(), fsetpos(), or rewind(), or an input operation that encounters end-of-file.</p>	“r” or “rb”	open for reading	“w” or “wb”	truncate to zero length or create for writing	“a” or “ab”	append; open for writing at end of file, or create for writing	“r+”, “r+b” or “rb+”	open for update (reading and writing)	“w+”, “w+b” or “wb+”	truncate or create for update	“a+”, “a+b” or “ab+”	append; open or create for update at end-of-file
“r” or “rb”	open for reading												
“w” or “wb”	truncate to zero length or create for writing												
“a” or “ab”	append; open for writing at end of file, or create for writing												
“r+”, “r+b” or “rb+”	open for update (reading and writing)												
“w+”, “w+b” or “wb+”	truncate or create for update												
“a+”, “a+b” or “ab+”	append; open or create for update at end-of-file												

When a file is opened for append (that is, when *type* is “a”, “ab”, “a+”, or “ab+”), it is impossible to overwrite information already in the file. **fseek()** may be used to reposition the file pointer to any position in the file, but when output is written to the file, the current file pointer is disregarded. All output is written at the end of the file and causes the file pointer to be repositioned at the end of the output. 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 inter-mixed in the file in the order in which it is written.

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 are cleared for the *stream*.

RETURN VALUES

The functions **fopen()** and **freopen()** return a null pointer if *path* cannot be accessed, or if *type* is invalid, or if the file cannot be opened.

The function **fdopen()** returns a null pointer if *fdes* is not an open file descriptor, or if *type* is invalid, or if the file cannot be opened.

The functions **fopen()** or **fdopen()** may fail and not set **errno** if there are no free **stdio** streams.

File descriptors used by **fdopen()** must be less than 255.

SEE ALSO

close(2), **creat(2)**, **dup(2)**, **open(2)**, **pipe(2)**, **write(2)**, **fclose(3S)**, **fseek(3S)**, **setbuf(3S)**, **stdio(3S)**

NAME	form_cursor, pos_form_cursor – position forms window cursor
SYNOPSIS	#include <form.h> int pos_form_cursor(FORM *form);
MT-LEVEL	Unsafe
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 curses library I/O routine.
RETURN VALUES	pos_form_cursor() 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.
SEE ALSO	curses(3X), forms(3X)
NOTES	The header <form.h> automatically includes the headers <eti.h> and <curses.h> .

NAME	form_data, data_ahead, data_behind – tell if forms field has off-screen data ahead or behind
SYNOPSIS	<pre>#include <form.h> int data_ahead(FORM *form); int data_behind(FORM *form);</pre>
MT-LEVEL	Unsafe
DESCRIPTION	<p>data_ahead() returns TRUE (1) if the current field has more off-screen data ahead; otherwise it returns FALSE (0).</p> <p>data_behind() returns TRUE (1) if the current field has more off-screen data behind; otherwise it returns FALSE (0).</p>
SEE ALSO	curses(3X) , forms(3X)
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>#include <form.h> int form_driver(FORM *form, int c);</pre>																																																								
MT-LEVEL	Unsafe																																																								
DESCRIPTION	<p>form_driver() is the workhorse of the forms subsystem; it checks to determine whether the character <i>c</i> is a forms 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 E_UNKNOWN_COMMAND. Application defined commands should be defined relative to MAX_COMMAND, 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> <tr><td>REQ_PREV_CHAR</td><td>Move to the previous character in the field.</td></tr> <tr><td>REQ_NEXT_LINE</td><td>Move to the next line in the field.</td></tr> <tr><td>REQ_PREV_LINE</td><td>Move to the previous line in the field.</td></tr> <tr><td>REQ_NEXT_WORD</td><td>Move to the next word in the field.</td></tr> <tr><td>REQ_PREV_WORD</td><td>Move to the previous word in the field.</td></tr> <tr><td>REQ_BEG_FIELD</td><td>Move to the first char in the field.</td></tr> <tr><td>REQ_END_FIELD</td><td>Move after the last char in the field.</td></tr> <tr><td>REQ_BEG_LINE</td><td>Move to the beginning of the line.</td></tr> <tr><td>REQ_END_LINE</td><td>Move after the last char in the line.</td></tr> <tr><td>REQ_LEFT_CHAR</td><td>Move left in the field.</td></tr> <tr><td>REQ_RIGHT_CHAR</td><td>Move right 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_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_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_HFHALF	Horizontal scroll forward half a line.
REQ_SCR_HBHALF	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 the form driver.

SEE ALSO `curses(3X)`, `forms(3X)`

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>#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>															
MT-LEVEL	Unsafe															
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" style="margin-left: 20px;"> <tr> <td>E_OK</td> <td>–</td> <td>The function returned successfully.</td> </tr> <tr> <td>E_CONNECTED</td> <td>–</td> <td>The field is already connected to a form.</td> </tr> <tr> <td>E_SYSTEM_ERROR</td> <td>–</td> <td>System error.</td> </tr> <tr> <td>E_BAD_ARGUMENT</td> <td>–</td> <td>An argument is incorrect.</td> </tr> <tr> <td>E_POSTED</td> <td>–</td> <td>The form is posted.</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_POSTED	–	The form is posted.
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.														
SEE ALSO	curses(3X) , forms(3X)															
NOTES	The header <form.h> automatically includes the headers <eti.h> and <curses.h> .															

NAME	form_field_attributes, set_field_fore, field_fore, set_field_back, field_back, set_field_pad, field_pad – format the general display attributes of forms									
SYNOPSIS	<pre>#include <form.h> int set_field_fore(FIELD *field, chtype attr); chtype field_fore(FIELD *field); int set_field_back(FIELD *field, chtype attr); chtype field_back(FIELD *field); int set_field_pad(FIELD *field, int pad); int field_pad(FIELD *field);</pre>									
MT-LEVEL	Unsafe									
DESCRIPTION	<p>set_field_fore() sets the foreground attribute of <i>field</i>. The foreground attribute is the low-level curses display attribute used to display the field contents. field_fore() returns the foreground attribute of <i>field</i>.</p> <p>set_field_back() sets the background attribute of <i>field</i>. The background attribute is the low-level curses display attribute used to display the extent of the field. field_back() returns the background attribute of <i>field</i>.</p> <p>set_field_pad() sets the pad character of <i>field</i> to <i>pad</i>. The pad character is the character used to fill within the field. field_pad() returns the pad character of <i>field</i>.</p>									
RETURN VALUES	<p>field_fore(), field_back(), and field_pad() return default values if <i>field</i> is NULL. If <i>field</i> is not NULL and is not a valid FIELD pointer, the return value from these routines is undefined.</p> <p>set_field_fore(), set_field_back(), and set_field_pad() return one of the following:</p> <table border="0" style="margin-left: 20px;"> <tr> <td>E_OK</td> <td>–</td> <td>The function returned successfully.</td> </tr> <tr> <td>E_SYSTEM_ERROR</td> <td>–</td> <td>System error.</td> </tr> <tr> <td>E_BAD_ARGUMENT</td> <td>–</td> <td>An argument is incorrect.</td> </tr> </table>	E_OK	–	The function returned successfully.	E_SYSTEM_ERROR	–	System error.	E_BAD_ARGUMENT	–	An argument is incorrect.
E_OK	–	The function returned successfully.								
E_SYSTEM_ERROR	–	System error.								
E_BAD_ARGUMENT	–	An argument is incorrect.								
SEE ALSO	curses(3X) , forms(3X)									
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>#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>									
MT-LEVEL	Unsafe									
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 forms 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=0</i> turns off any maximum growth.</p>									
RETURN VALUES	<p>field_buffer() returns NULL on error.</p> <p>field_status() returns TRUE or FALSE.</p> <p>set_field_buffer(), set_field_status(), and set_max_field() return one of the following:</p> <table border="0" style="margin-left: 20px;"> <tr> <td>E_OK</td> <td>–</td> <td>The function returned successfully.</td> </tr> <tr> <td>E_SYSTEM_ERROR</td> <td>–</td> <td>System error.</td> </tr> <tr> <td>E_BAD_ARGUMENT</td> <td>–</td> <td>An argument is incorrect.</td> </tr> </table>	E_OK	–	The function returned successfully.	E_SYSTEM_ERROR	–	System error.	E_BAD_ARGUMENT	–	An argument is incorrect.
E_OK	–	The function returned successfully.								
E_SYSTEM_ERROR	–	System error.								
E_BAD_ARGUMENT	–	An argument is incorrect.								
SEE ALSO	curses(3X) , forms(3X)									
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>#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>									
MT-LEVEL	Unsafe									
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 O_STATIC.</p>									
RETURN VALUES	<p>These routines return one of the following:</p> <table border="0"> <tr> <td>E_OK</td> <td>–</td> <td>The function returned successfully.</td> </tr> <tr> <td>E_SYSTEM_ERROR</td> <td>–</td> <td>System error.</td> </tr> <tr> <td>E_BAD_ARGUMENT</td> <td>–</td> <td>An argument is incorrect.</td> </tr> </table>	E_OK	–	The function returned successfully.	E_SYSTEM_ERROR	–	System error.	E_BAD_ARGUMENT	–	An argument is incorrect.
E_OK	–	The function returned successfully.								
E_SYSTEM_ERROR	–	System error.								
E_BAD_ARGUMENT	–	An argument is incorrect.								
SEE ALSO	curses(3X) , forms(3X)									
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>#include <form.h> int set_field_just(FIELD *field, int justification); int field_just(FIELD *field);</pre>
MT-LEVEL	Unsafe
DESCRIPTION	<p>set_field_just() sets the justification for <i>field</i>. Justification may be one of: NO_JUSTIFICATION, JUSTIFY_RIGHT, JUSTIFY_LEFT, or JUSTIFY_CENTER.</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> <p style="text-align: center;">NO_JUSTIFICATION, JUSTIFY_RIGHT, JUSTIFY_LEFT, or JUSTIFY_CENTER.</p> <p>set_field_just() returns one of the following:</p> <p>E_OK –The function returned successfully. E_SYSTEM_ERROR –System error. E_BAD_ARGUMENT –An argument is incorrect.</p>
SEE ALSO	curses(3X) , forms(3X)
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>#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>												
MT-LEVEL	Unsafe												
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 NULL on error. free_field() returns one of the following:</p> <table border="0" style="margin-left: 20px;"> <tr> <td>E_OK</td> <td>–</td> <td>The function returned successfully.</td> </tr> <tr> <td>E_CONNECTED</td> <td>–</td> <td>The field is already connected to a form.</td> </tr> <tr> <td>E_SYSTEM_ERROR</td> <td>–</td> <td>System error.</td> </tr> <tr> <td>E_BAD_ARGUMENT</td> <td>–</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.											
SEE ALSO	curses(3X) , forms(3X)												
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>#include <form.h> int set_field_opts(FIELD *field, OPTIONS opts); int set_field_opts(FIELD *field, OPTIONS opts); int field_opts_on(FIELD *field, OPTIONS opts); int field_opts_off(FIELD *field, OPTIONS opts); OPTIONS field_opts(FIELD *field);</pre>																				
MT-LEVEL	Unsafe																				
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>Field Options:</p> <table border="0"> <tr> <td style="padding-right: 20px;">O_VISIBLE</td> <td>The field is displayed.</td> </tr> <tr> <td>O_ACTIVE</td> <td>The field is visited during processing.</td> </tr> <tr> <td>O_PUBLIC</td> <td>The field contents are displayed as data is entered.</td> </tr> <tr> <td>O_EDIT</td> <td>The field can be edited.</td> </tr> <tr> <td>O_WRAP</td> <td>Words not fitting on a line are wrapped to the next line.</td> </tr> <tr> <td>O_BLANK</td> <td>The whole field is cleared if a character is entered in the first position.</td> </tr> <tr> <td>O_AUTOSKIP</td> <td>Skip to the next field when the current field becomes full.</td> </tr> <tr> <td>O_NULLOK</td> <td>A blank field is considered valid.</td> </tr> <tr> <td>O_STATIC</td> <td>The field buffers are fixed in size.</td> </tr> <tr> <td>O_PASSOK</td> <td>Validate field only if modified by user.</td> </tr> </table>	O_VISIBLE	The field is displayed.	O_ACTIVE	The field is visited during processing.	O_PUBLIC	The field contents are displayed as data is entered.	O_EDIT	The field can be edited.	O_WRAP	Words not fitting on a line are wrapped to the next line.	O_BLANK	The whole field is cleared if a character is entered in the first position.	O_AUTOSKIP	Skip to the next field when the current field becomes full.	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.
O_VISIBLE	The field is displayed.																				
O_ACTIVE	The field is visited during processing.																				
O_PUBLIC	The field contents are displayed as data is entered.																				
O_EDIT	The field can be edited.																				
O_WRAP	Words not fitting on a line are wrapped to the next line.																				
O_BLANK	The whole field is cleared if a character is entered in the first position.																				
O_AUTOSKIP	Skip to the next field when the current field becomes full.																				
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	<p>set_field_opts, field_opts_on and field_opts_off return one of the following:</p> <table border="0"> <tr> <td style="padding-right: 20px;">E_OK</td> <td style="padding-right: 20px;">–</td> <td>The function returned successfully.</td> </tr> <tr> <td>E_SYSTEM_ERROR</td> <td>–</td> <td>System error.</td> </tr> <tr> <td>E_CURRENT</td> <td>–</td> <td>The field is the current field.</td> </tr> </table>	E_OK	–	The function returned successfully.	E_SYSTEM_ERROR	–	System error.	E_CURRENT	–	The field is the current field.											
E_OK	–	The function returned successfully.																			
E_SYSTEM_ERROR	–	System error.																			
E_CURRENT	–	The field is the current field.																			
SEE ALSO	curses(3X) , forms(3X)																				

NOTES

The header `<form.h>` automatically includes the headers `<eti.h>` and `<curses.h>`.

NAME	form_field_userptr, set_field_userptr, field_userptr – associate application data with forms
SYNOPSIS	<pre>#include <form.h> int set_field_userptr(FIELD *field, char *ptr); char *field_userptr(FIELD *field);</pre>
MT-LEVEL	Unsafe
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	field_userptr() returns NULL on error. set_field_userptr() returns one of the following: E_OK – The function returned successfully. E_SYSTEM_ERROR – System error.
SEE ALSO	curses(3X) , forms(3X)
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>#include <form.h> int set_field_type(FIELD *field, FIELDTYPE *type, ...); FIELDTYPE *field_type(FIELD *field); char *field_arg(FIELD *field);</pre>
MT-LEVEL	Unsafe
DESCRIPTION	<p>set_field_type() associates the specified field type with <i>field</i>. Certain field types take additional arguments. TYPE_ALNUM, for instance, requires one, the minimum width specification for the field. The other predefined field types are: TYPE_ALPHA, TYPE_ENUM, TYPE_INTEGER, TYPE_NUMERIC, and TYPE_REGEX.</p> <p>field_type() returns a pointer to the field type of <i>field</i>. NULL 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>. NULL is returned if no field type is assigned.</p>
RETURN VALUES	<p>field_type() and field_arg() return NULL on error.</p> <p>set_field_type() returns one of the following:</p> <ul style="list-style-type: none"> E_OK – The function returned successfully. E_SYSTEM_ERROR – System error.
SEE ALSO	curses(3X) , forms(3X)
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>#include <form.h> FIELDTYPE *new_fieldtype(int (* field_check) (FIELD *, char *), int (* char_check)(int, char *)); int free_fieldtype(FIELDTYPE *fieldtype); int set_fieldtype_arg(FIELDTYPE *fieldtype, char *(* mak_arg)(va_list *), char *(* copy_arg) (char *), void (* free_arg)(char *));" int set_fieldtype_choice(FIELDTYPE *fieldtype, int (* next_choice)(FIELD *, char *), int (* prev_choice)(FIELD *, char *)); FIELDTYPE *link_fieldtype(FIELDTYPE *type1, FIELDTYPE *type2);</pre>
MT-LEVEL	Unsafe
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> <p>link_fieldtype() returns a pointer to the field type built from the two given types. The constituent types may be any application-defined or pre-defined types.</p>
RETURN VALUES	<p>Routines that return pointers always return NULL on error. Routines that return an integer return one of the following:</p> <ul style="list-style-type: none"> E_OK – The function returned successfully. E_SYSTEM_ERROR – System error. E_BAD_ARGUMENT – An argument is incorrect. E_CONNECTED – Type is connected to one or more fields.
SEE ALSO	curses(3X), forms(3X)

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>#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>
MT-LEVEL	Unsafe
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> <p>set_field_init() assigns an application-defined function to be called when the <i>form</i> is posted and just after the current field changes. field_init() returns a pointer to the function, if any.</p> <p>set_field_term() assigns an application-defined function to be called when the <i>form</i> is unposted and just before the current field changes. field_term() returns a pointer to the function, if any.</p>
RETURN VALUES	<p>Routines that return pointers always return NULL on error. Routines that return an integer return one of the following:</p> <ul style="list-style-type: none"> E_OK – The function returned successfully. E_SYSTEM_ERROR – System error.
SEE ALSO	curses(3X), forms(3X)

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>#include <form.h> FORM *new_form(FIELD **fields); int free_form(FORM *form);</pre>
MT-LEVEL	Unsafe
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> <ul style="list-style-type: none">E_OK – The function returned successfully.E_BAD_ARGUMENT – An argument is incorrect.E_POSTED – The form is posted.
SEE ALSO	curses(3X), forms(3X)
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>#include <form.h> int set_new_page(FIELD *field, int bool); int new_page(FIELD *field);</pre>
MT-LEVEL	Unsafe
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> <ul style="list-style-type: none">E_OK – The function returned successfully.E_CONNECTED – The field is already connected to a form.E_SYSTEM_ERROR – System error.
SEE ALSO	curses(3X) , forms(3X)
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>#include <form.h> int set_form_opts(FORM *form, OPTIONS opts); int form_opts_on(FORM *form, OPTIONS opts); int form_opts_off(FORM *form, OPTIONS opts); OPTIONS form_opts(FORM *form);</pre>
MT-LEVEL	Unsafe
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> <p>O_NL_OVERLOAD Overload the REQ_NEW_LINE form driver request. O_BS_OVERLOAD Overload the REQ_DEL_PREV form driver request.</p>
RETURN VALUES	<p>set_form_opts(), form_opts_on(), and form_opts_off() return one of the following:</p> <p>E_OK – The function returned successfully. E_SYSTEM_ERROR – System error.</p>
SEE ALSO	curses(3X) , forms(3X)
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>#include <form.h> int set_form_page(FORM *form, int page); int form_page(FORM *form); int set_current_field(FORM *form, FIELD *field); FIELD *current_field(FORM *form); int field_index(FIELD *field);</pre>																		
MT-LEVEL	Unsafe																		
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> <table border="0" style="margin-left: 20px;"> <tr> <td>E_OK</td> <td>–</td> <td>The function returned successfully.</td> </tr> <tr> <td>E_SYSTEM_ERROR</td> <td>–</td> <td>System error.</td> </tr> <tr> <td>E_BAD_ARGUMENT</td> <td>–</td> <td>An argument is incorrect.</td> </tr> <tr> <td>E_BAD_STATE</td> <td>–</td> <td>The routine was called from an initialization or termination function.</td> </tr> <tr> <td>E_INVALID_FIELD</td> <td>–</td> <td>The field contents are invalid.</td> </tr> <tr> <td>E_REQUEST_DENIED</td> <td>–</td> <td>The form driver request failed.</td> </tr> </table>	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 initialization or termination function.	E_INVALID_FIELD	–	The field contents are invalid.	E_REQUEST_DENIED	–	The form driver request failed.
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 initialization or termination function.																	
E_INVALID_FIELD	–	The field contents are invalid.																	
E_REQUEST_DENIED	–	The form driver request failed.																	
SEE ALSO	curses(3X) , forms(3X)																		
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>#include <form.h> int post_form(FORM *form); int unpost_form(FORM *form);</pre>																								
MT-LEVEL	Unsafe																								
DESCRIPTION	<p>post_form() writes <i>form</i> into its associated subwindow. The application programmer must use curses library routines to display the form on the physical screen or call update_panels() if the panels 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> <table><tr><td>E_OK</td><td>–</td><td>The function returned successfully.</td></tr><tr><td>E_SYSTEM_ERROR</td><td>–</td><td>System error.</td></tr><tr><td>E_BAD_ARGUMENT</td><td>–</td><td>An argument is incorrect.</td></tr><tr><td>E_POSTED</td><td>–</td><td>The form is posted.</td></tr><tr><td>E_NOT_POSTED</td><td>–</td><td>The form is not posted.</td></tr><tr><td>E_NO_ROOM</td><td>–</td><td>The form does not fit in the subwindow.</td></tr><tr><td>E_BAD_STATE</td><td>–</td><td>The routine was called from an initialization or termination function.</td></tr><tr><td>E_NOT_CONNECTED</td><td>–</td><td>The field is not connected to a form.</td></tr></table>	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 initialization or termination function.	E_NOT_CONNECTED	–	The field is not connected to a form.
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 initialization or termination function.																							
E_NOT_CONNECTED	–	The field is not connected to a form.																							
SEE ALSO	curses(3X) , forms(3X) , panel_update(3X) , panels(3X)																								
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>#include <form.h> int set_form_userptr(FORM *form, char *ptr); char *form_userptr(FORM *form);</pre>
MT-LEVEL	Unsafe
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	form_userptr() returns NULL on error. set_form_userptr() returns one of the following: E_OK – The function returned successfully. E_SYSTEM_ERROR – System error.
SEE ALSO	curses(3X) , forms(3X)
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>#include <form.h> int set_form_win(FORM *form, WINDOW *win); WINDOW *form_win(FORM *form); int set_form_sub(FORM *form, WINDOW *sub); WINDOW *form_sub(FORM *form); int scale_form(FORM *form, int *rows, int *cols);</pre>															
MT-LEVEL	Unsafe															
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> <table border="0" style="margin-left: 20px;"> <tr> <td>E_OK</td> <td>–</td> <td>The function returned successfully.</td> </tr> <tr> <td>E_SYSTEM_ERROR</td> <td>–</td> <td>System error.</td> </tr> <tr> <td>E_BAD_ARGUMENT</td> <td>–</td> <td>An argument is incorrect.</td> </tr> <tr> <td>E_NOT_CONNECTED</td> <td>–</td> <td>The field is not connected to a form.</td> </tr> <tr> <td>E_POSTED</td> <td>–</td> <td>The form is posted.</td> </tr> </table>	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.
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.														
SEE ALSO	curses(3X) , forms(3X)															
NOTES	The header <form.h> automatically includes the headers <eti.h> and <curses.h> .															

NAME	forms – character based forms package																																		
SYNOPSIS	#include <form.h>																																		
MT-LEVEL	Unsafe																																		
DESCRIPTION	<p>The form library is built using the curses library, and any program using forms routines must call one of the curses initialization routines such as initscr. A program using these routines must be compiled with -lform and -lcurses on the cc command line.</p> <p>The forms package gives the applications programmer a terminal-independent method of creating and customizing forms for user-interaction. The forms package includes: field routines, which are used to create and customize fields, link fields and assign field types; fieldtype 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 forms 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 NULL field pointer. If an application changes a current default field attribute value, subsequent fields created using new_field() 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 forms 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> <tr><td>field_index</td><td>form_page(3X)</td></tr> <tr><td>field_info</td><td>form_field_info(3X)</td></tr> <tr><td>field_init</td><td>form_hook(3X)</td></tr> <tr><td>field_just</td><td>form_field_just(3X)</td></tr> <tr><td>field_opts</td><td>form_field_opts(3X)</td></tr> <tr><td>field_opts_off</td><td>form_field_opts(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)	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)
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:

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_CURRENT	-	The field is the current field.
E_POSTED	-	The form is posted.
E_NOT_POSTED	-	The form is not posted.
E_INVALID_FIELD	-	The field contents are invalid.
E_NOT_CONNECTED	-	The field is not connected to a form.
E_NO_ROOM	-	The form does not fit in the subwindow.
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.

SEE ALSO

curses(3X), 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	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>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>There are five floating-point exceptions: divide-by-zero, overflow, underflow, imprecise (inexact) result, and invalid operation. 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 following floating-point exception masks are OR-ed together to form <i>mask</i>.</p> <pre>FP_X_INV /* invalid operation exception */ FP_X_OFL /* overflow exception */ FP_X_UFL /* 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 fpsetround () and returned by fpgetround().</p> <pre>FP_RN /* round to nearest representative number */ FP_RP /* round to plus infinity */ FP_RM /* round to minus infinity */ FP_RZ /* round to zero (truncate) */</pre> <p>The default environment is rounding mode set to nearest (FP_RN) and all traps disabled. Individual bits may be examined using the constants defined in <ieeefp.h>.</p>
RETURN VALUES	<p>fpgetround() returns the current rounding mode.</p> <p>fpsetround() sets the rounding mode and returns the previous rounding mode.</p> <p>fpgetmask() returns the current exception masks.</p> <p>fpsetmask() sets the exception masks and returns the previous setting.</p> <p>fpgetsticky() returns the current exception sticky flags.</p>

fpsetsticky() sets (clears) the exception sticky flags and returns the previous setting.

SEE ALSO

isnan(3C)

NOTES

fpsetsticky() modifies all sticky flags. **fpsetmask()** changes all mask bits. **fpsetmask()** clears the sticky bit corresponding to any exception being enabled.

C requires truncation (round to zero) for floating point to integral conversions. The current rounding mode has no effect on these conversions.

One must clear the sticky bit to recover from the trap and to proceed. If the sticky bit is not cleared before the next trap occurs, a wrong exception type may be signaled.

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>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>fread() 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>. fread() 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. fread() increments the data pointer in <i>stream</i> to point to the byte following the last byte read if there is one. fread() does not change the contents of <i>stream</i>. fread() returns the number of items read.</p> <p>fwrite() 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>. fwrite() stops writing when it has written <i>nitems</i> items of data or if an error condition is encountered on <i>stream</i>. fwrite() does not change the contents of the array pointed to by <i>ptr</i>. fwrite() increments the data-pointer in <i>stream</i> by the number of bytes written. fwrite() 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>The ferror() or feof() routines must be used to distinguish between an error condition and end-of-file condition.</p>
RETURN VALUES	If an error occurs, fread() and fwrite() return 0 and set the error indicator for <i>stream</i> .
SEE ALSO	read(2) , write(2) , fclose(3S) , fopen(3S) , getc(3S) , gets(3S) , printf(3S) , putc(3S) , puts(3S) , scanf(3S) , stdio(3S)

NAME	frexp, ldexp, logb, modf, modff, nextafter, scalb – manipulate parts of floating-point numbers
SYNOPSIS	<pre>cc [flag ...] file ... -lm [library ...] #include <math.h> double frexp(double value, int *epr); double ldexp(double value, int exp); double logb(double value); double modf(double value, double *iptr); float modff(float value, float *iptr); double nextafter(double value1, double value2); double scalb(double value, double exp);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>Every non-zero number can be written uniquely as $x * 2^n$, where the “mantissa” (fraction) x is in the range $0.5 \leq x < 1.0$, and the “exponent” n is an integer. frexp() returns the mantissa of a double <i>value</i>, and stores the exponent indirectly in the location pointed to by <i>epr</i>. If <i>value</i> is zero, both results returned by frexp() are zero.</p> <p>ldexp() and scalb() return the quantity $value * 2^{exp}$. The only difference between the two is that scalb() of a signaling NaN will result in the invalid operation exception being raised.</p> <p>logb() returns the unbiased exponent of its floating-point argument as a double-precision floating-point value.</p> <p>modf() and modff() (single-precision version) return the signed fractional part of <i>value</i> and store the integral part indirectly in the location pointed to by <i>iptr</i>.</p> <p>nextafter() returns the next representable double-precision floating-point value following <i>value1</i> in the direction of <i>value2</i>. Thus, if <i>value2</i> is less than <i>value1</i>, nextafter() returns the largest representable floating-point number less than <i>value1</i>.</p>
RETURN VALUES	<p>If ldexp() would cause overflow, \pmHUGE (defined in <code><math.h></code>) is returned (according to the sign of <i>value</i>), and errno is set to ERANGE. If ldexp() would cause underflow, zero is returned and errno is set to ERANGE. If the input <i>value</i> to ldexp() is NaN or infinity, that input is returned and errno is set to EDOM. The same error conditions apply to scalb() except that a signaling NaN as input will result in the raising of the invalid operation exception.</p>

logb() of NaN returns that NaN, **logb()** of infinity returns positive infinity, and **logb()** of zero returns negative infinity and results in the raising of the divide by zero exception. In each of these conditions **errno** is set to **EDOM**.

If input *value1* to **nextafter()** is positive or negative infinity, that input is returned and **errno** is set to **EDOM**. The overflow and inexact exceptions are signalled when input *value1* is finite, but **nextafter(*value1*, *value2*)** is not. The underflow and inexact exceptions are signalled when **nextafter(*value1*, *value2*)** lies strictly between $+2^{-1022}$ and -2^{-1022} . In both cases **errno** is set to **ERANGE**.

NAME	fseek, rewind, ftell – reposition a file pointer in a stream
SYNOPSIS	<pre>#include <stdio.h> int fseek(FILE *stream, long offset, int ptrname); void rewind(FILE *stream); long ftell(FILE *stream);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>fseek() sets the position of the next input or output operation on the <i>stream</i> (see intro(3)). The new position is at the signed distance <i>offset</i> bytes from the beginning, from the current position, or from the end of the file, according to a <i>ptrname</i> value of SEEK_SET, SEEK_CUR, or SEEK_END (defined in <stdio.h>) as follows:</p> <p>SEEK_SET set position equal to <i>offset</i> bytes. SEEK_CUR set position to current location plus <i>offset</i>. SEEK_END set position to EOF plus <i>offset</i>.</p> <p>fseek() allows the file position indicator to be set beyond the end of the existing data in the file. If data is later written at this point, subsequent reads of data in the gap will return zero until data is actually written into the gap. fseek(), by itself, does not extend the size of the file.</p> <p>rewind(stream) is equivalent to:</p> <pre>(void) fseek (stream, 0L, SEEK_SET);</pre> <p>except that rewind() also clears the error indicator on <i>stream</i>.</p> <p>fseek() and rewind() clear the EOF indicator and undo any effects of ungetc() on <i>stream</i>. After fseek() or rewind(), the next operation on a file opened for update may be either input or output.</p> <p>If <i>stream</i> is writable and buffered data has not been written to the underlying file, fseek() and rewind() cause the unwritten data to be written to the file.</p> <p>ftell() returns the offset of the current byte relative to the beginning of the file associated with the named <i>stream</i>.</p>
RETURN VALUES	fseek() returns -1 for improper seeks, otherwise zero. An improper seek can be, for example, an fseek() done on a file that has not been opened via fopen() ; in particular, fseek() may not be used on a terminal or on a file opened via popen() . After a stream is closed, no further operations are defined on that stream.
SEE ALSO	lseek(2) , write(2) , intro(3) , fopen(3S) , popen(3S) , stdio(3S) , ungetc(3S)

NOTES

Although on the UNIX system an offset returned by **ftell()** 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.

NAME	fsetpos , fgetpos – reposition a file pointer in a stream
SYNOPSIS	#include <stdio.h> int fsetpos(FILE *stream, const fpos_t *pos); int fgetpos(FILE *stream, fpos_t *pos);
MT-LEVEL	MT-Safe
DESCRIPTION	fsetpos() sets the position of the next input or output operation on the <i>stream</i> according to the value of the object pointed to by <i>pos</i> . The object pointed to by <i>pos</i> must be a value returned by an earlier call to fgetpos() on the same stream. fsetpos() clears the end-of-file indicator for the stream and undoes any effects of the ungetc() function on the same stream. After fsetpos() , the next operation on a file opened for update may be either input or output. fgetpos() stores the current value of the file position indicator for <i>stream</i> in the object pointed to by <i>pos</i> . The value stored contains information usable by fsetpos() for repositioning the stream to its position at the time of the call to fgetpos() .
RETURN VALUES	If successful, both fsetpos() and fgetpos() return zero. Otherwise, they both return nonzero.
SEE ALSO	lseek(2) , fseek(3S) , ungetc(3S)

NAME	fsync – synchronize a file's in-memory state with that on the physical medium								
SYNOPSIS	<pre>#include <unistd.h> int fsync(int <i>fildev</i>);</pre>								
DESCRIPTION	<p>fsync() moves all modified data and attributes of the file descriptor <i>fildev</i> to a storage device. When fsync() returns, all in-memory modified copies of buffers associated with <i>fildev</i> have been written to the physical medium. fsync() is different from sync(), which schedules disk I/O for all files but returns before the I/O completes. fsync() forces all outstanding data operations to synchronized file integrity completion (see fcntl(5) definition of O_SYNC.)</p> <p>fsync() should be used by programs that require that a file be in a known state. For example, a program 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>								
RETURN VALUES	Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and errno is set to indicate the error.								
ERRORS	<p>fsync() fails if one or more of the following are true:</p> <table><tr><td>EBADF</td><td><i>fildev</i> is not a valid file descriptor open for writing.</td></tr><tr><td>EINTR</td><td>A signal was caught during execution of the fsync() function.</td></tr><tr><td>EIO</td><td>An I/O error occurred while reading from or writing to the file system.</td></tr><tr><td>ENOLINK</td><td><i>fildev</i> is on a remote machine and the link to that machine is no longer active.</td></tr></table>	EBADF	<i>fildev</i> is not a valid file descriptor open for writing.	EINTR	A signal was caught during execution of the fsync() function.	EIO	An I/O error occurred while reading from or writing to the file system.	ENOLINK	<i>fildev</i> is on a remote machine and the link to that machine is no longer active.
EBADF	<i>fildev</i> is not a valid file descriptor open for writing.								
EINTR	A signal was caught during execution of the fsync() function.								
EIO	An I/O error occurred while reading from or writing to the file system.								
ENOLINK	<i>fildev</i> is on a remote machine and the link to that machine is no longer active.								
SEE ALSO	sync(2) , fdatasync(3R) , fcntl(5)								
NOTES	The way the data reach the physical medium depends on both implementation and hardware. fsync() returns when the device driver tells it that the write has taken place.								

NAME	ftime – get date and time
SYNOPSIS	<pre> /usr/ucb/cc [flag ...] file ... #include <sys/types.h> #include <sys/timeb.h> int ftime(tp) register struct timeb *tp; </pre>
DESCRIPTION	<p>The ftime() entry fills in a structure pointed to by its argument. The structure is defined in <sys/timeb.h> and contains the following members:</p> <pre> time_t time; unsigned short millitm; short timezone; short dstflag; </pre> <p>The structure contains the time since the epoch in seconds, up to 1000 milliseconds of more-precise interval, the local time zone (this member is ignored — it does not have a valid value on return), and a flag that, if nonzero, indicates that Daylight Saving time applies locally during the appropriate part of the year.</p>
SEE ALSO	date(1) , gettimeofday(3B) , ctime(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	ftw, nftw – walk a file tree
SYNOPSIS	<pre>#include <ftw.h> int ftw(const char *path, int (*fn) (const char *, const struct stat *, int), int depth); int nftw(const char *path, int (*fn) (const char *, const struct stat *, int, struct FTW*), int depth, int flags);</pre>
MT-LEVEL	See the NOTES section of this page.
DESCRIPTION	<p>ftw() 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 stat structure (see stat(2)) containing information about the object, and an integer. Possible values of the integer, defined in the <ftw.h> 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 stat 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 stat buffer passed to <i>fn</i> is undefined.</p> <p>ftw() 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 nonzero value, or some error is detected within ftw() (such as an I/O error). If the tree is exhausted, ftw() returns zero. If <i>fn</i> returns a nonzero value, ftw() stops its tree traversal and returns whatever value was returned by <i>fn</i>.</p>
RETURN VALUES	<p>If ftw() detects an error other than EACCES, it returns -1, and sets the error type in errno. The function nftw() is similar to ftw() except that it takes an additional argument, <i>flags</i>. The <i>flags</i> field is used to specify:</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.</p> <p>The function nftw() 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 stat buffer, the third is an integer giving additional information, and the fourth is a struct FTW that contains the following members:</p> <pre>int base;</pre>

int level;

base is the offset into the pathname of the base name of the object. **level** 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:

- FTW_F** The object is a file.
- FTW_D** The object is a directory.
- FTW_DP** The object is a directory and subdirectories have been visited.
- FTW_SL** The object is a symbolic link.
- FTW_SLN** The object is a symbolic link that points to a non-existent file.
- FTW_DNR** The object is a directory that cannot be read. *fn* will not be called for any of its descendants.
- FTW_NS** **stat** failed on the object because of lack of appropriate permission. The *stat* buffer passed to *fn* is undefined. **stat** failure other than lack of appropriate permission. **EACCES** is considered an error and **nftw()** will return **-1**.

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. *depth* must not be greater than the number of file descriptors currently available for use. **ftw()** 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 they have opened; they do not close any file descriptors that may have been opened by *fn*.

SEE ALSO **stat(2)**, **longjmp(3C)**, **malloc(3C)**

NOTES

Because **ftw()** is recursive, it is possible for it to terminate with a memory fault when applied to very deep file structures.

ftw() 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 nonzero value at its next invocation.

ftw() is safe in multi-thread applications. **nftw()** is safe in multi-thread applications when the **FTW_CHDIR** flag is not set.

NAME	getacinfo, getacdir, getacflg, getacmin, getacna, setac, endac – get audit control file information		
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -l<code>bsm</code> -l<code>socket</code> -l<code>nsi</code> -l<code>intl</code> [<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>		
MT-LEVEL	Safe.		
AVAILABILITY	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.		
DESCRIPTION	<p>When first called, getacdir() provides information about the first audit directory in the audit_control file; thereafter, it returns the next directory in the file. Successive calls list all the directories listed in audit_control(4). 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 audit_control file and returns the value in min_val. The minimum value specifies how full the file system to which the audit files are being written can get before the script audit_warn(1M) is invoked.</p> <p>getacflg() reads the system audit value from the audit_control file and returns the value in <i>auditstring</i>. The parameter <i>len</i> specifies the length of the buffer <i>auditstring</i>.</p> <p>getacna() reads the system audit value for non-attributable audit events from the audit_control file and returns the value in <i>auditstring</i>. The parameter <i>len</i> specifies the length of the buffer <i>auditstring</i>. Non-attributable events are events that cannot be attributed to an individual user. inetd(1M) and several other daemons record non-attributable events.</p> <p>Calling setac rewinds the audit_control file to allow repeated searches.</p> <p>Calling endac closes the audit_control file when processing is complete.</p>		
FILES	<table border="0"> <tr> <td style="padding-right: 20px;">/etc/security/audit_control</td> <td>contains default parameters read by the audit daemon, auditd(1M)</td> </tr> </table>	/etc/security/audit_control	contains default parameters read by the audit daemon, auditd(1M)
/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.

SEE ALSO

audit_warn(1M), **bsmconv(1M)**, **inetd(1M)**, **audit_control(4)**

NAME	getauclassnam, getauclassent, setauclass, endauclass, getauclassnam_r, getauclassent_r – get audit_class entry
SYNOPSIS	<pre>cc [<i>flag ...</i>] <i>file ...</i> -l<code>bsm</code> -l<code>socket</code> -l<code>insl</code> -l<code>intl</code> [<i>library ...</i>] #include <sys/param.h> #include <bsm/libbsm.h> struct au_class_ent *getauclassnam(const char *name); struct au_class_ent *getauclassnam_r(au_class_ent_t *, const char *name); struct au_class_ent *getauclassent(void); struct au_class_ent *getauclassent_r(au_class_ent_t *); void setauclass(void); void endauclass(void);</pre>
MT-LEVEL	<p>MT-Safe with exceptions.</p> <p>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.</p>
AVAILABILITY	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.
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> <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.</p> <p>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.</p> <p>The internal representation of an audit_user entry is an au_class_ent structure defined in <bsm/libbsm.h> with the following members:</p>

```
char      *ac_name;  
au_class_t ac_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

SEE ALSO **bsmconv(1M)**, **audit_class(4)**, **audit_event(4)**

NOTES All information is contained in a static area, so it must be copied if it is to be saved.

NAME	getauditflags, getauditflagsbin, getauditflagschar – convert audit flag specifications
SYNOPSIS	<pre>cc [<i>flag ...</i>] <i>file ...</i> -l<code>bsm</code> -l<code>socket</code> -l<code>insl</code> -l<code>intl</code> [<i>library ...</i>] #include <sys/param.h> #include <bsm/libbsm.h> int getauditflagsbin(char *auditstring, au_mask_t *masks); int getauditflagschar(char *auditstring, au_mask_t *masks, int verbose);</pre>
MT-LEVEL	MT-Safe.
AVAILABILITY	The functionality described in this man page is available only if the Basic Security Module (BSM) has been enabled. See <code>bsmconv(1M)</code> for more information.
DESCRIPTION	<p><code>getauditflagsbin()</code> 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><code>getauditflagschar()</code> 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	<code>getauditflagsbin()</code> and <code>getauditflagschar()</code> : -1 is returned on error and 0 on success.
SEE ALSO	<code>bsmconv(1M)</code> , <code>audit.log(4)</code> , <code>audit_control(4)</code>
BUGS	This is not a very extensible interface.

NAME	getauevent, getauevnam, getauevnum, getauevnonam, setauevent, endauevent, getauevent_r, getauevnam_r, getauevnum_r – get audit_user entry
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -l<code>bsm</code> -l<code>socket</code> -l<code>insl</code> -l<code>intl</code> [<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>, void); 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>
MT-LEVEL	<p>MT-Safe with exceptions.</p> <p>The functions <code>getauevent()</code>, <code>getauevnam()</code>, and <code>getauevnum()</code> are not MT-Safe; but, there are equivalent functions: <code>getauevent_r()</code>, <code>getauevnam_r()</code>, and <code>getauevnum_r()</code> - all of which provide the same functionality and a MT-Safe function call interface.</p>
AVAILABILITY	The functionality described in this man page is available only if the Basic Security Module (BSM) has been enabled. See <code>bsmconv(1M)</code> for more information.
DESCRIPTION	<p><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_r()</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 NULL.</p> <p><code>getauevnam_r()</code> and <code>getauevnam_r()</code> search for an <code>audit_event</code> entry with a given event name <i>name</i>.</p> <p><code>getauevnum_r()</code> and <code>getauevnum_r()</code> search for an <code>audit_event</code> entry with a given event number <i>number</i>.</p> <p><code>getauevnonum_r()</code> searches for an <code>audit_event</code> entry with a given event name <i>name</i> and returns the corresponding event number.</p> <p><code>setauevent_r()</code> “rewinds” to the beginning of the enumeration of <code>audit_event</code> entries. Calls to <code>getauevnam_r()</code>, <code>getauevnum_r()</code>, <code>getauevnonum_r()</code>, <code>getauevnam_r()</code>, or <code>getauevnum_r()</code> may leave the enumeration in an indeterminate state; so, <code>setauevent_r()</code> should be called before the first <code>getauevent_r()</code> or <code>getauevent_r()</code>.</p>

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()** and **getauevnum()** return a pointer to a **struct au_event_ent** if it successfully locates the requested entry; 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

SEE ALSO

bsmconv(1M), **getpwnam(3C)**, **getauclassent(3)**, **audit_class(4)**, **audit_event(4)**, **passwd(4)**

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.

NAME	getauusernam, getauuserent, setauuser, endauuser – get audit_user entry
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -l<code>bsm</code> -l<code>socket</code> -l<code>insl</code> -l<code>intl</code> [<i>library</i> ...] #include <sys/param.h> #include <bsm/libbsm.h> struct au_user_ent *getauusernam(const char *name); struct au_user_ent *getauuserent(void); void setauuser(void); void endauuser(void); struct au_user_ent *getauusernam_r(au_user_ent *u, const char *name); struct au_user_ent *getauuserent_r(au_user_ent *u);</pre>
MT-LEVEL	<p>MT-Safe with exceptions.</p> <p>The functions <code>getauusernam()</code> and <code>getauuserent()</code> are not MT-safe. However, the functions <code>getauusernam_r()</code> and <code>getauuserent_r()</code> provide the same functionality with MT-Safe interfaces.</p>
AVAILABILITY	The functionality described in this man page is available only if the Basic Security Module (BSM) has been enabled. See <code>bsmconv(1M)</code> for more information.
DESCRIPTION	<p><code>getauuserent()</code>, <code>getauusernam()</code>, <code>getauuserent_r()</code>, and <code>getauusernam_r()</code> each return an audit_user entry.</p> <p><code>getauusernam()</code> and <code>getauusernam_r()</code> search for an audit_user entry with a given login name <i>name</i>.</p> <p><code>getauuserent()</code> and <code>getauuserent_r()</code> enumerate audit_user entries: successive calls to these functions will return either successive audit_user entries or NULL.</p> <p><code>setauuser()</code> “rewinds” to the beginning of the enumeration of audit_user entries. Calls to <code>getauusernam()</code> and <code>getauusernam_r()</code> may leave the enumeration in an indeterminate state, so <code>setauuser()</code> should be called before the first <code>getauuserent()</code> or <code>getauuserent_r()</code>.</p> <p><code>endauuser()</code> 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.</p> <p>The two functions <code>getauuserent_r()</code> and <code>getauusernam_r()</code> both take an argument <i>u</i>, which is a pointer to an au_user_ent. This is the pointer that is returned on successful function calls.</p> <p>The internal representation of an audit_user entry is an <code>au_user_ent</code> structure defined in <code><bsm/libbsm.h></code> with the following members:</p> <pre>char *au_name; au_mask_t au_always; au_mask_t au_never;</pre>

RETURN VALUES

getausernam() returns a pointer to a **struct au_user_ent** if it successfully locates the requested entry; otherwise it returns **NULL**.

getauserent() 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.

FILES

/etc/security/audit_user Stores per-user audit event mask
/etc/passwd Stores user-id to username mappings

SEE ALSO

bsmconv(1M), **getpwnam(3C)**, **audit_user(4)**, **passwd(4)**

NOTES

All information for the functions **getauserent()** and **getausernam()** is contained in a static area, so it must be copied if it is to be saved.

NAME	getc, getc_unlocked, getchar, getchar_unlocked, fgetc, getw – get character or word from a stream
SYNOPSIS	<pre>#include <stdio.h> int getc(FILE *stream); int getc_unlocked(FILE *stream); int getchar(void); int getchar_unlocked(void); int fgetc(FILE *stream); int getw(FILE *stream);</pre>
MT-LEVEL	See the NOTES section of this page.
DESCRIPTION	<p>getc() returns the next character (that is, byte) from the named input <i>stream</i> (see intro(3)) as an unsigned char converted to an int. It also moves the file pointer, if defined, ahead one character in <i>stream</i>. getchar() is defined as getc(stdin). getc() and getchar() are macros.</p> <p>getc_unlocked() and getchar_unlocked() are respectively variants of getc() and getchar() that do not lock the stream. It is the caller's responsibility to acquire the stream lock before calling these functions and releasing the lock afterwards; see flockfile(3S) and stdio(3S).</p> <p>fgetc() behaves like getc(), but is a function rather than a macro. fgetc() runs more slowly than getc(), but it takes less space per invocation and its name can be passed as an argument to a function.</p> <p>getw() returns the next word (that is, integer) from the named input <i>stream</i>. getw() increments the associated file pointer, if defined, to point to the next word. The size of a word is the size of an integer and varies from machine to machine. getw() assumes no special alignment in the file.</p>
RETURN VALUES	These functions return the constant EOF at end-of-file or upon an error and set the EOF or error indicator of <i>stream</i> , respectively. Because <i>is</i> a valid integer, ferror() should be used to detect getw() errors.
SEE ALSO	intro(3) , fclose(3S) , ferror(3S) , flockfile(3S) , fopen(3S) , fread(3S) , gets(3S) , putc(3S) , scanf(3S) , stdio(3S) , ungetc(3S)
NOTES	If the integer value returned by getc() , getchar() , or fgetc() is stored into a character variable and then compared against the integer constant EOF , the comparison may never succeed, because sign-extension of a character on widening to integer is implementation dependent.

The macro version of **getc()** evaluates a *stream* argument more than once and may treat side effects incorrectly. In particular, **getc(*f++)** does not work sensibly. Use **fgetc()** instead.

Because of possible differences in word length and byte ordering, files written using **putw()** are implementation dependent, and may not be read using **getw()** on a different processor.

Functions exist for all the above-defined macros. To get the function form, the macro name must be undefined (for example, **#undef getc**).

fgetc(), **getc()**, **getchar()**, **getw()**, and **ungetc()** are MT-Safe in multi-thread applications. **getc_unlocked()** and **getchar_unlocked()** are unsafe in multi-thread applications.

NAME	getcwd – get pathname of current working directory
SYNOPSIS	#include <unistd.h> extern char *getcwd(char *buf, size_t size);
MT-LEVEL	Safe
DESCRIPTION	getcwd() 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. If <i>buf</i> is not NULL, the pathname will be stored in the space pointed to by <i>buf</i> . 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() .
RETURN VALUES	getcwd() returns NULL with errno set if <i>size</i> is not large enough, or if an error occurs in a lower-level function.
ERRORS	getcwd() will fail if one or more of the following are true: EACCES A parent directory cannot be read to get its name. EINVAL <i>size</i> is equal to 0. ERANGE <i>size</i> is greater than 0 and less than the length of the pathname plus 1.
EXAMPLE	Here is a program that prints the current working directory. <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>
SEE ALSO	chdir(2) , malloc(3C)
NOTES	Using chdir(2) in conjunction with getcwd can give unpredictable results.

NAME	getdate – convert user format date and time																																														
SYNOPSIS	<pre>#include <time.h> struct tm *getdate(const char *string); extern int getdate_err;</pre>																																														
MT-LEVEL	MT-Safe																																														
DESCRIPTION	<p>getdate() converts user-definable date and/or time specifications pointed to by <i>string</i> into a tm structure. The tm structure declaration is in the <time.h> header file.</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 DATEMSK. Each line in the template represents an acceptable date and/or time specification using conversion specifications similar to those used by strptime(3C) and strptime(3C). The first line in the template that matches the input specification is used for interpretation and conversion into the internal time format. If successful, the function getdate() returns a pointer to a tm structure; otherwise, it returns NULL and sets the global variable getdate_err to indicate the error.</p> <p>The following conversion specifications are supported:</p> <table border="0"> <tr><td>%%</td><td>same as %</td></tr> <tr><td>%a</td><td>locale's abbreviated weekday name</td></tr> <tr><td>%A</td><td>locale's full weekday name</td></tr> <tr><td>%b</td><td>locale's abbreviated month name</td></tr> <tr><td>%B</td><td>locale's full month name</td></tr> <tr><td>%c</td><td>locale's appropriate date and time representation</td></tr> <tr><td>%C</td><td>century number [0,99]; leading zero is permitted but not required</td></tr> <tr><td>%d</td><td>day of month [01,31]; leading zero is permitted but not required</td></tr> <tr><td>%D</td><td>date as %m/%d/%y</td></tr> <tr><td>%e</td><td>same as %d</td></tr> <tr><td>%h</td><td>locale's abbreviated month name</td></tr> <tr><td>%H</td><td>hour (24-hour clock) [0,23]; leading zero is permitted but not required</td></tr> <tr><td>%I</td><td>hour (12-hour clock) [1,12]; leading zero is permitted but not required</td></tr> <tr><td>%j</td><td>day number of the year [1,366]; leading zeros are permitted but not required</td></tr> <tr><td>%m</td><td>month number [1,12]; leading zero is permitted but not required</td></tr> <tr><td>%M</td><td>minute [0,59]; leading zero is permitted but not required</td></tr> <tr><td>%n</td><td>any white space</td></tr> <tr><td>%p</td><td>locale's equivalent of either a.m. or p.m.</td></tr> <tr><td>%r</td><td>appropriate time representation in the 12-hour clock format with %p</td></tr> <tr><td>%R</td><td>time as %H:%M</td></tr> <tr><td>%S</td><td>seconds [0,61]; leading zero is permitted but not required</td></tr> <tr><td>%t</td><td>any white space</td></tr> <tr><td>%T</td><td>time as %H:%M:%S</td></tr> </table>	%%	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	%C	century number [0,99]; leading zero is permitted but not required	%d	day of month [01,31]; leading zero is permitted but not required	%D	date as %m/%d/%y	%e	same as %d	%h	locale's abbreviated month name	%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	%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	%t	any white space	%T	time as %H:%M:%S
%%	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																																														
%C	century number [0,99]; leading zero is permitted but not required																																														
%d	day of month [01,31]; leading zero is permitted but not required																																														
%D	date as %m/%d/%y																																														
%e	same as %d																																														
%h	locale's abbreviated month name																																														
%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																																														
%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																																														
%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 the century [0,99]; leading zero is permitted but not required
%Y	year, including the century (for example, 1993)
%	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 behaviour will 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 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

On failure `getdate()` returns `NULL` and sets the variable `getdate_err` to indicate the error. The following is a complete list of the `getdate_err` settings and their meanings.

- | | |
|----------|--|
| 1 | The <code>DATMSK</code> 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 | <code>malloc()</code> 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). |

EXAMPLES

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
<code>getdate("11/27/86")</code>	<code>%m/%d/%y</code>
<code>getdate("27.11.86")</code>	<code>%d.%m.%y</code>
<code>getdate("86-11-27")</code>	<code>%y-%m-%d</code>
<code>getdate("Friday 12:00:00")</code>	<code>%A %H:%M:%S</code>

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

FILES */usr/lib/locale/locale/LC_TIME/time*
 locale specific date and time information
/usr/lib/locale/locale/LC_CTYPE/ctype
 character characterization information

SEE ALSO *setlocale(3C)*, *strftime(3C)*, *strptime(3C)*, *environ(5)*

NOTES Subsequent calls to *getdate()* alter the contents of *getdate_err*.
 Dates before 1970 and after 2037 are illegal.
 The range of values for %S is [00,61] rather than [00,59] to allow for the occasional leap second and even more occasional double leap second.
getdate() makes explicit use of macros described in *ctype(3C)*.

NAME	getdtablesize – get file descriptor table size
SYNOPSIS	<i>/usr/ucb/cc [flag ...] file ...</i> int getdtablesize()
DESCRIPTION	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.
SEE ALSO	getrlimit(2) , open(2) , pipe(2) , sysconf(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	getenv – return value for environment name
SYNOPSIS	#include <stdlib.h> char *getenv(const char *name);
MT-LEVEL	Safe
DESCRIPTION	getenv() 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.
SEE ALSO	exec(2) , putenv(3C) , environ(5)
NOTES	getenv() can be safely called from a multi-thread program. However, care must still be taken when using getenv() and putenv(3C) in a multi-thread program. These routines examine and modify the environment list. This list is shared by all threads in a program. The system prevents the list from being accessed simultaneously by two different threads. However, it does not prevent two threads from successively accessing the environment list using getenv() or putenv(3C) .

NAME	getfauditflags – generates the process audit state
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -l<code>bsm</code> -l<code>socket</code> -l<code>nsi</code> -l<code>intl</code> [<i>library</i> ...] #include <sys/param.h> #include <bsm/libbsm.h> int getfauditflags(au_mask_t *usremasks, au_mask_t *usrdmasks, au_mask_t *lastmasks);</pre>
MT-LEVEL	MT-Safe.
AVAILABILITY	The functionality described in this man page is available only if the Basic Security Module (BSM) has been enabled. See <code>bsmconv(1M)</code> for more information.
DESCRIPTION	<p><code>getfauditflags()</code> generates a process audit state by combining the audit masks passed as parameters with the system audit masks specified in the <code>audit_control(4)</code> file.</p> <p><code>getfauditflags()</code> obtains the system audit value by calling <code>getacflg()</code> (see <code>getacinfo(3)</code>).</p> <p><code>usremasks</code> points to <code>au_mask_t</code> fields which contains two values. The first value defines which events are <i>always</i> to be audited when they succeed. The second value defines which events are always to be audited when they fail.</p> <p><code>usrdmasks</code> also points to <code>au_mask_t</code> fields which contains two values. The first value defines which events are <i>never</i> to be audited when they succeed. The second value defines which events are never to be audited when they fail.</p> <p>The structures pointed to by <code>usremasks</code> and <code>usrdmasks</code> may be obtained from the <code>audit_user(4)</code> file by calling <code>getausernam()</code> which returns a pointer to a structure containing all <code>audit_user(4)</code> fields for a user.</p> <p>The output of this function is stored in <code>lastmasks</code> which is a pointer of type <code>au_mask_t</code> 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.</p> <p>Both <code>usremasks</code> and <code>usrdmasks</code> override the values in the system audit values.</p>
RETURN VALUES	-1 is returned on error and 0 on success.
SEE ALSO	<code>bsmconv(1M)</code> , <code>getacinfo(3)</code> , <code>getauditflags(3)</code> , <code>getausernam(3)</code> , <code>audit.log(4)</code> , <code>audit_control(4)</code> , <code>audit_user(4)</code>

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 *name); struct group *getgrnam_r(const char *name, struct group *result, char *buffer, int buflen); struct group *getgrent(void); struct group *getgrent_r(struct group *result, char *buffer, int buflen); struct group *getgrgid(gid_t gid); struct group *getgrgid_r(gid_t gid, struct group *result, char *buffer, int buflen); void setgrent(void); void endgrent(void); struct group *fgetgrent(FILE *f); struct group *fgetgrent_r(FILE *f, struct group *result, char *buffer, int buflen);</pre>
MT-LEVEL	See the subsection “Reentrant Interfaces” in the DESCRIPTION section of this page.
DESCRIPTION	<p>These functions are used to obtain entries describing Unix user groups. An entry may come from any of the sources for group specified in the <code>/etc/nsswitch.conf</code> file (see <code>nsswitch.conf(4)</code>).</p> <p>getgrnam() searches for an entry with the group name specified by the character string parameter <i>name</i>.</p> <p>getgrgid() searches for an entry with the (numeric) group id specified by the parameter <i>gid</i>.</p> <p>The functions setgrent(), getgrent(), and endgrent() are used to enumerate group entries from the database.</p> <p>setgrent() 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.</p> <p>Successive calls to getgrent() return either successive entries or NULL, indicating the end of the enumeration.</p> <p>endgrent() 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().</p> <p>fgetgrent(), unlike the other functions above, does not use <code>nsswitch.conf</code>; it reads and parses the next line from the stream <i>f</i>, which is assumed to have the format of the group file (see <code>group(4)</code>).</p>

Reentrant Interfaces

The functions **getgrnam()**, **getgrent()**, **getgruid()**, and **fgetgrent()** use static storage that is re-used in each call, making these routines unsafe for use in multithreaded applications.

The 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 parameter *result* 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* must be a pointer to a buffer supplied by the caller. This buffer is used as storage space for the group data. All of the pointers within the returned **struct group 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 group entry. The parameter *buffer* 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. **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 functions **getgrnam()**, **getgrnam_r()**, **getgrgid()**, and **getgrgid_r()** each return a pointer to a **struct group** if they successfully locate the requested entry; otherwise they return NULL.

The functions **getgrent()**, **getgrent_r()**, **fgetgrent()**, and **fgetgrent_r()** 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 functions **getgrnam()**, **getgrgid()**, **getgrent()**, and **fgetgrent()** 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 *result* pointer that was supplied by the caller.

ERRORS The reentrant functions **getgrnam_r()**, **getgrgid_r()**, **getgrent_r()**, and **fgetgrent_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/group*
/etc/nsswitch.conf

SEE ALSO **getpwnam(3C)**, **group(4)**, **nsswitch.conf(4)**, **passwd(4)**

WARNINGS The reentrant interfaces **getgrnam_r()**, **getgrgid_r()**, **getgrent_r()**, and **fgetgrent_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 Multithread Applications*, for information about the use of the **_REENTRANT** flag.

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.

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 <sys/types.h> #include <sys/socket.h> #include <netinet/in.h> #include <arpa/inet.h> #include <netdb.h> struct hostent *gethostbyname(const char *name); struct hostent *gethostbyname_r(const char *name, struct hostent *result, char *buffer, int buflen, int *h_errnop); struct hostent *gethostbyaddr(const char *addr, int len, int type); struct hostent *gethostbyaddr_r(const char *addr, int length, int type, struct hostent *result, char *buffer, int buflen, int *h_errnop); struct hostent *gethostent(void); struct hostent *gethostent_r(struct hostent *result, char *buffer, int buflen, int *h_errnop); int sethostent(int stayopen); int endhostent(void);</pre>
MT-LEVEL	See the subsection “Reentrant Interfaces” in the DESCRIPTION section of this page.
DESCRIPTION	<p>These functions are used to obtain entries describing hosts. An entry may come from any of the sources for hosts 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> <p>The functions sethostent(), gethostent(), and endhostent() are used to enumerate host entries from the database.</p> <p>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 <i>stayopen</i> flag is non-zero, the system may keep allocated resources such as open file descriptors until a subsequent call to endhostent().</p>

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 re-used in each call, making these functions unsafe for use in multithreaded 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 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 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 multithreaded applications, the position within the enumeration is a process-wide property shared by all threads. **sethostent()** may be used in a multithreaded 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 on 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.

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`. 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.

EXAMPLES

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)
{
    u_long addr;
    struct hostent *hp;
    char **p;

    if (argc != 2) {
        (void) printf("usage: %s IP-address\n", argv[0]);
        exit (1);
    }
    if ((int)(addr = inet_addr(argv[1])) == -1) {
        (void) printf("IP-address must be of the form a.b.c.d\n");
        exit (2);
    }

    hp = gethostbyaddr((char *)&addr, sizeof (addr), AF_INET);
    if (hp == NULL) {
        (void) printf("host information for %s not found\n", argv[1]);
        exit (3);
    }

    for (p = hp->h_addr_list; *p != 0; p++) {
        struct in_addr in;
        char **q;

        (void) memcpy(&in.s_addr, *p, sizeof (in.s_addr));
        (void) printf("%s\t%s", inet_ntoa(in), hp->h_name);
        for (q = hp->h_aliases; *q != 0; q++)
            (void) printf(" %s", *q);
        (void) putchar('\n');
    }
    exit (0);
}

```

Note that the above sample program is unsafe for use in multithreaded applications.

FILES /etc/hosts
 /etc/netconfig
 /etc/nsswitch.conf

SEE ALSO inet(3N), netdir(3N), hosts(4), netconfig(4), nsswitch.conf(4)

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 multithreaded 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	<i>/usr/ucb/cc [flag ...] file ...</i> long gethostid()
DESCRIPTION	gethostid() 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(1B) , sysinfo(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.

NAME	gethostname, sethostname – get/set name of current host
SYNOPSIS	<pre>/usr/ucb/cc [<i>flag ...</i>] <i>file ...</i> int gethostname(<i>name</i>, <i>namelen</i>) char *<i>name</i>; int <i>namelen</i>; int sethostname(<i>name</i>, <i>namelen</i>) char *<i>name</i>; int <i>namelen</i>;</pre>
DESCRIPTION	<p>gethostname() returns the standard host name for the current processor, as previously set by sethostname. The parameter <i>namelen</i> 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>sethostname() sets the name of the host machine to be <i>name</i>, which has length <i>namelen</i>. This call is restricted to the privileged user and is normally used only when the system is bootstrapped.</p>
RETURN VALUES	If the call succeeds a value of 0 is returned. If the call fails, then a value of –1 is returned and an error code is placed in the global location errno .
ERRORS	The following error may be returned by these calls: EFAULT The <i>name</i> or <i>namelen</i> parameter gave an invalid address. EPERM The caller was not the privileged user. Note: this error only applies to sethostname() .
SEE ALSO	uname(2) , sysinfo(2) , gethostid(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. Host names are limited to MAXHOSTNAMELEN characters, currently 256. (See the <param.h> header.)

NAME	gethrtime, gethrvtime – get high resolution time
SYNOPSIS	<pre>#include <sys/time.h> hrtime_t gethrtime(void); hrtime_t gethrvtime(void);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>gethrtime() 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, drifting, etc. via 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>gethrvtime() returns the current high-resolution LWP virtual time, expressed as total nanoseconds of execution time.</p> <p>gethrtime() and gethrvtime() both return an hrtime_t, which is a 64-bit (long long) signed integer.</p>
EXAMPLE	<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\n", (end - start) / iters);</pre>
SEE ALSO	adjtime(2) , gettimeofday(3C) , settimeofday(3C)
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	getlogin, getlogin_r – get login name
SYNOPSIS	<pre>#include <stdlib.h> char *getlogin(void); char *getlogin_r(char *name, int namelen);</pre>
MT-LEVEL	See the NOTES section of this page.
DESCRIPTION	<p>getlogin() returns a pointer to the login name as found in /var/adm/utmp. It may be used in conjunction with getpwnam() 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(), or to call getlogin() and if it fails to call getpwuid().</p> <p>getlogin_r() has the same functionality as getlogin() except that a buffer <i>name</i> with length <i>namelen</i> has to be supplied by the caller to store the result. <i>name</i> must be at least LOGNAME_MAX bytes in size (defined in limits.h).</p>
RETURN VALUES	Returns a null pointer if the login name is not found.
ERRORS	<p>getlogin_r() will fail if the following is true:</p> <p>ERANGE The size of the buffer is smaller than the result to be returned.</p>
FILES	/var/adm/utmp
SEE ALSO	cuserid(3S) , getgrnam(3C) , getpwnam(3C) , utmp(4)
NOTES	<p>The getlogin_r() 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> <p>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.</p> <p>The return values point to static data whose content is overwritten by each call.</p> <p>getlogin() is unsafe in multi-thread applications. getlogin_r() should be used instead.</p>

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>						
MT-LEVEL	Safe						
DESCRIPTION	<p>getmntent() and getmntany() each fill in the structure pointed to by <i>mp</i> with the broken-out fields of a line in the /etc/mnttab file. Each line in the file contains a mnttab structure, which is declared in the <sys/mnttab.h> header. The structure contains the following members:</p> <pre>char *mnt_special; char *mnt_mountp; char *mnt_fstype; char *mnt_mntopts; char *mnt_time;</pre> <p>The fields have meanings described in mnttab(4).</p> <p>getmntent() returns a pointer to the next mnttab structure in the file; so successive calls can be used to search the entire file. getmntany() searches the file referenced by <i>fp</i> until a match is found between a line in the file and <i>mpref</i>. <i>mpref</i> matches the line 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>hasmntopt() scans the mnt_mntopts field of the mnttab structure <i>mnt</i> for a substring that matches <i>opt</i>. It returns the address of the substring if a match is found, otherwise it returns 0.</p> <p>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. Note: the file should be opened in append mode (fopen(3S) with an "a" mode) so that the entry is appended to the file.</p>						
RETURN VALUES	<p>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 possible error values are:</p> <table border="0"> <tr> <td style="padding-right: 1em;">MNT_TOOLONG</td> <td>A line in the file exceeded the internal buffer size of MNT_LINE_MAX.</td> </tr> <tr> <td style="padding-right: 1em;">MNT_TOOMANY</td> <td>A line in the file contains too many fields.</td> </tr> <tr> <td style="padding-right: 1em;">MNT_TOOFEW</td> <td>A line in the file contains too few fields.</td> </tr> </table>	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.
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.						

On success, **putmntent()** returns the number of bytes printed to the specified file and on failure returns EOF.

FILES /etc/mnttab

SEE ALSO mnttab(4)

NOTES The members of the **mnttab** structure point to information contained in a static area, so it must be copied if it is to be saved.

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 *name); struct netent *getnetbyname_r(const char *name, struct netent *result, char *buffer, int buflen); struct netent *getnetbyaddr(long net, int type); struct netent *getnetbyaddr_r(long net, int type, struct netent *result, char *buffer, int buflen); struct netent *getnetent(void); struct netent *getnetent_r(struct netent *result, char *buffer, int buflen); int setnetent(int stayopen); int endnetent(void);</pre>
MT-LEVEL	See the subsection “Reentrant Interfaces” in the DESCRIPTION section of this page.
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>The functions setnetent(), getnetent(), and endnetent() are used to enumerate network entries from the database.</p> <p>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 <i>stayopen</i> flag is non-zero, the system may keep allocated resources such as open file descriptors until a subsequent call to endnetent().</p> <p>Successive calls to getnetent() return either successive entries or NULL, indicating the end of the enumeration.</p> <p>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().</p>

Reentrant Interfaces

The functions **getnetbyname()**, **getnetbyaddr()**, and **getnetent()** use static storage that is re-used in each call, making these routines unsafe for use in multithreaded 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 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 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 multithreaded applications, the position within the enumeration is a process-wide property shared by all threads. **setnetent()** may be used in a multithreaded application but resets the enumeration position for all threads. If multiple threads interleave calls to **getnetent_r()**, the threads will enumerate disjoint subsets of the network database.

Like their non-reentrant counterparts, **getnetbyname_r()** and **getnetbyaddr_r()** leave the enumeration position in an indeterminate state.

RETURN VALUES

Network entries are represented by the **struct netent** structure defined in `<netdb.h>`:

```
struct netent {  
    char   *n_name;  
    char   **n_aliases;  
    int    n_addrtype;  
    long   n_net;  
};
```

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 multithreaded applications.

FILES

/etc/networks
/etc/nsswitch.conf

SEE ALSO

inet(3N), **networks(4)**, **nsswitch.conf(4)**

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 multithreaded 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>
MT-LEVEL	MT-Safe
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 netconfig database, Network Selection includes the environment variable <code>NETPATH</code> (see environ(5)) and the <code>NETPATH</code> access routines described in getnetpath(3N).</p> <p>getnetconfig() returns a pointer to the current entry in the netconfig database, formatted as a struct netconfig. Successive calls will return successive netconfig entries in the netconfig database. getnetconfig() can be used to search the entire netconfig 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 netconfig 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(). 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 struct netconfig data structure. endnetconfig() may not be called before setnetconfig().</p> <p>getnetconfigent() returns a pointer to the struct netconfig structure corresponding to <i>netid</i>. It returns NULL if <i>netid</i> is invalid (that is, does not name an entry in the netconfig database).</p> <p>freenetconfigent() frees the netconfig structure pointed to by <i>netconfigp</i> (previously returned by getnetconfigent()).</p> <p>nc_perror() prints a message to the standard error indicating why any of the above routines failed. The message is prepended with the string <i>msg</i> and a colon. A NEWLINE is appended at the end of the message.</p>

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.

SEE ALSO

getnetpath(3N), **netconfig(4)**, **environ(5)**

Network Interfaces Programmer's Guide
File System Administration

NAME	getnetgrent, getnetgrent_r, setnetgrent, endnetgrent, inetgr – get network group entry
SYNOPSIS	<pre> int getnetgrent(char **<i>machinep</i>, char **<i>userp</i>, char **<i>domainp</i>); int getnetgrent_r(char **<i>machinep</i>, char **<i>userp</i>, char **<i>domainp</i>, char *<i>buffer</i>, int <i>buflen</i>); int setnetgrent(const char *<i>netgroup</i>); int endnetgrent(void); int inetgr(const char *<i>netgroup</i>, const char *<i>machine</i>, const char *<i>user</i>, const char *<i>domain</i>); </pre>
MT-LEVEL	See the DESCRIPTION section of this page.
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 netgroup in the /etc/nsswitch.conf file (see nsswitch.conf(4)).</p> <p>The function inetgr() returns 1 if there is a netgroup <i>netgroup</i> that contains the specified <i>machine</i>, <i>user</i>, <i>domain</i> triple as a member; otherwise it returns 0. Any of the supplied pointers <i>machine</i>, <i>user</i>, and <i>domain</i> may be NULL, signifying a "wild card" that matches all values in that position of the triple.</p> <p>The inetgr() function is safe for use in single-threaded and multi-threaded 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 <i>netgroup</i> 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>Upon successful return from getnetgrent(), the pointer <i>machinep</i> points to a string containing the name of the machine part of the member triple, <i>userp</i> points to a string containing the user name and <i>domainp</i> points to a string containing the domain name. If the pointer returned for <i>machinep</i>, <i>userp</i>, or <i>domainp</i> is NULL, it signifies that the element of the netgroup contains wild card specifier in that position of the triple.</p> <p>The pointers returned by getnetgrent() point into a buffer allocated by setnetgrent() that is re-used by in 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.</p> <p>The function getnetgrent_r() is similar to getnetgrent() but uses a buffer supplied by the caller for the space needed to store the results. The parameter <i>buffer</i> should be a pointer to a buffer allocated by the caller and the length of this buffer should be specified by the</p>

parameter *bufen*. 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.

FILES */etc/nsswitch.conf*

SEE ALSO **netgroup(4)** **nsswitch.conf(4)**,

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>
MT-LEVEL	MT-Safe
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, <code>/etc/netconfig</code>, as it is “filtered” by the <code>NETPATH</code> environment variable (see <code>environ(5)</code>). See <code>getnetconfig(3N)</code> for other routines that also access the network configuration database directly. The <code>NETPATH</code> variable is a list of colon-separated network identifiers.</p> <p><code>getnetpath()</code> returns a pointer to the <code>netconfig</code> database entry corresponding to the first valid <code>NETPATH</code> component. The <code>netconfig</code> entry is formatted as a <code>struct netconfig</code>. On each subsequent call, <code>getnetpath()</code> returns a pointer to the <code>netconfig</code> entry that corresponds to the next valid <code>NETPATH</code> component. <code>getnetpath()</code> can thus be used to search the <code>netconfig</code> database for all networks included in the <code>NETPATH</code> variable. When <code>NETPATH</code> has been exhausted, <code>getnetpath()</code> returns NULL.</p> <p>A call to <code>setnetpath()</code> “binds” to or “rewinds” <code>NETPATH</code>. <code>setnetpath()</code> must be called before the first call to <code>getnetpath()</code> and may be called at any other time. It returns a handle that is used by <code>getnetpath()</code>.</p> <p><code>getnetpath()</code> silently ignores invalid <code>NETPATH</code> components. A <code>NETPATH</code> component is invalid if there is no corresponding entry in the <code>netconfig</code> database.</p> <p>If the <code>NETPATH</code> variable is <i>unset</i>, <code>getnetpath()</code> behaves as if <code>NETPATH</code> were set to the sequence of “default” or “visible” networks in the <code>netconfig</code> database, in the order in which they are listed.</p> <p><code>endnetpath()</code> may be called to “unbind” from <code>NETPATH</code> when processing is complete, releasing resources for reuse. Programmers should be aware, however, that <code>endnetpath()</code> frees all memory allocated by <code>getnetpath()</code> for the <code>struct netconfig</code> data structure. <code>endnetpath()</code> returns <code>0</code> on success and <code>-1</code> on failure (for example, if <code>setnetpath()</code> was not called previously).</p>
RETURN VALUES	<code>setnetpath()</code> returns a handle that is used by <code>getnetpath()</code> . In case of an error, <code>setnetpath()</code> returns NULL. <code>nc_perror()</code> or <code>nc_sperror()</code> can be used to print out the reason for failure. See <code>getnetconfig(3N)</code> .

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).

SEE ALSO

getnetconfig(3N), **netconfig(4)**, **environ(5)**

Network Interfaces Programmer's Guide

File System Administration

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>
MT-LEVEL	Unsafe
DESCRIPTION	<p>getopt() 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><i>optstring</i> 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. <i>optarg</i> is set to point to the start of the option argument on return from getopt().</p> <p>getopt() 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	getopt() 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 opterr to 0. The value of the character that caused the error is in optopt .
EXAMPLES	<p>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:</p> <pre>#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;</pre>

```

char *ofile = NULL;

while ((c = getopt(argc, argv, "abo:")) != EOF)
    switch (c) {
        case 'a':
            if (bflg)
                errflg++;
            else
                aflg++;
            break;
        case 'b':
            if (aflg)
                errflg++;
            else
                bflg++;
            break;
        case 'o':
            ofile = optarg;
            (void)printf("ofile = %s\n", ofile);
            break;
        case '?':
            errflg++;
    }
if (errflg) {
    (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;
}

```

SEE ALSO [intro\(1\)](#), [getopts\(1\)](#), [getopt\(3C\)](#), [getsubopt\(3C\)](#), [setlocale\(3C\)](#), [gettext\(3I\)](#)

NOTES 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 library routine `getopt()` does not fully check for mandatory arguments. That is, given an option string `a:b` and the input `-a -b`, `getopt()` assumes that `-b` is the mandatory argument to the `-a` option and not that `-a` is missing a mandatory argument.

It is a violation of the command syntax standard (see **intro(1)**) for options with arguments to be grouped with other options, as in **cmd -abo filename**, where **a** and **b** are options, **o** is an option that requires an argument, and *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.

NAME	getpagesize – get system page size
SYNOPSIS	<i>/usr/ucb/cc [flag ...] file ...</i> int getpagesize()
DESCRIPTION	getpagesize() returns the number of bytes in a page. Page granularity is the granularity of many of the memory management calls. The page size is a system page size and need not be the same as the underlying hardware page size.
SEE ALSO	pagesize(1B) , brk(2) , sysconf(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	getpass – read a password
SYNOPSIS	#include <stdlib.h> char *getpass(const char *prompt);
MT-LEVEL	Unsafe
DESCRIPTION	getpass() reads up to a newline or EOF from the file /dev/tty , after prompting on the standard error output with the null-terminated string <i>prompt</i> and disabling echoing. A pointer is returned to a null-terminated string of at most 8 characters. If /dev/tty cannot be opened, a null pointer is returned. An interrupt will terminate input and send an interrupt signal to the calling program before returning.
FILES	/dev/tty
NOTES	The return value points to static data whose content is overwritten by each call.

NAME	getpeername – get name of connected peer
SYNOPSIS	cc [<i>flag ...</i>] <i>file ...</i> -lsocket -lnsl [<i>library ...</i>] int getpeername(int <i>s</i> , struct sockaddr * <i>name</i> , int * <i>namelen</i>);
MT-LEVEL	Safe
DESCRIPTION	getpeername() returns the name of the peer connected to socket <i>s</i> . The int pointed to by 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 of the name returned (in bytes). 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.
SEE ALSO	accept(3N) , bind(3N) , getsockname(3N) , socket(3N)

NAME	getpriority, setpriority – get/set scheduling priority for process, process group or user
SYNOPSIS	<pre> /usr/ucb/cc [flag ...] file ... #include <sys/time.h> #include <sys/resource.h> int getpriority(which, who) int which; int who; int setpriority(which, who, prio) int which; int who; int prio; </pre>
DESCRIPTION	<p>The scheduling priority of the process, process group, or user, as indicated by <i>which</i> and <i>who</i> is obtained with getpriority() and set with setpriority() The default priority is 0; lower priorities cause more favorable scheduling.</p> <p><i>which</i> is one of PRIO_PROCESS, PRIO_PGRP, or PRIO_USER, and <i>who</i> is interpreted relative to <i>which</i> (a process identifier for PRIO_PROCESS, process group identifier for PRIO_PGRP, and a user ID for PRIO_USER). A zero value of <i>who</i> denotes the current process, process group, or user.</p> <p>getpriority() returns the highest priority (lowest numerical value) enjoyed by any of the specified processes. setpriority() sets the priorities of all of the specified processes to the value specified by <i>prio</i>. If <i>prio</i> is less than -20, a value of -20 is used; if it is greater than 20, a value of 20 is used. Only the privileged user may lower priorities.</p>
RETURN VALUES	Since getpriority() can legitimately return the value -1 , it is necessary to clear the external variable errno prior to the call, then check it afterward to determine if a -1 is an error or a legitimate value. The setpriority() call returns 0 if there is no error, or -1 if there is.
ERRORS	<p>getpriority() and setpriority() may return one of the following errors:</p> <p>EINVAL <i>which</i> was not one of PRIO_PROCESS, PRIO_PGRP, or PRIO_USER.</p> <p>ESRCH No process was located using the <i>which</i> and <i>who</i> values specified.</p> <p>In addition to the errors indicated above, setpriority() may fail with one of the following errors returned:</p> <p>EPERM A process was located, but one of the following is true:</p> <ul style="list-style-type: none"> • Neither its effective nor real user ID matched the effective user ID of the caller, and neither the effective nor the real user ID of the process executing the setpriority() was the privileged user. • The call to getpriority() would have changed a process' priority to a value lower than its current value, and the effective user ID of the process executing the call was not that of the privileged user.

SEE ALSO**nice(1), renice(1B), fork(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.

It is not possible for the process executing **setpriority()** to lower any other process down to its current priority, without requiring privileged user permissions.

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 *name); struct protoent *getprotobyname_r(const char *name, struct protoent *result, char *buffer, int buflen); struct protoent *getprotobynumber(int proto); struct protoent *getprotobynumber_r(int proto, struct protoent *result, char *buffer, int buflen); struct protoent *getprotoent(void); struct protoent *getprotoent_r(struct protoent *result, char *buffer, int buflen); int setprotoent(int stayopen); int endprotoent(void);</pre>
MT-LEVEL	See the NOTES section of this page.
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 struct protoent 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> <p>getprotobyname_r(), getprotobynumber_r(), and getprotoent_r() each return a protocol entry.</p> <p>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.</p> <p>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.</p>

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:

```
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`

SEE ALSO [intro\(3\)](#), [nsswitch.conf\(4\)](#), [protocols\(4\)](#)

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]); int getsecretkey(const char netname[MAXNETNAMELEN], char secretkey[HEXKEYBYTES], const char *passwd);</pre>
MT-LEVEL	Safe
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 /etc/publickey 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 /etc/nsswitch.conf file (see nsswitch.conf(4)).</p> <p>getsecretkey() has an extra argument, <i>passwd</i>, 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 <i>secretkey</i> [0] will be set to NULL.
SEE ALSO	secure_rpc(3N) , nsswitch.conf(4) , publickey(4)
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	#include <stdlib.h> int getpw(uid_t uid, char *buf);
MT-LEVEL	Safe
DESCRIPTION	getpw() 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. This routine is included only for compatibility with prior systems and should not be used; see getpwnam(3C) for routines to use instead.
RETURN VALUES	getpw() returns non-zero on error.
FILES	/etc/passwd
SEE ALSO	getpwnam(3C) , passwd(4)
NOTES	If the /etc/passwd and the /etc/group files have the “+” for the NIS entry, then getpwent() and getgwent() will not return NULL when the end of file is reached.

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 *name); struct passwd *getpwnam_r(const char *name, struct passwd *result, char *buffer, int buflen); struct passwd *getpwent(void); struct passwd *getpwent_r(struct passwd *result, char *buffer, int buflen); struct passwd *getpwuid(uid_t uid); struct passwd *getpwuid_r(uid_t uid, struct passwd *result, char *buffer, int buflen); void setpwent(void); void endpwent(void); struct passwd *fgetpwent(FILE *f); struct passwd *fgetpwent_r(FILE *f, struct passwd *result, char *buffer, int buflen);</pre>
MT-LEVEL	See the subsection “Reentrant Interfaces” in the DESCRIPTION section of this page.
DESCRIPTION	<p>These functions are used to obtain password entries. An entry may come from any of the sources for passwd specified in the <code>/etc/nsswitch.conf</code> file. See <code>nsswitch.conf(4)</code>.</p> <p>getpwnam() searches for a password entry with the login name specified by the character string parameter <i>name</i>.</p> <p>getpwuid() searches for a password entry with the (numeric) user id specified by the parameter <i>uid</i>.</p> <p>The functions setpwent(), getpwent(), and endpwent() are used to enumerate password entries from the database.</p> <p>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.</p> <p>endpwent() 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().</p> <p>fgetpwent(), unlike the other functions above, does not use <code>nsswitch.conf</code>; it reads and parses the next line from the stream <i>f</i>, which is assumed to have the format of the passwd file. See <code>passwd(4)</code>.</p>

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 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 *result* 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* must be a pointer to a buffer supplied by the caller. This buffer is used as storage space for the password data. All of the pointers within the returned **struct passwd 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 password 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. **setpwent()** 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 functions **getpwnam()**, **getpwnam_r()**, **getpwuid()**, and **getpwuid_r()** each return a pointer to a **struct passwd** if they successfully locate the requested entry; otherwise they return NULL.

The functions **getpwent()**, **getpwent_r()**, **fgetpwent()**, and **fgetpwent_r()** 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 functions **getpwnam()**, **getpwuid()**, **getpwent()**, and **fgetpwent()** 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 *result* 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** 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/passwd
/etc/shadow
/etc/nsswitch.conf

SEE ALSO

nispasswd(1), **passwd(1)**, **yppasswd(1)**, **intro(2)**, **intro(3)**, **getgrnam(3C)**, **getlogin(3C)**, **getspnam(3C)**, **cuserid(3S)**, **nsswitch.conf(4)**, **passwd(4)**, **shadow(4)**

WARNINGS

The reentrant interfaces **getpwnam_r()**, **getpwent_r()**, **getpwuid_r()**, and **fgetpwent_r()** are included in this release on an uncommitted basis only, and are subject to change or removal in future minor releases.

NOTES

The **pw_passwd** field in the **passwd** structure should not be used as the encrypted password for the user; use **getspnam()** or **getspnam_r()** instead. See **getspnam(3C)**.

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 Multithread Applications*, for information about the use of the **_REENTRANT** flag.

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 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 **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)**.

NAME	getrpcbyname, getrpcbyname_r, getrpcbynumber, getrpcbynumber_r, getrpcent, getrpcent_r, setrpcent, endrpcent – 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 *getrpcent(void); struct rpcent *getrpcent_r(struct rpcent *<i>result</i>, char *<i>buffer</i>, int <i>buflen</i>); void setrpcent(const int <i>stayopen</i>); void endrpcent(void);</pre>
MT-LEVEL	See the subsection “Reentrant Interfaces” in the DESCRIPTION section of this page.
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 rpc 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 setrpcent(), getrpcent(), and endrpcent() are used to enumerate RPC entries from the database.</p> <p>setrpcent() sets (or resets) the enumeration to the beginning of the set of RPC entries. This function should be called before the first call to getrpcent(). Calls to getrpcbyname() and getrpcbynumber() leave the enumeration position in an indeterminate state. If the <i>stayopen</i> flag is non-zero, the system may keep allocated resources such as open file descriptors until a subsequent call to endrpcent().</p> <p>Successive calls to getrpcent() return either successive entries or NULL, indicating the end of the enumeration.</p> <p>endrpcent() 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 endrpcent().</p>

Reentrant Interfaces

The functions **getrpcbyname()**, **getrpcbynumber()**, and **getrpccent()** 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

```
getrpccent_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 rpcent** 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 rpcent 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. **setrpccent()** may be used in a multithreaded application but resets the enumeration position for all threads. If multiple threads interleave calls to **getrpccent_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 rpcent** structure defined in `<rpc/rpcent.h>`:

```
struct rpcent {  
    char *r_name;        /* name of this rpc service */  
    char **r_aliases;   /* zero-terminated list of  
                          alternate names */  
    long 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 **getrpccent()** and **getrpccent_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 **getrpccent()** 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 **getrpccent_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 **getrpccent_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

SEE ALSO

rpcinfo(1M), **rpc(3N)**, **nsswitch.conf(4)**, **rpc(4)**

WARNINGS

The reentrant interfaces **getrpcbyname_r()**, **getrpcbynumber_r()**, and **getrpccent_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 Multithread Applications*, for information about the use of the **_REENTRANT** flag.

Use of the enumeration interfaces **getrpccent()** and **getrpccent_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> /usr/ucb/cc [flag ...] file ... #include <sys/time.h> #include <sys/rusage.h> int getrusage(who, rusage) int who; struct rusage *rusage; </pre>
DESCRIPTION	<p>getrusage() returns information about the resources utilized by the current process, or all its terminated child processes. The interpretation for some values reported, such as ru_idrss, are dependent on the clock tick interval. This interval is an implementation dependent value.</p> <p>The who parameter is one of RUSAGE_SELF or RUSAGE_CHILDREN. The buffer to which rusage points will be filled in with a structure with the following members:</p> <pre> struct timeval ru_utime; /* user time used */ struct timeval ru_stime; /* system time used */ int ru_maxrss; /* maximum resident set size */ int ru_idrss; /* integral resident set size */ int ru_minflt; /* page faults not requiring physical I/O */ int ru_majflt; /* page faults requiring physical I/O */ int ru_nswap; /* swaps */ int ru_inblock; /* block input operations */ int ru_oublock; /* block output operations */ int ru_msgsnd; /* messages sent */ int ru_msrvcv; /* messages received */ int ru_nsignals; /* signals received */ int ru_nvcsw; /* voluntary context switches */ int ru_nivcsw; /* involuntary context switches */ </pre> <p>The fields are interpreted as follows:</p> <p>ru_utime The total amount of time spent executing in user mode. Time is given in seconds and microseconds.</p> <p>ru_stime The total amount of time spent executing in system mode. Time is given in seconds and microseconds.</p> <p>ru_maxrss The maximum resident set size. Size is given in pages (the size of a page, in bytes, is given by the getpagesize(3B) function). See the NOTES notes section of this page.</p>

ru_idrss	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 notes section of this page.
ru_minflt	The number of page faults serviced which did not require any physical I/O activity. See the NOTES notes section of this page.
ru_majflt	The number of page faults serviced which required physical I/O activity. This could include page ahead operations by the kernel. See the NOTES notes section of this page.
ru_nswap	The number of times a process was swapped out of main memory.
ru_inblock	The number of times the file system had to perform input in servicing a read(2) request.
ru_oublock	The number of times the file system had to perform output in servicing a write(2) request.
ru_msgsnd	The number of messages sent over sockets.
ru_msgrcv	The number of messages received from sockets.
ru_nsignals	The number of signals delivered.
ru_nvcsw	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).
ru_nivcsw	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

If successful, the value of the appropriate structure is filled in, and **0** is returned. If the call fails, **-1** is returned.

ERRORS

getrusage() will fail if:

EFAULT	The address specified by the <i>rusage</i> argument is not in a valid portion of the process’s address space.
EINVAL	The who parameter is not a valid value.

SEE ALSO

sar(1M), **read(2)**, **times(2)**, **wait(2)**, **write(2)**, **getpagesize(3B)**, **gettimeofday(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.

Only the *timeval* fields 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	#include <stdio.h> char *gets(char *s); char *fgets(char *s, int n, FILE *stream);
MT-LEVEL	MT-Safe
DESCRIPTION	<p>gets() reads characters from the standard input stream (see intro(3)), stdin, into the array pointed to by <i>s</i>, 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>fgets() reads characters from the <i>stream</i> into the array pointed to by <i>s</i>, until <i>n</i>–1 characters are read, or a newline character is read and transferred to <i>s</i>, 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 <i>s</i>, indeterminate behavior may result. For this reason, it is strongly recommended that gets() be avoided in favor of fgets().</p>
RETURN VALUES	If end-of-file is encountered and no characters have been read, no characters are transferred to <i>s</i> 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 <i>s</i> is returned.
SEE ALSO	lseek(2), read(2), ferror(3S), fopen(3S), fread(3S), getc(3S), scanf(3S), stdio(3S), ungetc(3S)

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 *name, const char *proto); struct servent *getservbyname_r(const char *name, const char *proto, struct servent *result, char *buffer, int buflen); struct servent *getservbyport(int port, const char *proto); struct servent *getservbyport_r(int port, const char *proto, struct servent *result, char *buffer, int buflen); struct servent *getservent(void); struct servent *getservent_r(struct servent *result, char *buffer, int buflen); int setservent(int stayopen); int endservent(void);</pre>
MT-LEVEL	See the subsection “Reentrant Interfaces” in the DESCRIPTION section of this man page.
DESCRIPTION	<p>These functions are used to obtain entries for Internet services. An entry may come from any of the sources for services specified in the <code>/etc/nsswitch.conf</code> file. See <code>nsswitch.conf(4)</code>.</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> <p>The string <i>proto</i> is used by both getservbyname() and getservbyport() to restrict the search to entries with the specified protocol. If <i>proto</i> is NULL, entries with any protocol may be returned.</p> <p>The functions setservent(), getservent(), and endservent() are used to enumerate entries from the services database.</p> <p>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 <i>stayopen</i> flag is non-zero, the system may keep allocated resources such as open file descriptors until a subsequent call to endservent().</p>

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:

s_name	The official name of the service.
s_aliases	A zero terminated list of alternate names for the service.
s_port	The port number at which the service resides. Port numbers are returned in network byte order.
s_proto	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 **getservent()** 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

SEE ALSO

intro(2), **intro(3)**, **netdir(3N)**, **netconfig(4)**, **nsswitch.conf(4)**, **services(4)**

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 name-toaddr 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 [<i>flag</i> ...] <i>file</i> ... -lsocket -lnsl [<i>library</i> ...] #include <sys/types.h> #include <sys/socket.h> int getsockname(int <i>s</i>, struct sockaddr *<i>name</i>, int *<i>namelen</i>);</pre>
MT-LEVEL	Safe
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: 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.
SEE ALSO	bind(3N) , getpeername(3N) , socket(3N)

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 <i>s</i>, int <i>level</i>, int <i>optname</i>, char *<i>optval</i>, int *<i>optlen</i>); int setsockopt(int <i>s</i>, int <i>level</i>, int <i>optname</i>, const char *<i>optval</i>, int <i>optlen</i>);</pre>														
MT-LEVEL	Safe														
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 SOL_SOCKET. 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. <i>optname</i> and any specified options are passed uninterpreted to the appropriate protocol module for interpretation. The include file <sys/socket.h> 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. SO_LINGER uses a struct linger parameter that specifies 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:</p> <table border="0" style="margin-left: 2em;"> <tr> <td style="padding-right: 2em;">l_onoff</td> <td>on = 1/off = 0</td> </tr> <tr> <td>l_linger</td> <td>linger time, in seconds</td> </tr> </table> <p>The following options are recognized at the socket level. Except as noted, each may be examined with getsockopt() and set with setsockopt().</p> <table border="0" style="margin-left: 2em;"> <tr> <td style="padding-right: 2em;">SO_DEBUG</td> <td>enable/disable recording of debugging information</td> </tr> <tr> <td>SO_REUSEADDR</td> <td>enable/disable local address reuse</td> </tr> <tr> <td>SO_KEEPAIVE</td> <td>enable/disable keep connections alive</td> </tr> <tr> <td>SO_DONTROUTE</td> <td>enable/disable routing bypass for outgoing messages</td> </tr> <tr> <td>SO_LINGER</td> <td>linger on close if data is present</td> </tr> </table>	l_onoff	on = 1/off = 0	l_linger	linger time, in seconds	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
l_onoff	on = 1/off = 0														
l_linger	linger time, in seconds														
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_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_KEEPAVIVE** 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. The Internet protocols place an absolute limit of 64 Kbytes on these values for **UDP** and **TCP** sockets.

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 *s* 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.

SEE ALSO

close(2), **ioctl(2)**, **bind(3N)**, **getprotobyname(3N)**, **socket(3N)**

NAME getspnam, getspnam_r, getspent, getspent_r, setspent, endspent, fgetspent, fgetspent_r – get password entry

SYNOPSIS

```
#include <shadow.h>
struct spwd *getspnam(const char *name);
struct spwd *getspnam_r(const char *name, struct spwd *result, char *buffer,
    int buflen);
struct spwd *getspent(void);
struct spwd *getspent_r(struct spwd *result, char *buffer, int buflen);
void setspent(void);
void endspent(void);
struct spwd *fgetspent(FILE *fp);
struct spwd *fgetspent_r(FILE *fp, struct spwd *result, char *buffer, int buflen);
```

MT-LEVEL See the subsection “Reentrant Interfaces” in the **DESCRIPTION** section of this page.

DESCRIPTION These functions are used to obtain shadow password entries. An entry may come from any of the sources for **shadow** specified in the `/etc/nsswitch.conf` file (see `nsswitch.conf(4)`).

getspnam() searches for a shadow password entry with the login name specified by the character string parameter *name*.

The functions **setspent()**, **getspent()**, and **endspent()** are used to enumerate shadow password entries from the database.

setspent() 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 **getspent()**. Calls to **getspnam()** leave the enumeration position in an indeterminate state.

Successive calls to **getspent()** return either successive entries or NULL, indicating the end of the enumeration.

endspent() 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 **endspent()**.

fgetspent(), 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 **shadow** file (see `shadow(4)`).

Reentrant Interfaces The functions **getspnam()**, **getspent()**, and **fgetspent()** use static storage that is re-used in each call, making these routines unsafe for use in multithreaded applications.

The functions:

```
    getspnam_r(),  
    getspent_r(),
```

and

```
    fgetspent_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 spwd** structure allocated by the caller. On successful completion, the function returns the shadow password 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 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 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. **setspent()** 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 information.

The functions **getspnam()** and **getspnam_r()** each return a pointer to a **struct spwd** if they successfully locate the requested entry; otherwise they return NULL.

The functions **getspent()**, **getspent_r()**, **fgetspent()**, and **fggetspent()** 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 functions **getspnam()**, **getspent()**, and **fgetspent()** 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 **fggetspent_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 **fggetspent_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/shadow**
/etc/nsswitch.conf
/etc/passwd

SEE ALSO **nispasswd(1)**, **passwd(1)**, **yppasswd(1)**, **intro(3)** **getlogin(3C)**, **getpwnam(3C)**, **nsswitch.conf(4)**, **passwd(4)**, **shadow(4)**

WARNINGS The reentrant interfaces **getspnam_r()**, **getspent_r()**, and **fggetspent_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 Multithread 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, const char * const *tokens, char **valuep);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>getsubopt() parses suboptions in a flag argument that was initially parsed by getopt(3C). These 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. A command that uses this syntax is mount(1M), which allows the user to specify mount parameters with the -o option as follows:</p> <pre style="text-align: center;">mount -o rw,hard,bg,wsize=1024 speed:/usr /usr</pre> <p>In this example there are four suboptions: rw, hard, bg, and wsize, the last of which has an associated value of 1024.</p> <p>getsubopt() 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 at <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, if <i>valuep</i> is not NULL, then the suboption processed included a value. The calling program may use this information to determine if the presence or lack of a value for this suboption is an error.</p> <p>Additionally, 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	<p>getsubopt() 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.</p> <p>The variable addressed by <i>optionp</i> points to the next option to be parsed, or a null character if there are no more options.</p>
EXAMPLE	The following code fragment shows how to process options to the mount(1M) command 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;
    }
}
break;
}
}
if (errflag) {
    /* print usage instructions etc. */
}
for (; optind < argc; optind++) {
    /* process remaining arguments */
}
.
.
.
}

```

SEE ALSO mount(1M), getopt(3C)

NOTES During parsing, commas in the option input string are changed to null characters. White space in tokens or token-value pairs must be protected from the shell by quotes.

NAME	gettext, dgettext, dcgettext, textdomain, bindtextdomain – message handling functions
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lintl [<i>library</i> ...] #include <libintl.h> #include <locale.h> /* needed for dcgettext() only */ char *gettext(const char *msgid); char *dgettext(const char *domainname, const char *msgid); char *dcgettext(const char *domainname, const char *msgid, int category); char *textdomain(const char *domainname); char *bindtextdomain(const char *domainname, const char *dirname);</pre>
MT-LEVEL	Safe with exceptions
DESCRIPTION	<p>gettext(), dgettext(), and dcgettext() 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 setting of the LC_MESSAGES category of the current locale determines the locale used by gettext() and dgettext() for string retrieval. <i>category</i> determines the locale used by dcgettext(). However, if the current locale is C, the routines simply return the message string that was passed.</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 messages) 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() call.</p> <p>textdomain() sets or queries the name of the current domain of the active LC_MESSAGES locale category. The <i>domainname</i> argument is a null-terminated string that can contain only the characters allowed in legal filenames.</p> <p>The <i>domainname</i> 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().</p> <p>The <i>domainname</i> argument is applied to the currently active LC_MESSAGES locale.</p> <p>The current setting of the domain can be queried without affecting the current state of the domain by calling textdomain() with <i>domainname</i> set to the null pointer. Calling textdomain() with a <i>domainname</i> argument of a null string sets the domain to the default domain (messages).</p>

bindtextdomain() 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*. *dirname* 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 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.

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_XXXdomainname.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.

SEE ALSO **msgfmt(1), xgettext(1), setlocale(3C)**

NOTES These routines impose no limit on message length. However, a text *domainname* is limited to **TEXTDOMAINMAX** (256) bytes.
gettext, dgettext, dcgettext, textdomain and **bindtextdomain** can be used safely in a multi-thread application, as long as **setlocale(3C)** is not being called to change the locale.

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> <p><i>tp</i> points to a timeval 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><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> <p>Only the privileged user may set the time of day.</p>
RETURN VALUES	A -1 return value indicates an error occurred; in this case an error code is stored in the global variable errno .
ERRORS	<p>The following error codes may be set in errno:</p> <p>EINVAL <i>tp</i> specifies an invalid time.</p> <p>EPERM A user other than the privileged user attempted to set the time.</p>
SEE ALSO	adjtime(2) , ctime(3C) , gettimeofday(3C) , TIMEZONE(4)
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><i>tzp</i> is ignored in SunOS 5.X releases.</p> <p>tv_usec is always 0.</p>

NAME	gettimeofday, settimeofday – get or set the date and time
SYNOPSIS	<pre>#include <sys/time.h> int gettimeofday(struct timeval *tp); int settimeofday(struct timeval *tp);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>gettimeofday() gets and settimeofday() 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><i>tp</i> points to a timeval 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. The TZ environment variable holds time zone information. See TIMEZONE(4). Only the privileged user may set the time of day.</p>
SEE ALSO	adjtime(2) , ctime(3C) , TIMEZONE(4)
RETURN VALUES	A -1 return value indicates that an error occurred and errno has been set.
ERRORS	<p>The following error codes may be set in errno:</p> <p>EINVAL <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>
NOTES	The implementation of settimeofday() ignores the tv_usec field of tp . If the time needs to be set with better than one second accuracy, call settimeofday() for the seconds and then adjtime() for finer accuracy.

NAME	gettext – retrieve a text string
SYNOPSIS	<pre>#include <nl_types.h> char *gettext(const char *msgid, const char *dflt_str);</pre>
MT-LEVEL	Safe with exceptions
DESCRIPTION	<p>gettext() 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. The text strings are in files created by the mkmsgs utility (see mkmsgs(1)) and installed in directories in /usr/lib/locale/locale/LC_MESSAGES.</p> <p>The directory <i>locale</i> 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 LC_MESSAGES. If LC_MESSAGES is not set, the environment variable LANG will be used. If LANG is not set, the files containing the strings are in /usr/lib/locale/C/LC_MESSAGES/*.</p> <p>The user can also change the language in which the messages are displayed by invoking the setlocale() 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 \0 (null) and the ASCII code for / (slash) and : (colon). The names of message files must be the same as the names of files created by mkmsgs and installed in /usr/lib/locale/locale/LC_MESSAGES/*. 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> <p>On failure to pass the correct msgid or a valid message number to gettext() a pointer to the text string "Message not found!!\n" is returned.</p>
EXAMPLES	<pre>gettext("UX:10", "hello world\n") gettext("UX:10", "")</pre> <p>UX is the name of the file that contains the messages. 10 is the message number.</p>

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
SEE ALSO	exstr(1), mkmsgs(1), srchtxt(1), gettext(3I), fmtmsg(3C), setlocale(3C), environ(5)	
NOTES	It is recommended that gettext(3I) be used in place of this routine.	

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>
MT-LEVEL	Unsafe
DESCRIPTION	<p>getutent(), getutid(), getutline(), and pututline() 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, lxxx) */ 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 exit status includes the following members:</p> <pre> short e_termination; /* termination status */ short e_exit; /* exit status */</pre> <p>getutent() 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() searches forward from the current point in the utmp file until it finds an entry with a ut_type matching <i>id</i>→ut_type if the type specified is RUN_LVL, BOOT_TIME, OLD_TIME, or NEW_TIME. If the type specified in <i>id</i> is INIT_PROCESS, LOGIN_PROCESS, USER_PROCESS, or DEAD_PROCESS, then getutid() will return a pointer to the first entry whose type is one of these four and whose ut_id field matches <i>id</i>→ut_id. If the end of file is reached without a match, it fails.</p> <p>getutline() searches forward from the current point in the utmp file until it finds an entry of the type LOGIN_PROCESS or ut_line string matching the <i>line</i>→ut_line string. If the end of file is reached without a match, it fails.</p> <p>pututline() writes out 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 routines. If so, pututline() will not search. If pututline() does not</p>

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* field must correspond to the actual user name associated with the process; the *ut_type* field must be either **USER_PROCESS** or **DEAD_PROCESS**; and the *ut_line* field must be a device special file and be writable by the user.

setutent() 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() closes the currently open file.

utmpname() 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. **utmpname()** does not open the file. It just 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.

FILES

/var/adm/utmp
/var/adm/wtmp

SEE ALSO

getutxent(3C), **ttyslot(3C)**, **utmp(4)**

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 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 **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()** routines, if the user has just modified those contents and passed the pointer back to **pututline()**.

These routines 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.

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 *id); struct utmpx *getutxline(const struct utmpx *line); struct utmpx *pututxline(const struct utmpx *utmpx); void setutxent(void); void endutxent(void); int utmpxname(const char *file); void getutmp(struct utmpx *utmpx, struct utmp *utmp); void getutmpx(struct utmp *utmp, struct utmpx *utmpx); void updwtmp(char *wfile, struct utmp *utmp); void updwtmpx(char *wfilex, struct utmpx *utmpx);</pre>
MT-LEVEL	Unsafe
DESCRIPTION	<p>getutxent(), getutxid(), and getutxline() 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]; /* /sbin/inittab id */ /* (usually line #) */ char ut_line[32]; /* device name (console, lnx) */ 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 */ 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> <p>getutxent() 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.</p> <p>getutxid() searches forward from the current point in the utmpx file until it finds an entry with a ut_type matching <i>id</i>→ut_type if the type specified is RUN_LVL, BOOT_TIME, OLD_TIME, or NEW_TIME. If the type specified in <i>id</i> is INIT_PROCESS, LOGIN_PROCESS, USER_PROCESS, or DEAD_PROCESS, then getutxid() will return a pointer to the first entry whose type is one of these four and whose <i>ut_id</i> field matches <i>id</i>→ut_id. If the end of file is reached without a match, it fails.</p>

getutxline() searches forward from the current point in the **utmpx** file until it finds an entry of the type **LOGIN_PROCESS** or **USER_PROCESS** which also has a *ut_line* string matching the *line*→**ut_line** string. If the end of file is reached without a match, it fails.

pututxline() writes out the supplied **utmpx** structure into the **utmpx** 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 **utmpx** structure. When called by a non-root user, **pututxline()** invokes a **setuid()** root program to verify and write the entry, since */etc/utmpx* is normally writable only by root. In this event, the *ut_name* field must correspond to the actual user name associated with the process; the *ut_type* field must be either **USER_PROCESS** or **DEAD_PROCESS**; and the *ut_line* field must be a device special file and be writable by the user.

setutxent() 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() closes the currently open file.

utmpxname() allows the user to change the name of the file examined, from */var/adm/utmpx* to any other file. It is most often expected that this other file will be */var/adm/wtmpx*. If the file does not exist, this will not be apparent until the first attempt to reference the file is made. **utmpxname()** does not open the file. It just closes the old file if it is currently open and saves the new file name. The new file name must end with the “x” character to allow the name of the corresponding **utmp** file to be easily obtainable; otherwise, an error code of **1** is returned.

getutmp() copies the information stored in the fields of the **utmpx** structure to the corresponding fields of the **utmp** structure. If the information in any field of **utmpx** does not fit in the corresponding **utmp** field, the data is truncated.

getutmpx() copies the information stored in the fields of the **utmp** structure to the corresponding fields of the **utmpx** structure.

updwtmp() checks the existence of *wfile* and its parallel file, whose name is obtained by appending an “x” 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() checks the existence of *wfilex* and its parallel file, whose name is obtained by truncating the final “x” 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 **tmp** 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.

FILES /var/adm/utmp
/var/adm/utmpx
/var/adm/wtmp
/var/adm/wtmpx

SEE ALSO **getutent(3C)**, **ttyslot(3C)**, **utmp(4)**, **utmpx(4)**

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()**.

These routines 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.

NAME	getvfsent, getvfsfile, getvfsspec, getvfsany – get vfstab file entry						
SYNOPSIS	<pre>#include <stdio.h> #include <sys/vfstab.h> int getvfsent(FILE *fp, struct vfstab *vp); int getvfsfile(FILE *fp, struct vfstab *vp, char *file); int getvfsspec(FILE *, struct vfstab *vp, char *spec); int getvfsany(FILE *, struct vfstab *vp, vfstab *vref);</pre>						
MT-LEVEL	Safe						
DESCRIPTION	<p>getvfsent(), getvfsfile(), getvfsspec(), and getvfsany() 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 vfstab structure, declared in the <code><sys/vfstab.h></code> header:</p> <pre>char *vfs_special; char *vfs_fsckdev; char *vfs_mountp; char *vfs_fstype; char *vfs_fsckpass; char *vfs_automnt; char *vfs_mntopts;</pre> <p>The fields have meanings described in vfstab(4).</p> <p>getvfsent() returns a pointer to the next vfstab structure in the file; so successive calls can be used to search the entire file. getvfsfile() 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. getvfsspec() 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. <i>spec</i> 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. getvfsany() searches the file referenced by <i>fp</i> until a match is found between a line in the file and <i>vref</i>. <i>vref</i> matches the line if all non-null entries in <i>vref</i> match the corresponding fields in the file.</p> <p>Note that these routines do not open, close, or rewind the file.</p>						
RETURN VALUES	<p>If the next entry is successfully read by getvfsent() or a match is found with getvfsfile(), 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:</p> <table border="0"> <tr> <td style="padding-right: 20px;">VFS_TOOLONG</td> <td>A line in the file exceeded the internal buffer size of VFS_LINE_MAX.</td> </tr> <tr> <td>VFS_TOOMANY</td> <td>A line in the file contains too many fields.</td> </tr> <tr> <td>VFS_TOOFEW</td> <td>A line in the file contains too few fields.</td> </tr> </table>	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.
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.						

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.

FILES

/etc/vfstab

SEE ALSO

vfstab(4)

NAME	getwc, getwchar, fgetwc – convert EUC character from the stream to Process Code
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lw [<i>library</i> ...] #include <stdio.h> #include <wchar.h> int getwc(FILE *<i>stream</i>); int getwchar(void); int fgetwc (FILE *<i>stream</i>);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>getwc() and fgetwc() convert the next Extended Unix Code (EUC) character from the named input <i>stream</i> into a <i>wchar_t</i> Process Code character, and return it as an integer. They also move the file pointer, if defined, ahead one EUC character in the <i>stream</i>. getwchar() is defined as fgetwc(stdin). getwc() and getwchar() are macros.</p>
RETURN VALUES	These functions return the integer constant EOF at end-of-file or upon an error. The end-of-file condition is remembered, even on a terminal, and all subsequent attempts to read will return EOF until the condition is cleared with clearerr() (see error(3S)).
SEE ALSO	error(3S) , fopen(3S) , fread(3S) , getws(3I) , putwc(3I) , scanf(3S) , ungetwc(3I)
WARNING	If the integer value returned by getwc() , getwchar() , or fgetwc() is stored into a <i>wchar_t</i> variable and then compared against the integer constant EOF, the comparison will not succeed, because <i>wchar_t</i> is defined as unsigned.

NAME	getwd – get current working directory pathname
SYNOPSIS	<pre>/usr/ucb/cc [<i>flag</i> ...] <i>file</i> ... #include <sys/param.h> char *getwd(<i>pathname</i>) char *pathname;</pre>
DESCRIPTION	getwd() copies the absolute pathname of the current working directory to <i>pathname</i> and returns a pointer to the result.
RETURN VALUES	getwd() returns zero and places a message in <i>pathname</i> if an error occurs. <i>pathname</i> points to a character array of at least MAXPATHLEN length.
SEE ALSO	getcwd(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	getwidth – get codeset information
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lw [<i>library</i> ...] #include <euc.h> #include <getwidth.h> void getwidth(eucwidth_t *ptr);</pre>
MT-LEVEL	MT-Safe with exceptions
DESCRIPTION	<p>The getwidth() function reads the character class table for the current locale, generated by wchrtbl() (see chrtbl(1M)) to get information on the supplementary codesets. getwidth() sets this information into the struct eucwidth_t. This struct is defined in <euc.h> 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 _eucw1, _eucw2, and _eucw3, respectively. Screen width values for supplementary codesets 1, 2, and 3 are set in _scrw1, _scrw2, and _scrw3, respectively.</p> <p>The width of Extended Unix Code (EUC) Process Code is set in _pcw. The _multibyte entry is set to 1 if multibyte characters are used, and set to 0 if only single-byte characters are used.</p>
SEE ALSO	chrtbl(1M) , euclen(3I) , setlocale(3C)
NOTES	getwidth can be used safely in a multi-thread application, as long as setlocale(3C) is not being called to change the locale.

NAME	getws, fgetws – convert a string of EUC characters from the stream to Process Code
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lw [<i>library</i> ...] #include <stdio.h> #include <wdec.h> wchar_t *getws(wchar_t *s); wchar_t *fgetws(wchar_t *s, int n, FILE *stream);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>getws() reads a string of Extended Unix Code (EUC) characters from the standard input stream, stdin, converts it to process code, and writes it to the array pointed to by <i>s</i>, until a new-line character is read or an end-of-file condition is encountered. The new-line character is discarded and the string is terminated with a <i>wchar_t</i> NULL character. getws() returns its argument.</p> <p>fgetws() reads EUC characters from the <i>stream</i>, converts them to Process Code, and writes them to the array pointed to by <i>s</i>. It stops when either <i>n</i>–1 characters are read, a new-line character is read and transferred to <i>s</i>, or an end-of-file condition is encountered. The string is then terminated with a <i>wchar_t</i> NULL character. fgetws() returns its first argument.</p>
RETURN VALUES	If end-of-file is encountered and no characters have been read, no characters are transferred to <i>s</i> 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 <i>s</i> is returned.
SEE ALSO	fferror(3S), fread(3S), getwc(3I), putws(3I), scanf(3S)

NAME	<code>gmatch</code> – shell global pattern matching
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lgen [<i>library</i> ...] #include <libgen.h> int gmatch(const char *<i>str</i>, const char *<i>pattern</i>);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<code>gmatch()</code> checks whether the null-terminated string <i>str</i> matches the null-terminated pattern string <i>pattern</i> . See the <code>sh(1)</code> section File Name Generation for a discussion of pattern matching. A backslash (\) is used as an escape character in pattern strings.
RETURN VALUES	<code>gmatch()</code> returns non-zero if the pattern matches the string, zero if the pattern does not.
EXAMPLE	In the following example, <code>gmatch()</code> returns non-zero (true) for all strings with “a” or “-” as their last character. <pre>char *s; gmatch (s, "[a\-")</pre>
SEE ALSO	<code>sh(1)</code>
NOTES	When compiling multi-thread applications, the <code>_REENTRANT</code> flag must be defined on the compile line. This flag should only be used in multi-thread applications.

NAME	grantpt – grant access to the slave pseudo-terminal device
SYNOPSIS	int grantpt(int <i>fildev</i>);
MT-LEVEL	Safe
DESCRIPTION	The function grantpt() 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 effective owner 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, writeable by the owner and writeable by the group.
RETURN VALUES	Upon successful completion, the function grantpt() returns 0 ; otherwise it returns -1 . Failure could occur if <i>fildev</i> is not an open file descriptor, if <i>fildev</i> is not associated with a master pseudo-terminal device, or if the corresponding slave device could not be accessed. grantpt() will also fail if it is unable to successfully invoke the <i>setuid</i> root program.
SEE ALSO	open(2), setuid(2), ptsname(3C), unlockpt(3C) <i>STREAMS Programmer's Guide</i>

NAME	hsearch , hcreate , hdestroy – manage hash search tables
SYNOPSIS	<pre>#include <search.h> ENTRY *hsearch(ENTRY item, ACTION action); int hcreate (size_t melements); void hdestroy(void);</pre>
MT-LEVEL	Safe
DESCRIPTION	<p>hsearch() 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)). <i>item</i> is a structure of type ENTRY (defined in the <search.h> 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.) <i>action</i> is a member of an enumeration type ACTION (defined in <search.h>) 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>hcreate() allocates sufficient space for the table, and must be called before hsearch() is used. <i>nel</i> 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>hdestroy() destroys the search table, and may be followed by another call to hcreate().</p>
RETURN VALUES	<p>hsearch() 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>hcreate() returns zero if it cannot allocate sufficient space for the table.</p>
EXAMPLE	<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 out.</p> <pre>#include <stdio.h> #include <search.h> #include <string.h> #include <stdlib.h> struct info { int age, room; }; /* this is the info stored in table */ /* other than the key */</pre>

```

#define NUM_EMPL    5000    /* # of elements in search table */

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;

```

```
}
```

SEE ALSO**bsearch(3C), lsearch(3C), malloc(3C), string(3C), tsearch(3C), malloc(3X)**

The Art of Computer Programming, Volume 3, Sorting and Searching by Donald E. Knuth, published by Addison-Wesley Publishing Company, 1973.

NOTES**hsearch()** and **hcreate()** use **malloc(3C)** to allocate space.

Only one hash search table may be active at any given time.

NAME	hyperbolic, sinh, cosh, tanh, asinh, acosh, atanh – hyperbolic functions
SYNOPSIS	<pre>#include <math.h> double sinh(double x); double cosh(double x); double tanh(double x); double asinh(double x); double acosh(double x); double atanh(double x);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	These functions compute the designated direct and inverse hyperbolic functions for real arguments.
RETURN VALUES	For exceptional cases, matherr (3M) tabulates the values to be returned as dictated by various Standards.
SEE ALSO	matherr (3M)
DIAGNOSTICS	In IEEE 754 mode (i.e. the -xlibmieee cc compilation option), sinh() and cosh() return $\pm\infty$ as appropriate on overflow and raise the overflow exception; acosh() returns a NaN and raises the invalid operation exception if its argument is less than 1; atanh() returns a NaN and raises the invalid operation exception if its argument has absolute value greater than 1; atanh (± 1) returns $\pm\infty$ and raises the division by zero exception.

NAME	hypot – Euclidean distance
SYNOPSIS	<pre>cc [flag ...] file ... -lm [library ...] #include <math.h> double hypot(double x, double y);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>hypot(x,y) returns $\sqrt{x*x + y*y}$, taking precautions against unwarranted IEEE exceptions. On IEEE overflow, hypot(x,y) may also set errno and call matherr(3M). hypot($\pm\infty$,y) is $+\infty$ for any y, even a <i>NaN</i>, and is exceptional only for a signaling <i>NaN</i>.</p> <p>hypot(x,y) and atan2(y,x) (see trig(3M)) convert rectangular coordinates (x,y) to polar (r,θ); hypot(x,y) computes r, the modulus or radius.</p>
SEE ALSO	matherr(3M) , trig(3M)

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>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>The iconv() function converts the sequence of characters from one codeset, in the array specified by <i>inbuf</i>, into a sequence of corresponding characters in another codeset, in the array specified by <i>outbuf</i>. The codesets 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 errno to E2BIG. 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 codeset, 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 codeset, 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 codeset.

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.

SEE ALSO

iconv_open(3), **iconv_close(3)**

NAME	iconv_close – code conversion deallocation function
SYNOPSIS	<pre>#include <iconv.h> int iconv_close(iconv_t cd);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>The iconv_close() function deallocates the conversion descriptor <i>cd</i> and all other associated resources allocated by the iconv_open(3) function.</p> <p>If a file descriptor is used to implement the type iconv_t, that file descriptor will be closed.</p>
RETURN VALUES	Upon successful completion, iconv_close() returns 0 ; otherwise, it returns -1 and sets errno to indicate the error.
ERRORS	The iconv_close() function may fail if: EBADF The conversion descriptor is invalid.
SEE ALSO	iconv(3) , iconv_open(3)

NAME	iconv_open – code conversion allocation function								
SYNOPSIS	<pre>#include <iconv.h> iconv_t iconv_open(const char *tocode, const char *fromcode);</pre>								
MT-LEVEL	MT-Safe								
DESCRIPTION	<p>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.</p> <p>Settings of <i>fromcode</i> and <i>tocode</i> and their permitted combinations are implementation-dependent.</p> <p>A conversion descriptor remains valid in a process until that process closes it.</p>								
RETURN VALUES	Upon successful completion iconv_open() returns a conversion descriptor for use on subsequent calls to iconv() . Otherwise, iconv_open() returns (iconv_t) -1 and sets errno to indicate the error.								
ERRORS	<p>The iconv_open function may fail if:</p> <table><tr><td>EMFILE</td><td>{OPEN_MAX} files descriptors are currently open in the calling process.</td></tr><tr><td>ENFILE</td><td>Too many files are currently open in the system.</td></tr><tr><td>ENOMEM</td><td>Insufficient storage space is available.</td></tr><tr><td>EINVAL</td><td>The conversion specified by <i>fromcode</i> and <i>tocode</i> is not supported by the implementation.</td></tr></table>	EMFILE	{ OPEN_MAX } 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.
EMFILE	{ OPEN_MAX } 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.								
SEE ALSO	iconv(3) , iconv_close(3) , malloc(3C)								
NOTES	<p>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.</p> <p>Portable applications must assume that conversion descriptors are not valid after a call to one of the exec functions.</p>								

NAME	ieee_functions, ilogb, isnan, copysign, fabs, fmod, nextafter, remainder, scalbn – appendix and related miscellaneous functions for IEEE arithmetic
SYNOPSIS	<pre>cc [flag ...] file ... -lm [library ...] #include <math.h> int ilogb(double x); int isnan(double x); double copysign(double x, double y); double fabs(double x); double fmod(double x, double y); double nextafter(double x, double y); double remainder(double x, double y); double scalbn(double x, int n);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>Most of these functions provide capabilities required by ANSI/IEEE Std 754-1985 or suggested in its appendix.</p> <p>ilogb(x) returns the unbiased exponent of x in integer format. ilogb($\pm\infty$) = +MAXINT and ilogb(0) = -MAXINT; <values.h> defines MAXINT as the largest int. ilogb(x) never generates an exception. When x is subnormal, ilogb(x) returns an exponent computed as if x were first normalized.</p> <p>isnan(x) returns 1 if x is NaN; otherwise it returns 0.</p> <p>copysign(x,y) returns a value with the magnitude of x and with the sign bit of y.</p> <p>fabs(x) returns the absolute value of x.</p> <p>nextafter(x,y) returns the next machine representable number from x in the direction y.</p> <p>remainder(x,y) and fmod(x,y) return a remainder of x with respect to y; that is, the result r is one of the numbers that differ from x by an integral multiple of y. Thus $(x-r)/y$ is an integral value, even though it might exceed MAXINT, the largest int defined in <values.h> if it were explicitly computed as an int. Both functions return one of the two such r smallest in magnitude. remainder(x,y) is the operation specified in ANSI/IEEE Std 754-1985; the result of fmod(x,y) may differ from remainder's result by $\pm y$. The magnitude of remainder's result cannot exceed half that of y; its sign might not agree with either x or y. The magnitude of fmod's result is less than that of y; its sign agrees with that of x. Neither function can generate an exception as long as both arguments are normal or subnormal.</p> <p>remainder(x,0), fmod(x,0), remainder(∞,y), and fmod(∞,y) are invalid operations; in IEEE 754 mode (i.e. the -xlibmieee cc compilation option), a NaN is returned.</p>

scalbn(x,n) returns $x*2**n$ computed by exponent manipulation rather than by actually performing an exponentiation or a multiplication. Thus

$$1 \leq \text{scalbn}(\text{fabs}(x), -\text{ilogb}(x)) < 2$$

for every x except 0, ∞ , and *NaN*.

RETURN VALUES

For exceptional cases, **matherr**(3M) tabulates the values to be returned as dictated by various Standards.

SEE ALSO

matherr(3M)

NAME	ieee_test, logb, scalb, significand – IEEE test functions for verifying standard compliance
SYNOPSIS	<pre>cc [flag ...] file ... -lm [library ...] #include <math.h> double logb(double x); double scalb(double x, double y); double significand(double x);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>These functions allow users to verify compliance to ANSI/IEEE Std 754-1985 by running certain test vectors distributed by the University of California.</p> <p>logb(x) returns the unbiased exponent of <i>x</i> in floating-point format, for exercising the logb(L) test vector. logb(±∞) = ±∞; logb(0) = −∞ with a division by 0 exception.</p> <p>scalb(x, (double)n) returns $x * 2^{**n}$ computed by exponent manipulation rather than by actually performing an exponentiation or a multiplication, for exercising the scalb(S) test vector. Thus</p> $0 \leq \text{scalb}(\text{fabs}(x), -\text{logb}(x)) < 2$ <p>for every <i>x</i> except 0, ∞ and NaN. scalb(x,y) is not defined when <i>y</i> is not an integral value. If <i>x</i> equals $\text{sig} * 2^{**n}$ with $1 \leq \text{sig} < 2$, then significand(x) returns <i>sig</i> for exercising the fraction-part(F) test vector. significand(x) is not defined when <i>x</i> is either 0, ±∞ or NaN.</p>
RETURN VALUES	For exceptional cases, matherr(3M) tabulates the values to be returned as dictated by various Standards.
SEE ALSO	matherr(3M)

NAME	index, rindex – string operations
SYNOPSIS	<pre>/usr/ucb/cc [<i>flag</i> ...] <i>file</i> ... #include <strings.h> char *index(<i>s</i>, <i>c</i>) register char *<i>s</i>; register char <i>c</i>; char *rindex(<i>s</i>, <i>c</i>) register char *<i>s</i>; register char <i>c</i>;</pre>
DESCRIPTION	<p>These functions operate on null-terminated strings.</p> <p>index() returns a pointer to the first occurrence of character <i>c</i> in string <i>s</i>, and rindex() returns a pointer to the last occurrence of character <i>c</i> in string <i>s</i>. 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>
SEE ALSO	bstring(3B) , malloc(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>On the Sun processor, as well as on many other machines, 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 have a pointer that points to an explicit null string. On some implementations of the C language on some machines, a null pointer, if dereferenced, would yield a null string; this highly non-portable trick was used in some programs. 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>

NAME	inet, inet_addr, inet_network, inet_makeaddr, inet_lnaof, inet_netof, inet_ntoa – Internet address manipulation
SYNOPSIS	<pre>cc [flag ...] file ... -lnsl [library ...] #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>
MT-LEVEL	Safe
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” described below.</p> <p>All Internet addresses are returned in network order (bytes ordered from left to right). All network numbers and local address parts are returned as machine format integer values.</p>
INTERNET ADDRESSES	<p>Values specified using the ‘.’ notation take one of the following forms:</p> <pre style="margin-left: 40px;">a.b.c.d a.b.c a.b a</pre> <p>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.</p> <p>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”.</p>

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.

All numbers supplied as “parts” in a ‘.’ notation may be decimal, octal, or hexadecimal, as specified in the C language (that is, 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 below.

SEE ALSO

gethostbyname(3N), **getnetbyname(3N)**, **hosts(4)**, **networks(4)**

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	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>
MT-LEVEL	Unsafe
DESCRIPTION	<p>initgroups() reads the group database, using getgrent(), to get the group membership for the user specified by <i>name</i> and then initializes the supplementary group access list of the calling process using setgroups(). (See getgrnam(3C)). 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, a value of 0 is returned. Otherwise, a value of -1 is returned and errno is set to indicate the error.
ERRORS	initgroups() will fail and not change the supplementary group access list if: EPERM The effective user id is not superuser.
SEE ALSO	getgroups(2) , getgrnam(3C)

NAME	insque, remque – insert/remove element from a queue
SYNOPSIS	<pre>include <search.h> void insque(struct qelem *elem, struct qelem *pred); void remque(struct qelem *elem);</pre>
MT-LEVEL	Unsafe
DESCRIPTION	<p>insque() and remque() manipulate queues built from doubly linked lists. Each element in the queue must be in the following form:</p> <pre>struct qelem { struct qelem *q_forw; struct qelem *q_back; char q_data[]; };</pre> <p>insque() inserts <i>elem</i> in a queue immediately after <i>pred</i>. remque() removes an entry <i>elem</i> from a queue.</p>

NAME	isastream – test a file descriptor
SYNOPSIS	<pre>#include <stropts.h> int isastream(int <i>fildev</i>);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	The function isastream() determines if a file descriptor represents a STREAMS file. <i>fildev</i> refers to an open file descriptor.
RETURN VALUES	If successful, isastream() returns 1 if <i>fildev</i> represents a STREAMS file, and 0 if not. On failure, isastream() returns -1 with errno set to indicate an error.
ERRORS	Under the following conditions, isastream() fails and sets errno to: EBADF <i>fildev</i> is not a valid file descriptor.
SEE ALSO	streamio(7) <i>STREAMS Programmer's Guide</i>

NAME	isencrypt – determine whether a buffer of characters is encrypted
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lgen [<i>library</i> ...] #include <libgen.h> int isencrypt(const char *fbuf, size_t ninbuf);</pre>
MT-LEVEL	MT-Safe
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, isencrypt() assumes that the buffer is encrypted if the setlocale() LC_CTYPE category is set to C or ascii.</p> <p>If the LC_CTYPE category is set to a value other than C or ascii, 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; and isencrypt() assumes the buffer is encrypted if it does. 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.
SEE ALSO	setlocale(3C)
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	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>																				
MT-LEVEL	MT-Safe																				
DESCRIPTION	<p>The functionality of isnan() is identical to that of isnand(). isnanf() is implemented as a macro included in the <ieeefp.h> header. fpclass() returns the class the <i>dsrc</i> belongs to. The 10 possible classes are as follows:</p> <table border="0"> <tr> <td style="padding-right: 1em;">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 generate any 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	<p>isnan(), isnand(), and isnanf() return true (1) if the argument <i>dsrc</i> or <i>fsrc</i> is a NaN; otherwise they return false (0).</p> <p>finite() returns true (1) if the argument <i>dsrc</i> is neither infinity nor NaN; otherwise it returns false (0).</p> <p>unordered() 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 of the arguments is NaN, false (0) is returned.</p>																				
SEE ALSO	fpgetround(3C)																				

NAME	iswalpha, iswupper, iswlower, iswdigit, iswxdigit, iswalnum, iswspace, iswpunct, iswprint, iswcntrl, iswascii, iswgraph, isphonogram, isideogram, isenglish, isnumber, isspecial – Process Code character classification macros and functions
SYNOPSIS	<pre>cc [flag ...] file ... -lw [library ...] #include <wdec.h> #include <wctype.h> int iswalpha(int c);</pre>
MT-LEVEL	MT-Safe with exceptions
WIDE CHARACTER CLASSIFICATION	<p>These functions classify Process Code characters (<i>wchar_t</i>) from the primary and supplementary codesets by table lookup. Each is a predicate returning nonzero for true, zero for false. The lookup table, generated by wchrtbl(), contains values for both ASCII and supplementary codesets.</p> <p>iswalpha(c) <i>c</i> is a Latin alphabet Process Code character, from either the primary or supplementary codesets.</p> <p>iswupper(c) <i>c</i> is an upper case Latin alphabet Process Code character, from either the primary or supplementary codesets.</p> <p>iswlower(c) <i>c</i> is a lower case Latin alphabet Process Code character, from either the primary or supplementary codesets.</p> <p>iswdigit(c) <i>c</i> is a Process Code digit [0-9], from either the primary or supplementary codesets.</p> <p>iswxdigit(c) <i>c</i> is an ASCII hexadecimal Process Code digit [0-9], [A-F], or [a-f].</p> <p>iswalnum(c) <i>c</i> is a Process Code Latin letter or a digit, from either the primary or supplementary codesets.</p> <p>iswspace(c) <i>c</i> is a Process Code space, tab, carriage return, newline, vertical tab, or formfeed, from either the primary or supplementary codesets.</p> <p>iswpunct(c) <i>c</i> is an ASCII Process Code punctuation character (neither control nor alphanumeric).</p> <p>iswprint(c) <i>c</i> is a Process Code printing character, from either the primary or supplementary codesets. It includes the <i>space</i> character.</p> <p>iswgraph(c) <i>c</i> is a Process Code visible graphic character, from the primary or supplementary codesets. It does not include the <i>space</i> character.</p> <p>iswcntrl(c) <i>c</i> is a Process Code ASCII delete character or ordinary control character, or a control character from a supplementary codeset.</p> <p>iswascii(c) <i>c</i> is a Process Code ASCII character.</p> <p>isphonogram(c) <i>c</i> is a Process Code phonetic language character from a supplementary codeset.</p>

isideogram(c)	<i>c</i> is a Process Code ideographic language character from a supplementary codeset.
isenglish(c)	<i>c</i> is a Process Code English language character from a supplementary codeset.
isnumber(c)	<i>c</i> is a Process Code digit [0-9] from a supplementary codeset.
isspecial(c)	<i>c</i> is a Process Code special language character from a supplementary codeset.

SEE ALSO [setlocale\(3C\)](#), [stdio\(3S\)](#), [wconv\(3I\)](#), [ascii\(5\)](#)

NOTES [iswalpha\(\)](#), [iswupper\(\)](#), [iswlower\(\)](#), [iswdigit\(\)](#), [iswxdigit\(\)](#), [iswalnum\(\)](#), [iswspace\(\)](#), [iswpunct\(\)](#), [iswprint\(\)](#), [iswcntrl\(\)](#), [iswascii\(\)](#), [iswgraph\(\)](#), [isphonogram\(\)](#), [isideogram\(\)](#), [isenglish\(\)](#), [isnumber\(\)](#) and [isspecial\(\)](#) can be used safely in a multi-thread application, as long as [setlocale\(3C\)](#) is not being called to change the locale.

NAME	iswctype – test character for specified class																						
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lw [<i>library</i> ...] #include <wchar.h> int iswctype(wint_t <i>wc</i>, wctype_t <i>charclass</i>);</pre>																						
MT-LEVEL	MT-Safe																						
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. iswctype() 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(3I) or <i>charclass</i> is invalidated by a subsequent call to setlocale(3C) that has affected category LC_CTYPE), the result is implementation-dependent.</p>																						
RETURN VALUE	iswctype() returns 0 for FALSE and non-zero for TRUE .																						
USAGE	<p>The twelve strings — "alnum", "alpha", "blank", "cntrl", "digit", "graph", "lower", "print", "punct", "space", "upper" and "xdigit" — are reserved for the standard character classes. In the table below, the functions in the left column are equivalent to the functions in the right column.</p> <table border="0"> <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> </table> <p>The call</p> <pre>iswctype(<i>wc</i>, wctype("blank"))</pre> <p>does not have an equivalent isw() function.</p>	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"))
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"))																						
SEE ALSO	iswalnum(3I) , iswalpha(3I) , iswcntrl(3I) , iswdigit(3I) , iswgraph(3I) , iswlower(3I) , iswprint(3I) , iswpunct(3I) , iswspace(3I) , iswupper(3I) , iswxdigit(3I) , setlocale(3C) , wctype(3I)																						

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	<pre>cc [flag ...] file ... -lkrb [library ...] #include <kerberos/krb.h> extern char *krb_err_txt[]; int krb_mk_req(KTEXT <i>authent</i>, const char *<i>service</i>, const char *<i>instance</i>, const char *<i>realm</i>, const u_long <i>checksum</i>); int krb_rd_req(const KTEXT <i>authent</i>, const char *<i>service</i>, char *<i>instance</i>, const u_long <i>from_addr</i>, AUTH_DAT *<i>ad</i>, const char *<i>fn</i>); int krb_kntoln(const AUTH_DAT *<i>ad</i>, char *<i>lname</i>); int krb_set_key(const char *<i>key</i>, const int <i>cvt</i>); int krb_get_cred(const char *<i>service</i>, const char *<i>instance</i>, const char *<i>realm</i>, CREDENTIALS *<i>c</i>); long krb_mk_safe(const u_char *<i>in</i>, u_char *<i>out</i>, const u_long <i>in_length</i>, const des_cblock <i>key</i>, const struct sockaddr_in <i>sender</i>, const struct sockaddr_in <i>receiver</i>); long krb_rd_safe(const u_char *<i>in</i>, const u_long <i>length</i>, const des_cblock <i>key</i>, const struct sockaddr_in <i>sender</i>, const struct sockaddr_in <i>receiver</i>, MSG_DAT *<i>msg_data</i>); long krb_mk_err(u_char *<i>out</i>, const long <i>code</i>, const char *<i>string</i>); long krb_rd_err(const u_char *<i>in</i>, const u_long <i>length</i>, long *<i>code</i>, MSG_DAT <i>msg_data</i>);</pre>
MT-LEVEL	Unsafe
DESCRIPTION	<p>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.</p> <p>krb_err_txt[] contains text string descriptions of various Kerberos error codes returned by some of the routines below.</p> <p>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 <i>authent</i> and is accessible to the calling procedure.</p> <p>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.</p>

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/tktuid
SEE ALSO	kerberos(1), kerberos_rpc(3N), krb_realmofhost(3N), krb_sendauth(3N), krb_set_tkt_string(3N), krb.conf(4), krb.realms(4)
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
MT-LEVEL	Unsafe
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> <ul style="list-style-type: none"> AUTH_NONE (none) no authentication. AUTH_SYS Traditional UNIX-style authentication. AUTH_DES DES encryption-based authentication. AUTH_KERB Kerberos encryption-based authentication. <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	<p>See rpc(3N) for the definition of the AUTH data structure.</p> <pre>cc [flag ...] file ... -lkrb [library ...] #include <rpc/rpc.h> #include <sys/types.h> int authkerb_getucred(const struct svc_rqst *rqst, uid_t *uidp, gid_t *gidp, short *gidlenp, int gidlist[NGROUPS]);</pre> <p>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.</p> <p>*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.</p>

```
AUTH *authkerb_seccreate(const char *service, const char *srv_inst,  
const char *realm, const unsigned int 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 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.

SEE ALSO **kerberos(1)**, **kinit(1)**, **kerbd(1M)**, **rpc(3N)**, **rpc_clnt_auth(3N)**, **secure_rpc(3N)**

NOTES These interfaces are unsafe in multithreaded applications. Unsafe interfaces should be called only from the main thread.

NAME	killpg – send signal to a process group						
SYNOPSIS	<pre>/usr/ucb/cc [flag ...] file ... int killpg(<i>pgrp</i>, <i>sig</i>) int pgrp; int sig;</pre>						
DESCRIPTION	<p>killpg() sends the signal <i>sig</i> to the process group <i>pgrp</i>. See sigvec(3B) 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.</p>						
RETURN VALUES	Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and the global variable errno is set to indicate the error.						
ERRORS	<p>killpg() will fail and no signal will be sent if any of the following occur:</p> <table><tr><td>EINVAL</td><td><i>sig</i> is not a valid signal number.</td></tr><tr><td>EPERM</td><td>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.</td></tr><tr><td>ESRCH</td><td>No processes were found in the specified process group.</td></tr></table>	EINVAL	<i>sig</i> 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.
EINVAL	<i>sig</i> 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) , 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	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>
MT-LEVEL	Unsafe
DESCRIPTION	<p>krb_realmofhost() returns the Kerberos realm of the host <i>host</i>, as determined by the translation table <i>/etc/krb.realms</i>. <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 krb.realms(4). If <i>host</i> exactly matches a <i>host_name</i> line, the corresponding realm is returned. Otherwise, if the domain portion of <i>host</i> matches a <i>domain_name</i> 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 krb_get_lrealm(), 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 (rlogin, rcp, rsh). 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 configuration file (<i>/etc/krb.conf</i> 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 <i>n</i> equals 1, then the hostname kerberos is filled in, and KSUCCESS is returned. If there are fewer than <i>n</i> hosts running a Kerberos KDC for the requested realm, or the configuration file is malformed, the routine returns KFAILURE.</p> <p>When there is both a local <i>/etc/krb.conf</i> 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 <i>/etc/krb.conf</i> file contains two entries which match <i>realm</i>, and the NIS map contains one matching entry, then there are three possible matches that krb_get_krbhst() can return. The first two (for <i>n</i> values 1 and 2) come from the file, and the third (for <i>n</i></p>

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.

SEE ALSO **kerberos(3N)**, **krb.conf(4)**, **krb.realms(4)**

FILES `/etc/krb.realms` translation file for host-to-realm mapping.
`/etc/krb.conf` 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 u_long checksum, MSG_DATA *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_version 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>
MT-LEVEL	Unsafe
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> <p>The <i>version</i> 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 <i>version</i> string can be up to KSEND_VNO_LEN (see <kerberos/krb.h>) characters in length.</p> <p>The <i>checksum</i> 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 <i>checksum</i> argument is passed as the checksum argument to krb_mk_req() (see kerberos(3N)).</p>

**krb_sendauth() and
Mutual
Authentication**

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.

krb_recvauth()

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.

	<p>The <i>filename</i> argument specifies the filename which the service program should use to obtain its service key. krb_recvauth() passes <i>filename</i> 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 <i>/etc/srvtab</i>.</p> <p>If the client and server are performing mutual authentication, the <i>schedule</i> argument should point to an allocated <i>Key_schedule</i>. Otherwise it is ignored and may be <i>NULL</i>.</p> <p>The <i>version</i> argument should point to a character array of at least <i>KSEND_VNO_LEN</i> characters. It is filled in with the version string passed by the client to krb_sendauth().</p>
krb_net_write() and krb_net_read()	<p>The krb_net_write() function emulates the write(2) system call, but guarantees that all data specified is written to <i>fd</i> before returning, unless an error condition occurs.</p> <p>The krb_net_read() function emulates the read(2) system call, but guarantees that the requested amount of data is read from <i>fd</i> before returning, unless an error condition occurs.</p>
SEE ALSO	read(2) , write(2) , kerberos(3N) , kerberos_rpc(3N) , krb_realmofhost(3N)
NOTES	These interfaces are unsafe in multithreaded applications. Unsafe interfaces should be called only from the main thread.
BUGS	krb_sendauth() , krb_recvauth() , krb_net_write() , and krb_net_read() 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 <kerberos/mit-copyright.h> .

NAME	krb_set_tkt_string – set Kerberos ticket cache file name		
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lkrb [<i>library</i> ...] #include <kerberos/krb.h> void krb_set_tkt_string(const char *filename);</pre>		
MT-LEVEL	Unsafe		
DESCRIPTION	<p>krb_set_tkt_string() sets the name of the file that holds the user's cache of Kerberos server tickets and associated session keys.</p> <p>The string <i>filename</i> 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.</p> <p>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.</p>		
FILES	<table><tr><td><i>/tmp/tktuid</i></td><td>default ticket file name, unless the environment variable KRBTKFILE is set. <i>uid</i> denotes the user's uid, in decimal.</td></tr></table>	<i>/tmp/tktuid</i>	default ticket file name, unless the environment variable KRBTKFILE is set. <i>uid</i> denotes the user's uid, in decimal.
<i>/tmp/tktuid</i>	default ticket file name, unless the environment variable KRBTKFILE is set. <i>uid</i> denotes the user's uid, in decimal.		
SEE ALSO	kerberos(3N)		
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 kstat 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 kstats. Each kstat has a common header section and a type-specific data section. The header section is defined by the kstat_t structure:

```

kstat header  typedef long      kid_t;                               /* unique kstat id */
               typedef struct kstat {
               /*
                * Fields relevant to both kernel and user
                */
               hrtime_t    ks_crtime;                            /* 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 */
               ulong_t     ks_ndata;                              /* # of data records */
               ulong_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;

```

The fields that are of significance to the user are:

ks_crtime The time the kstat was created. This allows you to compute the rates of various counters since the kstat was created; "rate since boot" is replaced by the more general concept of "rate since kstat creation".

All times associated with kstats (e.g. creation time, last snapshot time, **kstat_timer_t** and **kstat_io_t** timestamps, etc.) are 64-bit nanosecond values. The accuracy of kstat timestamps is machine dependent, but the precision (units) is the same across all platforms. See **gethrtime(3C)** for

general information about high-resolution timestamps.

ks_next kstats are stored as a linked list, or chain. **ks_next** points to the next kstat in the chain.

ks_kid A unique identifier for the kstat.

ks_module, ks_instance contain the name and instance of the the module that created the kstat. In cases where there can only be one instance, **ks_instance** is 0.

ks_name gives a meaningful name to a kstat. The full kstat namespace is **<ks_module, ks_instance, ks_name** so the name only need be unique within a module.

ks_type The type of data in this kstat. kstat data types are discussed below.

ks_class Each kstat can be characterized as belonging to some broad class of statistics, e.g. disk, tape, net, vm, streams, etc. This field can be used as a filter to extract related kstats. The following values are currently in use: **disk, tape, controller, net, rpc, vm, kvm, hat, kmem, kmem_cache, kstat,** and **misc.** (The kstat class encompasses things like *kstat_types*.)

ks_data, ks_ndata, ks_data_size

ks_data is a pointer to the kstat's data section. The type of data stored there depends on **ks_type**.

ks_ndata indicates the number of data records. Only some kstat types support multiple data records. Currently, **KSTAT_TYPE_RAW, KSTAT_TYPE_NAMED** and **KSTAT_TYPE_TIMER** kstats support multiple data records. **KSTAT_TYPE_INTR** and **KSTAT_TYPE_IO** kstats support only one data record.

ks_data_size is the total size of the data section, in bytes.

ks_snaptime The timestamp for the last data snapshot. This allows you to compute activity rates:

$$\text{rate} = (\text{new_count} - \text{old_count}) / (\text{new_snaptime} - \text{old_snaptime});$$
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 (e.g., **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 */
        long      l;
        ulong_t   ul;
        longlong_t ll;
        u_longlong_t ull;
        float     f;
        double    d;
    } value; /* value of counter */
} kstat_named_t;

#define KSTAT_DATA_CHAR      0
#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 {
    ulong_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 */
    hrtime_t     elapsed_time;        /* cumulative elapsed time */
    hrtime_t     min_time;            /* shortest event duration */
    hrtime_t     max_time;            /* longest event duration */
    hrtime_t     start_time;          /* previous event start time */
    hrtime_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 */
    ulong_t      reads;                /* number of read operations */
    ulong_t      writes;               /* number of write operations */
    /*
```

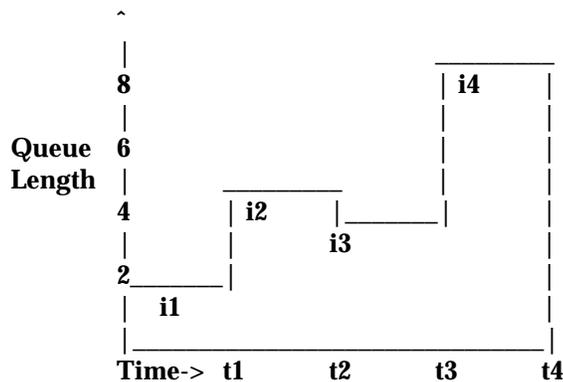
*** Accumulated time and queue length statistics.**

*** 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 --**

*** i.e., a Riemann sum for queue length integrated against time.**



*** 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    wlen*time;           /* cumulative wait length*time product */
hrtime_t    wlastupdate;        /* last time wait queue changed */
hrtime_t    rtime;              /* cumulative run (service) time */
hrtime_t    rlen*time;          /* cumulative run length*time product */
hrtime_t    rlastupdate;        /* last time run queue changed */
ulong_t     wcnt;               /* count of elements in wait state */
ulong_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);
    }
}

```

kc_chain_id is the kstat chain ID, or KCID, of your copy of the kstat chain. See **kstat_chain_update**(3K) for an explanation of KCIDs.

FILES /dev/kstat kernel statistics driver
 /usr/include/kstat.h
 /usr/include/sys/kstat.h

SEE ALSO **ioctl**(2), **gethrtime**(3C), **kstat_chain_update**(3K), **kstat_close**(3K),
kstat_data_lookup(3K), **kstat_lookup**(3K), **kstat_open**(3K), **kstat_read**(3K),
kstat_write(3K)

NAME	<code>kstat_chain_update</code> – update the kstat header chain
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lkstat [<i>library</i> ...] #include <kstat.h> kid_t *kstat_chain_update(kstat_ctl_t *kc);</pre>
DESCRIPTION	<p><code>kstat_chain_update()</code> brings the user's kstat header chain in sync with the kernel's. The kstat chain is a linked list of kstat headers (<code>kstat_t</code>'s), pointed to by <code>kc->kc_chain</code>, which is initialized by <code>kstat_open(3K)</code>. 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><code>kstat_chain_update()</code> 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, <code>kc->kc_chain_id</code>. If the KCID's match, <code>kstat_chain_update()</code> does nothing. Otherwise, it deletes any invalid kstat headers from the user's kstat chain, and adds any new ones, and sets <code>kc->kc_chain_id</code> to the new KCID. All other kstat headers in the user's kstat chain are unmodified.</p>
RETURN VALUES	<code>kstat_chain_update()</code> returns the new KCID if the kstat chain has changed, <code>0</code> if it hasn't, or <code>-1</code> on failure.
FILES	<code>/dev/kstat</code> kernel statistics driver
SEE ALSO	<code>kstat(3K)</code> , <code>kstat_close(3K)</code> , <code>kstat_data_lookup(3K)</code> , <code>kstat_lookup(3K)</code> , <code>kstat_open(3K)</code> , <code>kstat_read(3K)</code> , <code>kstat_write(3K)</code>

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 NULL, <i>ks_instance</i> is -1, or <i>ks_name</i> is NULL, then those fields will be ignored in the search. For example, kstat_lookup(NULL, -1, "foo") 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 KSTAT_TYPE_NAMED and KSTAT_TYPE_TIMER kstats have named data records.</p>
RETURN VALUES	<p>kstat_lookup() returns a pointer to the requested kstat if it is found, or NULL if it isn't.</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 NULL.</p>
FILES	<pre>/dev/kstat kernel statistics driver</pre>
SEE ALSO	<pre>kstat(3K), kstat_chain_update(3K), kstat_close(3K), kstat_open(3K), kstat_read(3K), kstat_write(3K)</pre>

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 libkstat function calls.</p> <p>kstat_close() frees all resources that were associated with <i>kc</i>. This is done automatically on exit(2) and execve() (see exec(2)).</p>
RETURN VALUES	<p>kstat_open() returns a pointer to a kstat control structure. On failure, it returns NULL and no resources are allocated.</p> <p>kstat_close() returns 0 on success, -1 on failure.</p>
FILES	<p>/dev/kstat kernel statistics driver</p>
SEE ALSO	<p>kstat(3K), kstat_chain_update(3K), kstat_data_lookup(3K), kstat_lookup(3K), kstat_read(3K), kstat_write(3K)</p>

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	<i>/dev/kstat</i> 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 *kd, struct proc *proc); int kvm_getcmd(kvm_t *kd, struct proc *proc, struct user *u, char ***arg, char ***env);</pre>				
MT-LEVEL	Unsafe				
DESCRIPTION	<p>kvm_getu() 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><i>kd</i> is a pointer to a kernel descriptor returned by kvm_open(3K). <i>proc</i> 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> <p>kvm_getcmd() 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><i>kd</i> is a pointer to a kernel descriptor returned by kvm_open(3K). <i>u</i> 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 NULL, then 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 NULL, 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() (see malloc(3C)) when no longer needed. 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 -1 if the user process data is unrecognizable.</p>				
RETURN VALUES	<p>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 NULL.</p> <p>kvm_getcmd() returns:</p> <table border="0"> <tr> <td style="padding-right: 10px;">0</td> <td>on success.</td> </tr> <tr> <td style="padding-right: 10px;">-1</td> <td>on failure.</td> </tr> </table>	0	on success.	-1	on failure.
0	on success.				
-1	on failure.				

SEE ALSO `kvm_nextproc(3K)`, `kvm_open(3K)`, `kvm_read(3K)`, `malloc(3C)`

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.

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_getproc(kvm_t *kd, int pid); struct proc *kvm_nextproc(kvm_t *kd); int kvm_setproc (kvm_t *kd);</pre>
MT-LEVEL	Unsafe
DESCRIPTION	<p>kvm_nextproc() 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 <i>all</i> processes will be reported.</p> <p>kvm_setproc() rewinds the process list, enabling kvm_nextproc() to rescan from the beginning of the system process table. kvm_setproc() will always flush the process structure cache, allowing an application to re-scan the process table of a running system.</p> <p>kvm_getproc() locates the proc structure of the process specified by <i>pid</i> and returns a pointer to it. kvm_getproc() 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 NULL.</p> <p>On success, kvm_getproc() returns a pointer to the proc structure of the process specified by <i>pid</i>. On failure, it returns NULL.</p> <p>kvm_setproc() returns:</p> <p>0 on success.</p> <p>-1 on failure.</p>
SEE ALSO	kvm_getu(3K) , kvm_open(3K) , kvm_read(3K)

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>
MT-LEVEL	Unsafe
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 nlist 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.
SEE ALSO	nlist(3B) , nlist(3E) , kvm_open(3K) , kvm_read(3K)

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>				
MT-LEVEL	Unsafe				
DESCRIPTION	<p>kvm_open() initializes a set of file descriptors to be used in subsequent calls to kernel 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 NULL, 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 <i>/dev/ksyms</i> as a default <i>namelist</i> file).</p> <p><i>corefile</i> specifies a file that contains an image of physical memory, for instance, a kernel crash dump file (see savecore(1M)) or the special device <i>/dev/mem</i>. If <i>corefile</i> is NULL, the currently running kernel is accessed (using <i>/dev/mem</i> and <i>/dev/kmem</i>).</p> <p><i>swapfile</i> specifies a file that represents the swap device. If both <i>corefile</i> and <i>swapfile</i> are NULL, the swap device of the “currently running kernel” is accessed. Otherwise, if <i>swapfile</i> is NULL, kvm_open() may succeed but subsequent kvm_getu(3K) function calls may fail if the desired information is swapped out.</p> <p><i>flag</i> 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: 40px;"> <tr> <td>O_RDONLY</td> <td>open for reading</td> </tr> <tr> <td>O_RDWR</td> <td>open for reading and writing</td> </tr> </table> <p><i>errstr</i> 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 kvm_open. Errors are printed to stderr. A useful value to supply for <i>errstr</i> would be argv[0]. This has the effect of printing the process name in front of any error messages.</p> <p>kvm_close() closes all file descriptors that were associated with <i>kd</i>. 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.</p>	O_RDONLY	open for reading	O_RDWR	open for reading and writing
O_RDONLY	open for reading				
O_RDWR	open for reading and writing				
RETURN VALUES	<p>kvm_open() returns a non-NULL value suitable for use with subsequent kernel VM function calls. On failure, it returns NULL and no files are opened.</p> <p>kvm_close() returns:</p> <table border="0" style="margin-left: 40px;"> <tr> <td>0</td> <td>on success.</td> </tr> </table>	0	on success.		
0	on success.				

-1 on failure.

FILES /kernel/unix
/dev/kmem
/dev/ksyms
/dev/mem

SEE ALSO savecore(1M), exec(2), exit(2), kvm_getu(3K), kvm_nextproc(3K), kvm_nlist(3K),
kvm_read(3K)

NOTES Programs using **libkvm** are likely to be platform and release dependent.
Kernel core dumps should be examined on the same platform they were created on.

NAME	kvm_read, kvm_write – copy data to or from a kernel image or running system
SYNOPSIS	<pre>#include <kvm.h> int kvm_read(kvm_t *kd, unsigned long addr, char *buf, unsigned nbytes); int kvm_write(kvm_t *kd, unsigned long addr, char *buf, unsigned nbytes);</pre>
MT-LEVEL	Unsafe
DESCRIPTION	<p>kvm_read() 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>.</p> <p>kvm_write() is like kvm_read(), except that the direction of data transfer is reversed. In order to use this function, the kvm_open(3K) call that returned <i>kd</i> 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.</p>
RETURN VALUES	On success, kvm_read() and kvm_write() return the number of bytes actually transferred. On failure, they return -1.
SEE ALSO	kvm_getu(3K) , kvm_nlist(3K) , kvm_open(3K)

NAME	lckpwnf , ulckpwnf – manipulate shadow password database lock file
SYNOPSIS	#include <shadow.h> int lckpwnf(void); int ulckpwnf(void);
MT-LEVEL	MT-Safe
DESCRIPTION	lckpwnf() and ulckpwnf() are routines that are used to gain 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 /etc/passwd or /etc/shadow password database. Upon completing modifications, a process should release the lock on the lock file using ulckpwnf() . This mechanism prevents simultaneous modification of the password databases. /etc.pwd.lock is the lock file. It is used to coordinate modification access to the password databases /etc/passwd and /etc/shadow . lckpwnf() attempts to lock the file /etc.pwd.lock within 15 seconds. If unsuccessful, for example, /etc.pwd.lock is already locked, it returns -1 . If successful, a return code other than -1 is returned. ulckpwnf() attempts to unlock the file /etc.pwd.lock . If unsuccessful, for example, /etc.pwd.lock is already unlocked, it returns -1 . If successful, it returns 0 .
RETURN VALUES	lckpwnf() and ulckpwnf() return -1 on failure, and 0 otherwise.
FILES	/etc/shadow /etc/passwd /etc.pwd.lock
SEE ALSO	getpwnam(3C) , getspnam(3C)
NOTES	These routines are for internal use only; compatibility is not guaranteed.

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>
MT-LEVEL	MT-safe
DESCRIPTION	<p>lfmt() retrieves a format string from a locale-specific message database (unless MM_NOGET is specified) and uses it for printf() style formatting of <i>args</i>. The output is displayed on <i>stream</i>. If <i>stream</i> is NULL, no output is displayed.</p> <p>lfmt() encapsulates the output in the standard error message format (unless MM_NOSTD is specified, in which case the output is simply printf() like).</p> <p>lfmt() forwards its output to the logging and monitoring facility, even if <i>stream</i> is NULL. Optionnally, lfmt() will display 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> <p style="padding-left: 40px;"><i><catalog>:<msgnum>:<defmsg></i>.</p> <p>If MM_NOGET is specified, only the <i><defmsg></i> part must be specified.</p> <p><i><catalog></i> is used to indicate the message database that contains the localized version of the format string. <i><catalog></i> 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><i><msgnum></i> 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() using the LC_ALL or LC_MESSAGES 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> part 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(). In this case, the <i>format</i> argument has the following structure:</p> <p style="padding-left: 40px;"><i>:<msgnum>:<defmsg></i>.</p> <p>lfmt() will output Message not found!!\n as format string if <i><catalog></i> is not a valid catalog name, if no catalog is specified (either explicitly or via 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> <p>The <i>flags</i> determine the type of output (i.e. whether the <i>format</i> should be interpreted as is or encapsulated in the standard message format), and the access to message catalogs to retrieve a localized version of <i>format</i>.</p>

The *flags* are 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 a **printf()** *format*. Only *catalog access control flags*, *console display control* and *logging information* 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>* part 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** generates a localized version of 4HALT, but does not halt the machine.
- MM_ERROR** generates a localized version of **ERROR** (default, value 0).
- MM_WARNING** generates a localized version of **WARNING**.
- MM_INFO** generates 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()**. The numeric value ORed with other *flags* will generate the specified severity.

If the severity is not defined, **lfmt()** used 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** specifies 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 0).

*Logging information**Major classification*

Identifies the source of the condition. Identifiers are: **MM_HARD** (hardware), **MM_SOFT** (software), and **MM_FIRM** (firmware).

Message source subclassification

Identifies the type of software in which the problem is spotted. Identifiers are: **MM_APPL** (application), **MM_UTIL** (utility), and **MM_OPSYS** (operating system).

**STANDARD
ERROR MESSAGE
FORMAT**

lfmt() displays error messages in the following format:

label: severity: text

If no *label* was defined by a call to **setlabel()**, 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 can look like:

label: severity: text

label: TO FIX: text

RETURN VALUE

Upon success, **lfmt()** returns the number of bytes transmitted. Upon failure, it returns a negative value:

-1 write error to *stream*.

-2 cannot log and/or display at console.

EXAMPLES

Example 1:

```
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:

```
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

NOTES

Since **lfmt()** uses **gettext(3C)**, it is recommended that **lfmt()** not be used.

SEE ALSO

addsev(3C), **gettext(3C)**, **pfmt(3C)**, **printf(3S)**, **setcat(3C)**, **setlabel(3C)**, **setlocale(3C)**, **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>
MT-LEVEL	See the NOTES section of this page.
DESCRIPTION	<p>Both lgamma() and lgamma_r() return $\ln \Gamma(x)$ 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 signgam to return the sign of $\Gamma(x)$ while lgamma_r() uses the user-allocated space addressed by <i>signgamp</i>.</p>
IDIOSYNCRASIES	<p>In the case of lgamma(), do <i>not</i> use the expression signgam*exp(lgamma(x)) to compute 'g := $\Gamma(x)$'. Instead compute lgamma() first:</p> <pre>lg = lgamma(x); g = signgam*exp(lg);</pre> <p>only after lgamma() has returned can signgam be correct. Note that $\Gamma(x)$ 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.
SEE ALSO	matherr(3M)
NOTES	<p>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.</p> <p>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.</p> <p>lgamma() is unsafe in multithreaded applications. lgamma_r() should be used instead.</p>

NAME	lio_listio – list directed I/O
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lposix4 [<i>library</i> ...] #include <aio.h> int lio_listio(int <i>mode</i>, struct aiocb * const <i>list</i>[], int <i>nent</i>, struct sigevent *<i>sig</i>); struct aiocb { int aio_fildes; /* file descriptor */ volatile void *aio_buf; /* buffer location */ size_t aio_nbytes; /* length of transfer */ off_t aio_offset; /* file offset */ int aio_reqprio; /* request priority offset */ struct sigevent aio_sigevent; /* signal number and offset */ int aio_lio_opcode; /* listio operation */ }; struct sigevent { int sigev_notify; /* notification mode */ int sigev_signo; /* signal number */ union signal sigev_value; /* signal value */ }; union signal { int sival_int; /* integer value */ void *sival_ptr; /* pointer value */ };</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>lio_listio() allows the calling process, LWP, or thread, to initiate a list of I/O requests within a single function call.</p> <p>If <i>mode</i> is set to LIO_WAIT, lio_listio() behaves synchronously, waiting until all I/O is completed, and the <i>sig</i> argument is ignored. If <i>mode</i> is set to LIO_NOWAIT, lio_listio() behaves asynchronously; returning immediately, and signal delivery will occur, according to the <i>sig</i> argument, when all the I/O operations from this function complete. If <i>sig</i> is NULL, or the sigev_signo member of the sigevent structure referenced by <i>sig</i> is zero, then no signal delivery will occur. Otherwise, the signal number indicated by sigev_signo will be delivered when all the requests in <i>list</i> have completed.</p> <p><i>list</i> is an array of pointers to aiocb structures. This array consists of <i>nent</i> elements. The array may contain NULL pointers, which will be ignored.</p>

The **lio_opcode** field of each **aiocb** structure in *list* specifies the operation to be performed (see `/usr/include/aio.h`).

LIO_READ

requests **aio_read(3R)**.

LIO_WRITE

requests **aio_write(3R)**.

LIO_NOP

causes the *list* entry to be ignored.

nent specifies the length of the array (number of members of the list).

When *mode* has the value **LIO_WAIT**, a pointer to a signal control structure, *sig*, is used to define both the signal to be generated and how the calling process will be notified upon I/O completion. If *sig->sigev_notify* 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 *sig->sigev_notify* is **SIGEV_SIGNAL**, then the signal specified in *sig->sigev_signo* 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 *sig->sigev_value* will be the **si_value** component of the generated signal (see **siginfo(5)**).

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**.)

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 condition.

If the *mode* argument has the value **LIO_WAIT**, and when all the indicated I/O has completed successfully, **lio_listio()** returns **0**; otherwise, it returns **-1**, and sets **errno** to indicate the error condition.

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 *aiocb* 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

EAGAIN The resources necessary to queue all the I/O requests were not available. The error status for each request is recorded in the **aio_error** member of the corresponding **aiocb** structure, and can be retrieved using **aio_error(3R)**.

nent entries exceed the system-wide limit, **AIO_MAX**.

EINVAL

The *mode* argument is an improper value.

	The value of <i>nent</i> is greater than AIO_LISTIO_MAX .
EINTR	A signal was delivered while waiting for all I/O requests to complete during an LIO_WAIT operation. However, the outstanding I/O requests are not canceled. 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. Using aio_error(3R) with each aio_cb structure will determine the individual request(s) that failed.
ENOSYS	lio_listio() is not supported by this implementation.

If either **lio_listio()** succeeds in queuing all of its requests, or **errno** is set to **EAGAIN**, **EINTR**, or **EIO**, then some of the I/O specified from the list may have been initiated. In this event, each **aio_cb** structure contains errors specific to the **read(2)** or **write(2)** function being performed; i.e.:

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.
EINPROGRESS	The requested I/O is in progress.

SEE ALSO **close(2)**, **exec(2)**, **exit(2)**, **fork(2)**, **lseek(2)**, **read(2)**, **write(2)**, **aio_cancel(3R)**, **aio_read(3R)**, **aio_return(3R)**, **fcntl(5)**, **siginfo(5)**

BUGS In Solaris 2.4, these functions always return **-1** and set **errno** to **ENOSYS**, because this release does not support the Asynchronous Input and Output option. It is our intention to provide support for these interfaces in future releases.

NAME	listen – listen for connections 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> int listen(int <i>s</i>, int <i>backlog</i>);</pre>
MT-LEVEL	Safe
DESCRIPTION	<p>To accept connections, a socket is first created with socket(3N), a backlog for incoming connections is specified with listen() and then the connections are accepted with accept(3N). The listen() call applies only to sockets of type SOCK_STREAM or SOCK_SEQPACKET.</p> <p>The <i>backlog</i> parameter defines the maximum length the queue of pending connections may grow to.</p> <p>If a connection request arrives with the queue full, the client will receive an error with an indication of ECONNREFUSED for AF_UNIX sockets. If the underlying protocol supports retransmission, the connection request may be ignored so that retries may succeed. For AF_INET sockets, the tcp will retry the connection. If the <i>backlog</i> is not cleared by the time the tcp times out, the connect will fail with ETIMEDOUT.</p>
RETURN VALUES	A 0 return value indicates success; -1 indicates an error.
ERRORS	<p>The call fails if:</p> <p>EBADF The argument <i>s</i> is not a valid file descriptor.</p> <p>ENOTSOCK The argument <i>s</i> is not a socket.</p> <p>EOPNOTSUPP The socket is not of a type that supports the operation listen().</p>
SEE ALSO	accept(3N) , connect(3N) , socket(3N)
NOTES	There is currently no <i>backlog</i> limit.

NAME	localeconv – get numeric formatting information
SYNOPSIS	<pre>#include <locale.h> struct lconv *localeconv(void);</pre>
MT-LEVEL	Safe with exceptions
DESCRIPTION	<p>localeconv() sets the components of an object with type struct lconv (defined in <locale.h>) 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 struct lconv 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 char * are strings, any of which (except decimal_point) can point to a null string (“”), to indicate that the value is not available in the current locale or is of zero length. The members with type char are nonnegative numbers, any of which can be CHAR_MAX (defined in the <limits.h> header) to indicate that the value is not available in the current locale. The members are the following:</p> <p>char *decimal_point The decimal-point character used to format non-monetary quantities.</p> <p>char *thousands_sep 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:

CHAR_MAX No further grouping is to be performed.

0 The previous element is to be repeatedly used for the remainder of the digits.

other 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 nonnegative-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 nonnegative 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 nonnegative 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 nonnegative 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

localeconv() 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()**.

EXAMPLES

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	-L.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*

SEE ALSO **chrtbl(1M)**, **montbl(1M)**, **setlocale(3C)**

NOTES **localeconv()** can be used safely in a multi-thread application, as long as **setlocale(3C)** is not being called to change the locale.

LC_MONETARY

Determines how monetary formats are handled. In the "C" locale, monetary handling follows the U.S. rules.

LC_NUMERIC

Determines how numeric formats are handled. In the "C" locale, numeric handling follows the U.S. rules.

NAME	lockf – record locking on files
SYNOPSIS	<pre>#include <unistd.h> int lockf(int <i>fildes</i>, int <i>function</i>, long <i>size</i>);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>lockf() 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><i>fildes</i> 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><i>function</i> 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 return 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><i>size</i> 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 added to the table of active locks and this table is already full, an error is returned, and the new section is not locked.</p>

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.

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.

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, a value of `0` is returned. Otherwise, a value of `-1` is returned and **errno** is set to indicate the error.

ERRORS

lockf() will fail if one or more of the following are true:

EAGAIN	<i>cmd</i> is F_TLOCK or F_TEST and the section is already locked by another process.
EBADF	<i>fildev</i> is not a valid open descriptor.
ECOMM	<i>fildev</i> is on a remote machine and the link to that machine is no longer active.
EDEADLK	<i>cmd</i> is F_LOCK and a deadlock occurred.
EDEADLK	<i>cmd</i> is F_LOCK , F_TLOCK , or F_ULOCK and the number of entries in the lock table exceeded the number allocated on the system.

SEE ALSO

intro(2), **alarm(2)**, **chmod(2)**, **close(2)**, **creat(2)**, **fcntl(2)**, **open(2)**, **read(2)**, **write(2)**

NOTES

Unexpected results may occur in processes that do buffering in the user address space. The process may later read/write data that is/was locked. The standard I/O package is the most common source of unexpected buffering.

In the past, the variable **errno** was set to **EACCES** rather than **EAGAIN**. When a section of a file is already locked by another process, portable application programs should expect and test for either value.

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>
MT-LEVEL	Safe
DESCRIPTION	<p>lsearch() 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. <i>key</i> points to the datum to be sought in the table. <i>base</i> points to the first element in the table. <i>nelp</i> 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. <i>width</i> is the size of an element in bytes. <i>compar</i> is a pointer to the comparison function that the user must supply (strcmp, 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>lfind() 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>Note that:</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.
EXAMPLES	<p>This program will read in less than TABSIZE strings of length less than ELSIZE and store them in a table, eliminating duplicates, and then will print each entry.</p> <pre>#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 */</pre>

```
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;
}
```

SEE ALSO **bsearch(3C)**, **hsearch(3C)**, **string(3C)**, **tsearch(3C)**

The Art of Computer Programming, Volume 3, Sorting and Searching by Donald E. Knuth, published by Addison-Wesley Publishing Company, 1973.

NOTES If the searched-for datum is found, both **lsearch()** and **lfind()** return a pointer to it. Otherwise, **lfind()** returns NULL and **lsearch()** returns a pointer to the newly added element.

Undefined results can occur if there is not enough room in the table to add a new item.

NAME	madvise – provide advice to VM system
SYNOPSIS	<pre>#include <sys/types.h> #include <sys/mman.h> int madvise(caddr_t addr, size_t len, int advice);</pre>
MT-LEVEL	MT-Safe
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 <code>cat(1)</code> and <code>cp(1)</code> utilities.</p> <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.</p>

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.

SEE ALSO

cat(1), **cp(1)**, **mmap(2)**, **sysconf(3C)**

NAME	maillock – manage lockfile for user’s mailbox
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lmail [<i>library</i> ...] #include <maillock.h> int maillock(const char *user, int retrycnt); int mailunlock(void);</pre>
MT-LEVEL	Unsafe
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 it’s own lockfile.</p> <p>It is crucial that programs locking mail files refresh their locks at least every minute. Lockfiles are refreshed by calling the routine touchlock() with no arguments. It should be called before doing anything that might keep the system busy for a long time. It should be called at least every 3 minutes or the process may lose its lock on the mail file. In practice, arranging to call it once a minute works well.</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(). <i>user</i> is the login name of the user for whose mailbox the lockfile will be created. maillock() assumes that users’ mailfiles are in the “standard” place as defined in <maillock.h>.</p>
RETURN VALUES	<p>The following return code definitions are contained in <maillock.h>.</p> <pre>#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 */</pre>
FILES	<pre>LIBDIR/lib-mail.ln LIBDIR/mail.a /var/mail/* /var/mail/*.lock</pre>

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, ucontext_t *ucp);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>These functions are useful for implementing user-level context switching between multiple threads of control within a process.</p> <p>makecontext() 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 match the number of arguments that follow <i>argc</i>. Otherwise the behavior is undefined.</p> <p>swapcontext() 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 return a value of zero. Otherwise, a value of -1 is returned and errno is set to indicate the error.
ERRORS	<p>These functions will fail if either of the following is true:</p> <p>EFAULT <i>ucp</i> or <i>oucp</i> points to an invalid address.</p> <p>ENOMEM <i>ucp</i> does not have enough stack left to complete the operation.</p>
SEE ALSO	exit(2) , getcontext(2) , sigaction(2) , sigprocmask(2) , 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>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>The makedev() routine returns a formatted device number on success and NODEV on failure. <i>maj</i> is the major number. <i>min</i> is the minor number. makedev() can be used to create a device number for input to mknod(2).</p> <p>The major() routine returns the major number component from <i>device</i>.</p> <p>The minor() routine returns the minor number component from <i>device</i>.</p>
RETURN VALUES	On failure, NODEV is returned and errno is set to indicate the error.
ERRORS	<p>makedev() will fail if one or more of the following are true:</p> <ul style="list-style-type: none">EINVAL One or both of the arguments <i>maj</i> and <i>min</i> is too large.EINVAL The <i>device</i> number created from <i>maj</i> and <i>min</i> is NODEV. <p>major() will fail if one or more of the following are true:</p> <ul style="list-style-type: none">EINVAL The <i>device</i> argument is NODEV.EINVAL The major number component of <i>device</i> is too large. <p>minor() will fail if the following is true:</p> <ul style="list-style-type: none">EINVAL The <i>device</i> argument is NODEV.
SEE ALSO	mknod(2) , stat(2)

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>
MT-LEVEL	Safe
DESCRIPTION	<p>malloc() and free() provide a simple general-purpose memory allocation package. 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(), calloc() or realloc(). After free() is performed this space is made available for further allocation. If <i>ptr</i> 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 handed to free().</p> <p>calloc() allocates space for an array of <i>nelem</i> elements of size <i>elsize</i>. The space is initialized to zeros.</p> <p>memalign() allocates <i>size</i> 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 <i>alignment</i>. Note: the value of <i>alignment</i> must be a power of two, and must be greater than or equal to the size of a word.</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 will be unchanged up to the lesser of the new and old sizes. If <i>ptr</i> is NULL, 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 pointed to is freed.</p> <p>valloc() is equivalent to memalign(sysconf(_SC_PAGESIZE),size).</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>malloc(), realloc(), memalign(), and valloc() will fail if there is not enough available memory.</p> <p>alloca() allocates <i>size</i> 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. Note: if the allocated block is beyond the current stack limit, the resulting behavior is undefined.</p>

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.

SEE ALSO

brk(2), **getrlimit(2)**, **bsdmalloc(3X)**, **malloc(3X)**

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()**.

alloca() is machine-, compiler-, and most of all, system-dependent. Its use is strongly discouraged.

NOTES

Comparative Features of **malloc(3C)**, **bsdmalloc(3X)**, and **malloc(3X)**:

- 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(3C)** routines are a trade-off between performance and space-efficiency.

NAME	malloc, free, realloc, calloc, mallopt, mallinfo – memory allocator
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lmalloc [<i>library</i> ...] #include <stdlib.h> void *malloc(size_t size); void free(void *ptr); void *realloc(void *ptr, size_t size); void *calloc(size_t nelem, size_t elsize); #include <malloc.h> int mallopt(int cmd, int value); struct mallinfo mallinfo(void);</pre>
MT-LEVEL	Safe
DESCRIPTION	<p>malloc() and free() provide a simple general-purpose memory allocation package. malloc() returns a pointer to a block of at least <i>size</i> bytes suitably aligned for any use. 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 mallopt() 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> <p>calloc() allocates space for an array of <i>nelem</i> elements of size <i>elsize</i>. The space is initialized to zeros.</p> <p>mallopt() provides for control over the allocation algorithm. The available values for <i>cmd</i> are:</p> <p>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 doles them out very quickly. The default value for <i>maxfast</i> is 24.</p> <p>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.</p> <p>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</p>

alignment of any data type. Value will be rounded up to a multiple of the default when *grain* 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 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:

```

int arena;      /* total space in arena */
int ordblks;    /* number of ordinary blocks */
int smlblks;    /* number of small blocks */
int hblkhd;     /* space in holding block headers */
int hblks;     /* number of holding blocks */
int usmlblks;   /* space in small blocks in use */
int fsmblks;   /* space in free small blocks */
int uordblks;   /* space in ordinary blocks in use */
int fordblks;   /* space in free ordinary blocks */
int keepcost;   /* space penalty if keep option */
               /* 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.

SEE ALSO

brk(2), **malloc(3C)**, **bsdmalloc(3X)**

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.

NAME	mapmalloc, calloc, cfree, free, realloc, – 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 cfree(void *ptr, unsigned num, unsigned size); void free(void *ptr); void *realloc(void *ptr, size_t size);</pre>
MT-LEVEL	Safe
DESCRIPTION	<p>The collection of malloc routines in this library use mmap(2) instead of sbrk(2) for acquiring new heap space. The routines in this library are intended to be used only if necessary, when applications must call sbrk(), but need to call other library routines that might call malloc. The algorithms used by these routines are not sophisticated. There is no reclaiming of memory.</p> <p>malloc() and free() provide a simple general-purpose memory allocation package. malloc() returns a pointer to a block of at least <i>size</i> bytes suitably aligned for any use. The argument to free() is a pointer to a block previously allocated by malloc(), calloc() or realloc(). If <i>ptr</i> 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 handed to free().</p> <p>calloc() allocates space for an array of <i>nelem</i> elements of size <i>elsize</i>. The space is initialized to zeros.</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 will be unchanged up to the lesser of the new and old sizes. If <i>ptr</i> is NULL, 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 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>malloc() and realloc() will fail if there is not enough available memory.</p> <p>Entry points for malloc_debug(), malloccmap(), mallopt(), mallinfo(), memalign(), and valloc(), are empty routines, and are provided only to protect the user from mixing malloc() functions from different implementations.</p>
RETURN VALUES	<p>If there is no available memory, malloc(), realloc(), and calloc() return a null pointer. When realloc() returns NULL, the block pointed to by <i>ptr</i> is left intact. If <i>size</i>, <i>nelem</i>, or <i>elsize</i> is 0, a unique pointer to the arena is returned.</p>

FILES /usr/lib/libmapmalloc

SEE ALSO brk(2), getrlimit(2), mmap(2), realloc(3C)

NAME matherr – math library exception-handling function

SYNOPSIS `#include <math.h>`
`int matherr(struct exception *exc);`

MT-LEVEL MT-Safe

DESCRIPTION The SVID3 (*System V Interface Definition Third Edition*) specifies that certain **libm** functions call **matherr()** when exceptions are detected. Users may define their own mechanisms for handling exceptions, by including a function named **matherr()** in their programs. **matherr()** is of the form described above. When an exception occurs, a pointer to the exception structure *exc* will be passed to the user-supplied **matherr()** function. This structure, which is defined in the `<math.h>` header file, is as follows:

```
struct exception {
    int type;
    char *name;
    double arg1, arg2, retval;
};
```

The element **type** is an integer describing the type of exception that has occurred, from the following list of constants (defined in the header file):

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

Note that both **TLOSS** and **PLOSS** reflect limitations of particular algorithms for trigonometric functions that suffer abrupt declines in accuracy at definite boundaries. Since the Sun implementation does not suffer such abrupt declines, **PLOSS** is never signaled. **TLOSS** is signaled for Bessel functions *only* to satisfy SVID3 requirements.

The element **name** points to a string containing the name of the function that incurred the exception. The elements **arg1** and **arg2** are the arguments with which the function was invoked. **retval** is set to the default value that will be returned by the function unless the user's **matherr()** sets it to a different value.

If the user's **matherr()** function returns non-zero, no exception message will be printed, and **errno** will not be set.

**SVID3
STANDARD
CONFORMANCE**

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, errno 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, **errno** 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, **errno** is set to **ERANGE**.

UNDERFLOW

0.0 is returned, and **errno** is set to **ERANGE**.

TLOSS 0.0 is returned, **errno** 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.

DEFAULT ERROR HANDLING PROCEDURES (SVID3)					
Types of Errors					
<math.h> type	DOMAIN	SING	OVERFLOW	UNDERFLOW	TLOSS
errno	EDOM	EDOM	ERANGE	ERANGE	ERANGE
IEEE Exception	Invalid Operation	Division by Zero	Overflow	Underflow	–
fp_exception_type	fp_invalid	fp_division	fp_overflow	fp_underflow	–
ACOS, ASIN ($ x > 1$):	Md, 0.0	–	–	–	–
ACOSH ($x < 1$), ATANH ($ x > 1$):	NaN	–	–	–	–
ATAN2 (0,0):	Md, 0.0	–	–	–	–
COSH, SINH:	–	–	\pm HUGE	–	–
EXP:	–	–	+HUGE	0.0	–
FMOD (x,0):	x	–	–	–	–
HYPOT:	–	–	+HUGE	–	–
J0, J1, JN ($ x > X_TLOSS$):	–	–	–	–	Mt, 0.0
LGAMMA: usual cases ($x = 0$ or $-integer$)	– –	– Ms, +HUGE	+HUGE –	– –	– –
LOG, LOG10: ($x < 0$) ($x = 0$)	Md, –HUGE –	– Ms, –HUGE	– –	– –	– –
POW: usual cases ($x < 0$) ** (y not an integer) 0 ** 0 0 ** (y < 0)	– Md, 0.0 Md, 0.0 Md, 0.0	– – – –	\pm HUGE – – –	\pm 0.0 – – –	– – – –
REMAINDER (x,0):	NaN	–	–	–	–
SCALB:	–	–	\pm HUGE_VAL	\pm 0.0	–
SQRT ($x < 0$):	Md, 0.0	–	–	–	–
Y0, Y1, YN: ($x < 0$) ($x = 0$) ($x > X_TLOSS$)	Md, –HUGE – –	– Md, –HUGE –	– – –	– – –	– – 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()** may not be called on overflow or underflow, and the SUN-provided **matherr()** may return results that differ from those in this table.

**X/OPEN (XPG3)
STANDARD
CONFORMANCE**

XPG3 (*X/Open Portability Guide Issue 3*) no longer sanctions the use of the **matherr()** interface. The following table summarizes the values returned in the exceptional cases. In general, XPG3 dictates that as long as one of the input argument(s) is a NaN, NaN shall be returned. In particular, **pow(NaN,0) = NaN**.

X/Open (XPG3) ERROR HANDLING PROCEDURES (compile with "cc -Xa")					
Types of Errors					
<math.h> type	DOMAIN	SING	OVERFLOW	UNDERFLOW	TLOSS
errno	EDOM	EDOM	ERANGE	ERANGE	ERANGE
ACOS, ASIN ($ x > 1$):	0.0	-	-	-	-
ATAN2 (0,0):	0.0	-	-	-	-
COSH, SINH:	-	-	{ \pm HUGE_VAL}	-	-
EXP:	-	-	{+HUGE_VAL}	{0.0}	-
FMOD (x,0):	{NaN}	-	-	-	-
HYPOT:	-	-	{+HUGE_VAL}	-	-
J0, J1, JN ($ x > X_TLOSS$):	-	-	-	-	{0.0}
LGAMMA: usual cases (x = 0 or -integer)	- -	- +HUGE_VAL	{+HUGE_VAL} -	- -	- -
LOG, LOG10: (x < 0) (x = 0)	-HUGE_VAL -	- -HUGE_VAL	- -	- -	- -
POW: usual cases (x < 0) ** (y not an integer) 0 ** 0 0 ** (y < 0)	- 0.0 {1.0} {-HUGE_VAL}	- - - -	\pm HUGE_VAL - - -	\pm 0.0 - - -	- - - -
SQRT (x < 0):	0.0	-	-	-	-
Y0, Y1, YN: (x < 0) (x = 0) (x > X_TLOSS)	{-HUGE_VAL} - -	- {-HUGE_VAL} -	- - -	- - -	- - 0.0

**ANSI/ISO-C
STANDARD
CONFORMANCE**

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>.

The ANSI/ISO-C standard covers a small subset of XPG3.

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) (x = 0)	-HUGE_VAL -	- -HUGE_VAL	- -	- -
POW: usual cases (x < 0) ** (y not an integer) 0 ** (y < 0)	- 0.0 -HUGE_VAL	- - -	\pm HUGE_VAL - -	\pm 0.0 - -
SQRT (x < 0):	0.0	-	-	-

ABBREVIATIONS	
NaN	IEEE NaN result and invalid operation exception.
HUGE_VAL	IEEE ∞ result and division-by-zero exception.

EXAMPLES

```
#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 */  
}
```

NAME	mbchar, mbtowc, mblen, wctomb – multibyte character handling
SYNOPSIS	<pre>#include <stdlib.h> #include <limits.h> int mbtowc(wchar_t *pwc, const char *s, size_t n); int mblen(const char *s, size_t n); int wctomb(char *s, wchar_t wchar);</pre>
MT-LEVEL	MT-Safe with exceptions
DESCRIPTION	<p>Multibyte characters are used to represent characters in an extended character set. This is needed for locales where 8 bits are not enough to represent all the characters in the character set.</p> <p>The multibyte character handling functions provide the means of translating multibyte characters into wide characters and back again. Wide characters have type wchar_t (defined in <stdlib.h>), which is an integral type whose range of values can represent distinct codes for all members of the largest extended character set specified among the supported locales.</p> <p>A maximum of 3 extended character sets are supported for each locale. The number of bytes in an extended character set is defined by the LC_CTYPE category of the locale (see setlocale(3C)). However, the maximum number of bytes in any multibyte character will never be greater than MB_LEN_MAX, which is defined in <limits.h>. The maximum number of bytes in a character in an extended character set in the current locale is given by the macro, MB_CUR_MAX, defined in <stdlib.h>.</p> <p>mbtowc() determines the number of bytes that comprise the multibyte character pointed to by <i>s</i>. Also, if <i>pwc</i> is not a null pointer, mbtowc() converts the multibyte character to a wide character and places the result in the object pointed to by <i>pwc</i>. (The value of the wide character corresponding to the null character is zero.) At most <i>n</i> bytes will be examined, starting at the byte pointed to by <i>s</i>.</p> <p>If <i>s</i> is a null pointer, mbtowc() simply returns 0. If <i>s</i> is not a null pointer, then, if <i>s</i> points to the null character, mbtowc() returns 0; if the next <i>n</i> or fewer bytes form a valid multibyte character, mbtowc() returns the number of bytes that comprise the converted multibyte character; otherwise, <i>s</i> does not point to a valid multibyte character and mbtowc() returns -1.</p> <p>mblen() determines the number of bytes comprising the multibyte character pointed to by <i>s</i>. It is equivalent to</p> <pre>mbtowc((wchar_t *)0, s, n);</pre> <p>wctomb() determines the number of bytes needed to represent the multibyte character corresponding to the code whose value is <i>wchar</i>, and, if <i>s</i> is not a null pointer, stores the multibyte character representation in the array pointed to by <i>s</i>. At most MB_CUR_MAX bytes are stored.</p>

If *s* is a null pointer, **wctomb()** simply returns 0. If *s* is not a null pointer, **wctomb()** returns -1 if the value of *wchar* does not correspond to a valid multibyte character; otherwise it returns the number of bytes that comprise the multibyte character corresponding to the value of *wchar*.

SEE ALSO [chrtbl\(1M\)](#), [mbstring\(3C\)](#), [setlocale\(3C\)](#), [environ\(5\)](#)

NOTES [mbchar](#), [mbtowc](#), [mblen](#) and [wctomb](#) can be used safely in a multi-thread application, as long as [setlocale\(3C\)](#) is not being called to change the locale.

NAME	mbstring, mbstowcs, wcstombs – multibyte string functions
SYNOPSIS	<pre>#include <stdlib.h> size_t mbstowcs(wchar_t *pwcs, const char *s, size_t n); size_t wcstombs(char *s, const wchar_t *pwcs, size_t n);</pre>
MT-LEVEL	MT-Safe with exceptions
DESCRIPTION	<p>mbstowcs() converts a sequence of multibyte characters from the array pointed to by <i>s</i> into a sequence of corresponding wide character codes and stores these codes into the array pointed to by <i>pwcs</i>, stopping after <i>n</i> codes are stored or a code with value zero (a converted null character) is stored. If an invalid multibyte character is encountered, mbstowcs() returns (size_t) -1; otherwise, mbstowcs() returns the number of array elements modified, not including the terminating zero code, if any.</p> <p>wcstombs() converts a sequence of wide character codes from the array pointed to by <i>pwcs</i> into a sequence of multibyte characters and stores these multibyte characters into the array pointed to by <i>s</i>, stopping if a multibyte character would exceed the limit of <i>n</i> total bytes or if a null character is stored. If a wide character code is encountered that does not correspond to a valid multibyte character, wcstombs() returns (size_t) -1; otherwise, wcstombs() returns the number of bytes modified, not including a terminating null character, if any. If <i>s</i> is a null pointer, wcstombs() returns the number of bytes required for the character array.</p>
SEE ALSO	chrtbl(1M) , mbchar(3C) , setlocale(3C) , environ(5)
NOTES	mbstowcs and wcstombs can be used safely in a multi-thread application, as long as setlocale(3C) is not being called to change the locale.

NAME	mctl – memory management control										
SYNOPSIS	<pre> /usr/ucb/cc [flag ...] file ... #include <sys/types.h> #include <sys/mman.h> int mctl(addr, len, function, arg) caddr_t addr; size_t len; int function; int arg; </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 mman.h 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> <table border="0" style="margin-left: 40px;"> <tr> <td style="padding-right: 20px;">MCL_CURRENT</td> <td>Lock current mappings</td> </tr> <tr> <td>MCL_FUTURE</td> <td>Lock future mappings</td> </tr> </table> <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> <table border="0" style="margin-left: 40px;"> <tr> <td style="padding-right: 20px;">MS_SYNC</td> <td>Synchronized write</td> </tr> <tr> <td>MS_ASYNC</td> <td>Return immediately</td> </tr> <tr> <td>MS_INVALIDATE</td> <td>Invalidate mappings</td> </tr> </table> <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 memory, requiring them to be re-obtained from the object's permanent storage location upon the next reference.</p> <p>MC_UNLOCK Unlock the pages in the range. This function is used to support munlock(). <i>arg</i> is ignored.</p>	MCL_CURRENT	Lock current mappings	MCL_FUTURE	Lock future mappings	MS_SYNC	Synchronized write	MS_ASYNC	Return immediately	MS_INVALIDATE	Invalidate mappings
MCL_CURRENT	Lock current mappings										
MCL_FUTURE	Lock future mappings										
MS_SYNC	Synchronized write										
MS_ASYNC	Return immediately										
MS_INVALIDATE	Invalidate mappings										

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(3B)**, **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	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>
MT-LEVEL	MT-Safe
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>memccpy() copies bytes from memory area <i>s2</i> into <i>s1</i>, stopping after the first occurrence of <i>c</i> (converted to an unsigned char) 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>memchr() returns a pointer to the first occurrence of <i>c</i> (converted to an unsigned char) in the first <i>n</i> bytes (each interpreted as an unsigned char) of memory area <i>s</i>, or a null pointer if <i>c</i> does not occur.</p> <p>memcmp() compares its arguments, looking at the first <i>n</i> bytes (each interpreted as an unsigned char), 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>memcpy() copies <i>n</i> bytes from memory area <i>s2</i> to <i>s1</i>. It returns <i>s1</i>.</p> <p>memmove() 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>memset() sets the first <i>n</i> bytes in memory area <i>s</i> to the value of <i>c</i> (converted to an unsigned char). It returns <i>s</i>.</p>
SEE ALSO	string(3C)

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>#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>						
MT-LEVEL	Unsafe						
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 curses 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 curses 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 curses 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	<p>These routines return one of the following:</p> <table border="0"> <tr> <td style="padding-right: 20px;">E_OK</td> <td>The routine returned successfully.</td> </tr> <tr> <td>E_SYSTEM_ERROR</td> <td>System error.</td> </tr> <tr> <td>E_BAD_ARGUMENT</td> <td>An incorrect argument was passed to the routine.</td> </tr> </table>	E_OK	The routine returned successfully.	E_SYSTEM_ERROR	System error.	E_BAD_ARGUMENT	An incorrect argument was passed to the routine.
E_OK	The routine returned successfully.						
E_SYSTEM_ERROR	System error.						
E_BAD_ARGUMENT	An incorrect argument was passed to the routine.						
SEE ALSO	curses(3X) , menus(3X)						
NOTES	The header <menu.h> automatically includes the headers <eti.h> and <curses.h> .						

NAME	menu_cursor, pos_menu_cursor – correctly position a menu cursor
SYNOPSIS	<pre>#include <menu.h> int pos_menu_cursor(MENU *menu);</pre>
MT-LEVEL	Unsafe
DESCRIPTION	pos_menu_cursor() moves the cursor in the window of <i>menu</i> 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 The routine returned successfully. E_SYSTEM_ERROR System error. E_BAD_ARGUMENT An incorrect argument was passed to the routine. E_NOT_POSTED The menu has not been posted.
SEE ALSO	curses(3X), menus(3X), panel_update(3X), panels(3X)
NOTES	The header <menu.h> automatically includes the headers <eti.h> and <curses.h> .

NAME	menu_driver – command processor for the menus subsystem																																		
SYNOPSIS	#include <menu.h> int menu_driver(MENU *menu, int c);																																		
MT-LEVEL	Unsafe																																		
DESCRIPTION	<p>menu_driver() is the workhorse of the menus subsystem. It checks to determine whether the character <i>c</i> is a menu request or data. If <i>c</i> is a request, the menu driver executes the request and reports the result. If <i>c</i> 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 E_NO_MATCH. If the character is not recognized, the menu driver assumes it is an application-defined command and returns E_UNKNOWN_COMMAND.</p> <p>Menu driver requests:</p> <table border="0"> <tr> <td>REQ_LEFT_ITEM</td> <td>Move left to an item.</td> </tr> <tr> <td>REQ_RIGHT_ITEM</td> <td>Move right to an item.</td> </tr> <tr> <td>REQ_UP_ITEM</td> <td>Move up to an item.</td> </tr> <tr> <td>REQ_DOWN_ITEM</td> <td>Move down to an item.</td> </tr> <tr> <td>REQ_SCR_ULINE</td> <td>Scroll up a line.</td> </tr> <tr> <td>REQ_SCR_DLINE</td> <td>Scroll down a line.</td> </tr> <tr> <td>REQ_SCR_DPAGE</td> <td>Scroll up a page.</td> </tr> <tr> <td>REQ_SCR_UPAGE</td> <td>Scroll down a page.</td> </tr> <tr> <td>REQ_FIRST_ITEM</td> <td>Move to the first item.</td> </tr> <tr> <td>REQ_LAST_ITEM</td> <td>Move to the last item.</td> </tr> <tr> <td>REQ_NEXT_ITEM</td> <td>Move to the next item.</td> </tr> <tr> <td>REQ_PREV_ITEM</td> <td>Move to the previous item.</td> </tr> <tr> <td>REQ_TOGGLE_ITEM</td> <td>Select/de-select an item.</td> </tr> <tr> <td>REQ_CLEAR_PATTERN</td> <td>Clear the menu pattern buffer.</td> </tr> <tr> <td>REQ_BACK_PATTERN</td> <td>Delete the previous character from pattern buffer.</td> </tr> <tr> <td>REQ_NEXT_MATCH</td> <td>Move the next matching item.</td> </tr> <tr> <td>REQ_PREV_MATCH</td> <td>Move to the previous matching item.</td> </tr> </table>	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.
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.																																		
RETURN VALUES	<p>menu_driver() returns one of the following:</p> <table border="0"> <tr> <td>E_OK</td> <td>The routine returned successfully.</td> </tr> <tr> <td>E_SYSTEM_ERROR</td> <td>System error.</td> </tr> <tr> <td>E_BAD_ARGUMENT</td> <td>An incorrect argument was passed to the routine.</td> </tr> <tr> <td>E_BAD_STATE</td> <td>The routine was called from an initialization or termination function.</td> </tr> <tr> <td>E_NOT_POSTED</td> <td>The menu has not been posted.</td> </tr> </table>	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_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.
E_REQUEST_DENIED	The menu driver could not process the request.

SEE ALSO `curses(3X)`, `menus(3X)`

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>#include <menu.h> int set_menu_format(MENU *menu, int rows, int cols); void menu_format(MENU *menu, int *rows, int *cols);</pre>								
MT-LEVEL	Unsafe								
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> <table><tr><td>E_OK</td><td>The routine returned successfully.</td></tr><tr><td>E_SYSTEM_ERROR</td><td>System error.</td></tr><tr><td>E_BAD_ARGUMENT</td><td>An incorrect argument was passed to the routine.</td></tr><tr><td>E_POSTED</td><td>The menu is already posted.</td></tr></table>	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_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.								
SEE ALSO	curses(3X) , menus(3X)								
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>#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>				
MT-LEVEL	Unsafe				
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. menu_term() returns a pointer to the menu termination routine, if any, called when the <i>menu</i> is unposted and just before the top row changes on a posted menu.</p>				
RETURN VALUES	<p>Routines that return pointers always return NULL on error. Routines that return an integer return one of the following:</p> <table border="0"> <tr> <td style="padding-right: 20px;">E_OK</td> <td>The routine returned successfully.</td> </tr> <tr> <td>E_SYSTEM_ERROR</td> <td>System error.</td> </tr> </table>	E_OK	The routine returned successfully.	E_SYSTEM_ERROR	System error.
E_OK	The routine returned successfully.				
E_SYSTEM_ERROR	System error.				
SEE ALSO	curses(3X) , menus(3X)				

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>#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>										
MT-LEVEL	Unsafe										
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 <i>N</i>-1, where <i>N</i> 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> <table border="0"> <tr> <td style="padding-right: 20px;">E_OK</td> <td>The routine returned successfully.</td> </tr> <tr> <td>E_SYSTEM_ERROR</td> <td>System error.</td> </tr> <tr> <td>E_BAD_ARGUMENT</td> <td>An incorrect argument was passed to the routine.</td> </tr> <tr> <td>E_BAD_STATE</td> <td>The routine was called from an initialization or termination function.</td> </tr> <tr> <td>E_NOT_CONNECTED</td> <td>No items are connected to the menu.</td> </tr> </table>	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_CONNECTED	No items are connected to the menu.
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_CONNECTED	No items are connected to the menu.										
SEE ALSO	curses(3X) , menus(3X)										
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	#include <menu.h> char *item_name(ITEM *item); char *item_description(ITEM *item);
MT-LEVEL	Unsafe
DESCRIPTION	item_name() returns a pointer to the name of <i>item</i> . item_description() returns a pointer to the description of <i>item</i> .
RETURN VALUES	These routines return NULL on error.
SEE ALSO	curses(3X) , menus(3X) , menu_new(3X)
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	#include <menu.h> ITEM *new_item(char *name, char *desc); int free_item(ITEM *item);
MT-LEVEL	Unsafe
DESCRIPTION	new_item() creates a new item from <i>name</i> and <i>description</i> , and returns a pointer to the new item. 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.
RETURN VALUES	new_item() returns NULL on error. free_item() returns 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_CONNECTED One or more items are already connected to another menu.
SEE ALSO	curses(3X), menus(3X)
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>#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>
MT-LEVEL	Unsafe
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 style="padding-left: 20px;">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 The routine returned successfully.</p> <p>E_SYSTEM_ERROR System error.</p>
SEE ALSO	curses(3X) , menus(3X)
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 menus items
SYNOPSIS	<pre>#include <menu.h> int set_item_userptr(ITEM *item, char *userptr); char *item_userptr(ITEM *item);</pre>
MT-LEVEL	Unsafe
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	item_userptr() returns NULL on error. set_item_userptr() returns one of the following: E_OK The routine returned successfully. E_SYSTEM_ERROR System error.
SEE ALSO	curses(3X) , menus(3X)
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 menu item values						
SYNOPSIS	<pre>#include <menu.h> int set_item_value(ITEM *item, int bool); int item_value(ITEM *item);</pre>						
MT-LEVEL	Unsafe						
DESCRIPTION	<p>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 <i>item</i> — 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)).</p> <p>item_value() returns the select value of <i>item</i>, either TRUE (selected) or FALSE (unselected).</p>						
RETURN VALUES	<p>set_item_value() returns one of the following:</p> <table><tr><td>E_OK</td><td>The routine returned successfully.</td></tr><tr><td>E_SYSTEM_ERROR</td><td>System error.</td></tr><tr><td>E_REQUEST_DENIED</td><td>The menu driver could not process the request.</td></tr></table>	E_OK	The routine returned successfully.	E_SYSTEM_ERROR	System error.	E_REQUEST_DENIED	The menu driver could not process the request.
E_OK	The routine returned successfully.						
E_SYSTEM_ERROR	System error.						
E_REQUEST_DENIED	The menu driver could not process the request.						
SEE ALSO	curses(3X) , menus(3X) , menu_opts(3X)						
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	<pre>#include <menu.h> int item_visible(ITEM *item);</pre>
MT-LEVEL	Unsafe
DESCRIPTION	A menu item is visible if it currently appears in the subwindow of a posted menu. item_visible() returns TRUE if <i>item</i> is visible, otherwise it returns FALSE .
SEE ALSO	curses(3X) , menus(3X) , menu_new(3X)
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>#include <menu.h> int set_menu_items(MENU *menu, ITEM **items); ITEM **menu_items(MENU *menu); int item_count(MENU *menu);</pre>										
MT-LEVEL	Unsafe										
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> <table border="0"> <tr> <td style="padding-right: 20px;">E_OK</td> <td>The routine returned successfully.</td> </tr> <tr> <td>E_SYSTEM_ERROR</td> <td>System error.</td> </tr> <tr> <td>E_BAD_ARGUMENT</td> <td>An incorrect argument was passed to the routine.</td> </tr> <tr> <td>E_POSTED</td> <td>The menu is already posted.</td> </tr> <tr> <td>E_CONNECTED</td> <td>One or more items are already connected to another menu.</td> </tr> </table>	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_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.										
SEE ALSO	curses(3X), menus(3X)										
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>#include <menu.h> int set_menu_mark(MENU *menu, char *mark); char *menu_mark(MENU *menu);</pre>
MT-LEVEL	Unsafe
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	menu_mark() returns NULL on error. set_menu_mark() returns 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.
SEE ALSO	curses(3X) , menus(3X)
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>#include <menu.h> MENU *new_menu(ITEM **items); int free_menu(MENU *menu);</pre>								
MT-LEVEL	Unsafe								
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> <table><tr><td>E_OK</td><td>The routine returned successfully.</td></tr><tr><td>E_SYSTEM_ERROR</td><td>System error.</td></tr><tr><td>E_BAD_ARGUMENT</td><td>An incorrect argument was passed to the routine.</td></tr><tr><td>E_POSTED</td><td>The menu is already posted.</td></tr></table>	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_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.								
SEE ALSO	curses(3X) , menus(3X)								
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>#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>												
MT-LEVEL	Unsafe												
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>O_SHOWDESC</td> <td>Display the description of the items.</td> </tr> <tr> <td>O_ROWMAJOR</td> <td>Display the menu in row major order.</td> </tr> <tr> <td>O_IGNORECASE</td> <td>Ignore the case when pattern matching.</td> </tr> <tr> <td>O_SHOWMATCH</td> <td>Place the cursor within the item name when pattern matching.</td> </tr> <tr> <td>O_NONCYCLIC</td> <td>Make certain menu driver requests non-cyclic.</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_NONCYCLIC	Make certain menu driver requests non-cyclic.
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_NONCYCLIC	Make certain menu driver requests non-cyclic.												
RETURN VALUES	<p>Except for menu_opts(), these routines return one of the following:</p> <table border="0"> <tr> <td style="padding-right: 20px;">E_OK</td> <td>The routine returned successfully.</td> </tr> <tr> <td>E_SYSTEM_ERROR</td> <td>System error.</td> </tr> <tr> <td>E_POSTED</td> <td>The menu is already posted.</td> </tr> </table>	E_OK	The routine returned successfully.	E_SYSTEM_ERROR	System error.	E_POSTED	The menu is already posted.						
E_OK	The routine returned successfully.												
E_SYSTEM_ERROR	System error.												
E_POSTED	The menu is already posted.												
SEE ALSO	curses(3X) , menus(3X)												
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>#include <menu.h> char *menu_pattern(MENU *menu); int set_menu_pattern(MENU *menu, char *pat);</pre>
MT-LEVEL	Unsafe
DESCRIPTION	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> .
RETURN VALUES	menu_pattern() returns NULL on error. set_menu_pattern() returns 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_NO_MATCH The character failed to match.
SEE ALSO	curses(3X) , menus(3X)
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>#include <menu.h> int post_menu(MENU *menu); int unpost_menu(MENU *menu);</pre>																
MT-LEVEL	Unsafe																
DESCRIPTION	<p>post_menu() writes <i>menu</i> to the subwindow. The application programmer must use curses library routines to display the menu on the physical screen or call update_panels() if the panels 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> <table border="0"> <tr> <td style="padding-right: 20px;">E_OK</td> <td>The routine returned successfully.</td> </tr> <tr> <td>E_SYSTEM_ERROR</td> <td>System error.</td> </tr> <tr> <td>E_BAD_ARGUMENT</td> <td>An incorrect argument was passed to the routine.</td> </tr> <tr> <td>E_POSTED</td> <td>The menu is already posted.</td> </tr> <tr> <td>E_BAD_STATE</td> <td>The routine was called from an initialization or termination function.</td> </tr> <tr> <td>E_NO_ROOM</td> <td>The menu does not fit within its subwindow.</td> </tr> <tr> <td>E_NOT_POSTED</td> <td>The menu has not been posted.</td> </tr> <tr> <td>E_NOT_CONNECTED</td> <td>No items are connected to the menu.</td> </tr> </table>	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_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_NOT_CONNECTED	No items are connected to the menu.
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_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_NOT_CONNECTED	No items are connected to the menu.																
SEE ALSO	curses(3X) , menus(3X) , panels(3X)																
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>#include <menu.h> char *menu_userptr(MENU *menu); int set_menu_userptr(MENU *menu, char *userptr);</pre>
MT-LEVEL	Unsafe
DESCRIPTION	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> . menu_userptr() returns the user pointer of <i>menu</i> .
RETURN VALUES	menu_userptr() returns NULL on error. set_menu_userptr() returns one of the following: E_OK The routine returned successfully. E_SYSTEM_ERROR System error.
SEE ALSO	curses(3X) , menus(3X)
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>#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>										
MT-LEVEL	Unsafe										
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> <table border="0"> <tr> <td style="padding-right: 20px;">E_OK</td> <td>The routine returned successfully.</td> </tr> <tr> <td>E_SYSTEM_ERROR</td> <td>System error.</td> </tr> <tr> <td>E_BAD_ARGUMENT</td> <td>An incorrect argument was passed to the routine.</td> </tr> <tr> <td>E_POSTED</td> <td>The menu is already posted.</td> </tr> <tr> <td>E_NOT_CONNECTED</td> <td>No items are connected to the menu.</td> </tr> </table>	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_NOT_CONNECTED	No items are connected to the menu.
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_NOT_CONNECTED	No items are connected to the menu.										
SEE ALSO	curses(3X) , menus(3X)										
NOTES	The header <menu.h> automatically includes the headers <eti.h> and <curses.h> .										

NAME menus – character based menus package

SYNOPSIS `#include <menu.h>`

MT-LEVEL Unsafe

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 menus package gives the applications programmer a terminal-independent method of creating and customizing menus for user interaction. The menus 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 menus 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
current_item	menu_item_current(3X)
free_item	menu_item_new(3X)
free_menu	menu_new(3X)
item_count	menu_items(3X)
item_description	menu_item_name(3X)
item_index	menu_item_current(3X)
item_init	menu_hook(3X)
item_name	menu_item_name(3X)
item_opts	menu_item_opts(3X)
item_opts_off	menu_item_opts(3X)
item_opts_on	menu_item_opts(3X)
item_term	menu_hook(3X)
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)

<code>menu_format</code>	<code>menu_format(3X)</code>
<code>menu_grey</code>	<code>menu_attributes(3X)</code>
<code>menu_init</code>	<code>menu_hook(3X)</code>
<code>menu_items</code>	<code>menu_items(3X)</code>
<code>menu_mark</code>	<code>menu_mark(3X)</code>
<code>menu_opts</code>	<code>menu_opts(3X)</code>
<code>menu_opts_off</code>	<code>menu_opts(3X)</code>
<code>menu_opts_on</code>	<code>menu_opts(3X)</code>
<code>menu_pad</code>	<code>menu_attributes(3X)</code>
<code>menu_pattern</code>	<code>menu_pattern(3X)</code>
<code>menu_sub</code>	<code>menu_win(3X)</code>
<code>menu_term</code>	<code>menu_hook(3X)</code>
<code>menu_userptr</code>	<code>menu_userptr(3X)</code>
<code>menu_win</code>	<code>menu_win(3X)</code>
<code>new_item</code>	<code>menu_item_new(3X)</code>
<code>new_menu</code>	<code>menu_new(3X)</code>
<code>pos_menu_cursor</code>	<code>menu_cursor(3X)</code>
<code>post_menu</code>	<code>menu_post(3X)</code>
<code>scale_menu</code>	<code>menu_win(3X)</code>
<code>set_current_item</code>	<code>menu_item_current(3X)</code>
<code>set_item_init</code>	<code>menu_hook(3X)</code>
<code>set_item_opts</code>	<code>menu_item_opts(3X)</code>
<code>set_item_term</code>	<code>menu_hook(3X)</code>
<code>set_item_userptr</code>	<code>menu_item_userptr(3X)</code>
<code>set_item_value</code>	<code>menu_item_value(3X)</code>
<code>set_menu_back</code>	<code>menu_attributes(3X)</code>
<code>set_menu_fore</code>	<code>menu_attributes(3X)</code>
<code>set_menu_format</code>	<code>menu_format(3X)</code>
<code>set_menu_grey</code>	<code>menu_attributes(3X)</code>
<code>set_menu_init</code>	<code>menu_hook(3X)</code>
<code>set_menu_items</code>	<code>menu_items(3X)</code>
<code>set_menu_mark</code>	<code>menu_mark(3X)</code>
<code>set_menu_opts</code>	<code>menu_opts(3X)</code>
<code>set_menu_pad</code>	<code>menu_attributes(3X)</code>
<code>set_menu_pattern</code>	<code>menu_pattern(3X)</code>
<code>set_menu_sub</code>	<code>menu_win(3X)</code>
<code>set_menu_term</code>	<code>menu_hook(3X)</code>
<code>set_menu_userptr</code>	<code>menu_userptr(3X)</code>
<code>set_menu_win</code>	<code>menu_win(3X)</code>
<code>set_top_row</code>	<code>menu_item_current(3X)</code>
<code>top_row</code>	<code>menu_item_current(3X)</code>
<code>unpost_menu</code>	<code>menu_post(3X)</code>

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.

SEE ALSO

curses(3X)

NOTES

The header **<menu.h>** automatically includes the headers **<eti.h>** and **<curses.h>**.

NAME	mkdirp, rmdirp – create, remove directories in a path
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lgen [<i>library</i> ...] #include <libgen.h> int mkdirp(const char *path, mode_t mode); int rmdirp(char *dir, char *dir1);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>mkdirp() creates all the missing directories in the given <i>path</i> with the given <i>mode</i>. See chmod(2) for the values of <i>mode</i>.</p> <p>rmdirp() removes directories in path <i>dir</i>. 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 <i>dir1</i>. rmdirp() returns a 0 only if it is able to remove every directory in the path.</p>
EXAMPLES	<pre>/* create scratch directories */ if(mkdirp("/tmp/sub1/sub2/sub3", 0755) == -1) { fprintf(stderr, "cannot create directory"); exit(1); } chdir("/tmp/sub1/sub2/sub3"); . . . /* cleanup */ chdir("/tmp"); rmdirp("sub1/sub2/sub3");</pre>
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.
SEE ALSO	mkdir(2) , rmdir(2)
NOTES	<p>mkdirp() uses malloc(3C) to allocate temporary space for the string.</p> <p>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.</p> <p>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.</p>

NAME	mkfifo – create a new FIFO
SYNOPSIS	<pre>#include <sys/types.h> #include <sys/stat.h> int mkfifo(const char *path, mode_t mode);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>The mkfifo() routine 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)).</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 S_ISGID bit is set in the parent directory then the group id of the FIFO is inherited from the parent directory.</p> <p>mkfifo() calls the mknod(2) function to make the file.</p>
RETURN VALUES	Upon successful completion a value of 0 is returned. Otherwise, a value of -1 is returned and errno is set to indicate the error.
SEE ALSO	mkdir(1) , chmod(2) , exec(2) , mknod(2) , umask(2) , fs_ufs(4) , stat(5)
NOTES	Bits other than the file permission bits in <i>mode</i> are ignored.

NAME	mktemp – make a unique file name
SYNOPSIS	#include <stdlib.h> char *mktemp(char *template);
MT-LEVEL	Safe
DESCRIPTION	mktemp() 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 Xs ; mktemp() will replace the Xs with a character string that can be used to create a unique file name.
RETURN VALUES	mktemp() will assign to <i>template</i> the empty string if it cannot create a unique name.
SEE ALSO	tmpfile(3S) , tmpnam(3S)
NOTES	mktemp() can create only 26 unique file names per thread for each unique <i>template</i> .

NAME	mktime – converts a tm structure to a calendar time
SYNOPSIS	#include <time.h> time_t mktime(struct tm *timeptr);
MT-LEVEL	Unsafe
DESCRIPTION	<p>mktime() 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 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 timezone is not valid. If tm_isdst is negative, the correct timezone is determined and the components are not adjusted.</p> <p>Local timezone information is used as if mktime() had called tzset() (see ctime(3C)).</p> <p>mktime() returns the specified calendar time. If the calendar time cannot be represented, the function returns the value (time_t)-1.</p>

EXAMPLES

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]);
```

SEE ALSO

ctime(3C), **getenv(3C)**, **TIMEZONE(4)**

NOTES

tm_year of the **tm** structure must be for year 1970 or later. Calendar times before 00:00:00 UTC, January 1, 1970 or after 03:14:07 UTC, January 19, 2038 cannot be represented.

NAME	mlock, munlock – lock (or unlock) pages in memory
SYNOPSIS	<pre>#include <sys/types.h> int mlock(caddr_t addr, size_t len); int munlock(caddr_t addr, size_t len);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>The function mlock() 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 MAP_PRIVATE page is made upon the first store, the lock will be transferred to the newly copied private page.</p> <p>munlock() 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>Because of the impact on system resources, the use of mlock() and munlock() is restricted to the super-user.</p> <p>Attempts to mlock() more memory than a system-specific limit will fail.</p>
RETURN VALUES	Upon successful completion, the functions mlock() and munlock() return 0; otherwise, they return -1 and set errno to indicate the error.
ERRORS	<p>EAGAIN mlock() only. Some or all of the memory identified by the range [<i>addr</i>, <i>addr + len</i>) could not be locked because of insufficient system resources.</p> <p>EINVAL <i>addr</i> is not a multiple of the page size as returned by sysconf(3C).</p> <p>ENOMEM Addresses in the range [<i>addr</i>, <i>addr + len</i>) are invalid for the address space of a process, or specify one or more pages which are not mapped.</p> <p>EPERM The process's effective user ID is not super-user.</p>

SEE ALSO `fork(2)`, `memcntl(2)`, `mmap(2)`, `plock(3C)`, `mlockall(3C)`, `sysconf(3C)`

NOTES `mlock` and `munlock` require super-user privileges.

NAME	mlockall, munlockall – lock or unlock address space						
SYNOPSIS	<pre>#include <sys/mman.h> int mlockall(int flags); int munlockall(void);</pre>						
MT-LEVEL	MT-Safe						
DESCRIPTION	<p>The function mlockall() causes all pages mapped by an address space to be locked in memory.</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> <table border="0"> <tr> <td style="padding-right: 20px;">MCL_CURRENT</td> <td>Lock current mappings</td> </tr> <tr> <td>MCL_FUTURE</td> <td>Lock future mappings</td> </tr> </table> <p>If MCL_FUTURE is specified to mlockall(), then as mappings are added to the address space (or existing mappings are replaced) they will also be locked, provided sufficient memory is available.</p> <p>Mappings locked via mlockall() with any option may be explicitly unlocked with a munlockall() call.</p> <p>The function munlockall() removes address space locks and locks on mappings in the address space.</p> <p>All conditions and constraints on the use of locked memory as exist for mlock() apply to mlockall().</p> <p>Locks established with mlockall() are not inherited by a child process after a fork() and are not nested.</p>	MCL_CURRENT	Lock current mappings	MCL_FUTURE	Lock future mappings		
MCL_CURRENT	Lock current mappings						
MCL_FUTURE	Lock future mappings						
RETURN VALUES	Upon successful completion, the functions mlockall() and munlockall() return 0; otherwise, they return -1 and set errno to indicate the error.						
ERRORS	<table border="0"> <tr> <td style="padding-right: 20px;">EAGAIN</td> <td>mlockall() only. Some or all of the memory in the address space could not be locked due to sufficient resources.</td> </tr> <tr> <td>EINVAL</td> <td><i>flags</i> contains values other than MCL_CURRENT and MCL_FUTURE.</td> </tr> <tr> <td>EPERM</td> <td>The process's effective user ID is not super-user.</td> </tr> </table>	EAGAIN	mlockall() only. Some or all of the memory in the address space could not be locked due to sufficient resources.	EINVAL	<i>flags</i> contains values other than MCL_CURRENT and MCL_FUTURE .	EPERM	The process's effective user ID is not super-user.
EAGAIN	mlockall() only. Some or all of the memory in the address space could not be locked due to sufficient resources.						
EINVAL	<i>flags</i> contains values other than MCL_CURRENT and MCL_FUTURE .						
EPERM	The process's effective user ID is not super-user.						
SEE ALSO	fork(2) , memcntl(2) , mmap(2) , plock(3C) , mlock(3C) , sysconf(3C)						
NOTES	mlockall() and munlockall() require super-user privileges.						

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>
MT-LEVEL	Safe
DESCRIPTION	<p>monitor() is an interface to profil(), and is called automatically with default parameters by any program created by cc(1B) -p. 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 mon.out.</p> <p>The name of the file written by monitor() is controlled by the environment variable PROFDIR. If PROFDIR does not exist, the file mon.out is created in the current directory. If PROFDIR exists but has no value, monitor() does no profiling and creates no output file. If PROFDIR is <i>dirname</i>, and monitor() is called automatically by compilation with cc -p, the file created is <i>dirname/pid.progname</i> where <i>progname</i> is the name of the program.</p> <p><i>lowpc</i> and <i>highpc</i> are the beginning and ending addresses of the region to be profiled.</p> <p><i>buffer</i> is the address of a user-supplied array of WORD (WORD is defined in the header <mon.h>). <i>buffer</i> is used by monitor() to store the histogram generated by profil() and the call counts.</p> <p><i>bufsize</i> identifies the number of array elements in <i>buffer</i>.</p> <p><i>nfunc</i> is the number of call count cells that have been reserved in <i>buffer</i>. Additional call count cells will be allocated automatically as they are needed.</p> <p><i>bufsize</i> should be computed using the following formula:</p> $\begin{aligned} \text{size_of_buffer} = & \text{sizeof}(\text{struct hdr}) + \\ & \text{nfunc} * \text{sizeof}(\text{struct cnt}) + \\ & ((\text{highpc} - \text{lowpc}) / \text{BARSIZE}) * \text{sizeof}(\text{WORD}) + \\ & \text{sizeof}(\text{WORD}) - 1 ; \\ \text{bufsize} = & (\text{size_of_buffer} / \text{sizeof}(\text{WORD})) ; \end{aligned}$ <p>where:</p> <p><i>lowpc</i>, <i>highpc</i>, <i>nfunc</i> are the same as the arguments to monitor();</p>

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 shown below:

```
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

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.

FILES

mon.out

SEE ALSO

cc(1B), **profil(2)**, **end(3C)**, **prof(5)**

NOTE

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.

NAME	mq_close – close a message queue
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lposix4 [<i>library</i> ...] #include <mqqueue.h> int mq_close(mqd_t mqdes);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>mq_close() removes the association between the message queue descriptor, <i>mqdes</i>, and its message queue.</p> <p>If the process (or thread) has registered a notification request to the message queue via this <i>mqdes</i>, this registration is removed and the message queue is available for another process to attach for notification.</p>
RETURN VALUES	Upon successful completion, mq_close() returns 0 ; otherwise, the function returns -1 and sets errno to indicate the error condition.
ERRORS	<p>EBADF <i>mqdes</i> is an invalid message queue descriptor.</p> <p>ENOSYS sem_open() is not supported by this implementation.</p>
SEE ALSO	mq_notify(3R) , mq_open(3R) , mq_unlink(3R)
BUGS	In Solaris 2.4, these functions always return -1 and set errno to ENOSYS , because this release does not support the Message Passing option. It is our intention to provide support for these interfaces in future releases.

NAME	mq_notify – notify process (or thread) that a message is available on a queue
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lposix4 [<i>library</i> ...] #include <mqqueue.h> int mq_notify(mqd_t <i>mqdes</i>, const struct sigevent *<i>notification</i>); struct sigevent { int sigev_notify /* notification type */ int sigev_signo; /* signal number */ union signal sigev_value; /* signal value */ }; union signal { int sival_int; /* integer value */ void *sival_ptr; /* pointer value */ };</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>mq_notify() 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 NULL, 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 becomes non-empty. At any time, only one process may be registered for notification by a specific message queue. Also, 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><i>notification</i> 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 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>notification->sigev_notify</i> is SIGEV_SIGNAL, then the signal specified in <i>notification->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>notification->sigev_value</i> will be the si_value component of the generated signal (see siginfo(5)).</p> <p>If <i>notification</i> is NULL 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>

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.

Any notification registration is removed if the calling process either closes the message queue or exits.

RETURN VALUES

Upon successful completion, **mq_notify()** returns **0**; otherwise, it returns a value of **-1** and sets **errno** to indicate the error condition.

ERRORS

EBADF *mqdes* is not a valid message queue descriptor.
EBUSY A process is already registered for notification by the message queue.
ENOSYS **mq_notify()** is not supported by this implementation.

SEE ALSO

mq_close(3R), **mq_open(3R)**, **mq_receive(3R)**, **mq_send(3R)**, **siginfo(5)**

BUGS

In Solaris 2.4, these functions always return **-1** and set **errno** to **ENOSYS**, because this release does not support the Message Passing option. It is our intention to provide support for these interfaces in future releases.

NAME	mq_open – open a message queue				
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lposix4 [<i>library</i> ...] #include <mqqueue.h> mqd_t mq_open(const char *name, int oflag, /* unsigned long mode, mq_attr attr */ ...); struct mq_attr { long mq_flags; /* message queue flags */ long mq_maxmsg; /* maximum number of messages */ long mq_msgsize; /* maximum message size */ long mq_curmsgs; /* number of messages currently queued */ ... };</pre>				
MT-LEVEL	MT-Safe				
DESCRIPTION	<p>mq_open() establishes a connection to a named message queue, <i>name</i>, returning the address of the message queue descriptor to the caller for subsequent calls to mq_send(3R) or mq_receive(3R). The message queue once opened remains usable by this process until the message queue is closed by a successful call to mq_close(3R), exit(2), or exec(2).</p> <p><i>name</i> 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.</p> <p><i>oflag</i> 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" style="margin-left: 20px;"> <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>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> </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_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, unless combined with **O_EXCL** (see below). Otherwise, a message queue is created without any messages in it. The message queue's user ID is set to the process's effective user ID, and the message queue's group ID is set to the process's effective group ID. The message queue's permission bits will be 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)**). "AND-NOT" those already set in the file mode creation mask of the process.

If *attr* is **NULL**, the message queue is created with the default message queue attributes, (**mq_maxmsg** = 128 and **mq_maxsize** = 1024). If *attr* is non-**NULL**, the message queue **mq_maxmsg** and **mq_msgsize** attributes are set to the values of the corresponding members in the **mq_attr** structure referred to by *attr*.

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.

O_NONBLOCK The setting of this flag is associated with the open message queue descriptor and determines whether a calling **mq_send(3R)** waits for message buffer space or a calling **mq_receive(3R)** waits for messages that are not currently available; or whether the calling function fails, thereby setting **errno** to **EAGAIN**.

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

EACCESS The message queue exists and the permissions specified by *oflag* are denied, or the message queue does not exist and permission to create the message queue is denied.

EEXIST **O_CREAT** and **O_EXCL** are set and the named message queue already exists.

EINTR	The mq_open() operation was interrupted by a signal.
EINVAL	<i>name</i> is not a valid name. O_CREAT was specified in <i>oflag</i> , the value of <i>attr</i> is not NULL , and either mq_maxmsg or mq_msgsize was less than or equal to zero.
EMFILE	The number of open message queue descriptors in this process exceeds {MQ_OPEN_MAX} . The number of open file descriptors in this process exceeds {OPEN_MAX} .
ENAMETOOLONG	The length of the <i>name</i> string exceeds PATH_MAX , or a pathname component is longer than NAME_MAX while _POSIX_NO_TRUNC is in effect.
ENFILE	The system file table is full
ENOENT	O_CREAT is not set and the named message queue, <i>name</i> , does not exist.
ENOSPC	There is insufficient space for the creation of the new message queue.
ENOSYS	mq_open() is not supported by this implementation.

SEE ALSO **exec(2)**, **exit(2)**, **umask(2)**, **sysconf(3C)**, **mq_close(3R)**, **mq_receive(3R)**, **mq_send(3R)**, **mq_setattr(3R)**, **mq_unlink(3R)**,

BUGS In Solaris 2.4, these functions always return **-1** and set **errno** to **ENOSYS**, because this release does not support the Message Passing option. It is our intention to provide support for these interfaces in future releases.

NAME	mq_receive – receive a message from a message queue
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lposix4 [<i>library</i> ...] #include <mqqueue.h> ssize_t mq_receive(mqd_t mqdes, char *msg_ptr, size_t msg_len, unsigned int msg_prio); struct mq_attr { long mq_flags; /* message queue flags */ long mq_maxmsg; /* maximum number of messages */ long mq_msgsize; /* maximum message size */ long mq_curmsgs; /* number of messages currently queued */ ... };</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>mq_receive() 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 mq_msgsize attribute 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 O_NONBLOCK 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 O_NONBLOCK 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 will have been removed from the queue. Otherwise, no message is removed from the queue, the function returns a value of -1 , and sets errno to indicate the error condition.
ERRORS	<p>EAGAIN O_NONBLOCK was set in the message description associated with <i>mqdes</i>, and the specified message queue is empty.</p> <p>EBADF <i>mqdes</i> is not a valid message queue descriptor open for reading.</p> <p>EMSGSIZE <i>msg_len</i> is less than the message size attribute of the message queue.</p> <p>EINTR mq_receive() operation was interrupted by a signal.</p>

ENOSYS **mq_receive()** is not supported by this implementation.

SEE ALSO **mq_open(3R)**, **mq_send(3R)**, **mq_setattr(3R)**

BUGS In Solaris 2.4, these functions always return **-1** and set **errno** to **ENOSYS**, because this release does not support the Message Passing option. It is our intention to provide support for these interfaces in future releases.

NAME	mq_send – send a message to a message queue
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lposix4 [<i>library</i> ...] #include <mqqueue.h> int mq_send(mqd_t mqdes, const char *msg_ptr, size_t msg_len, unsigned int msg_prio); struct mq_attr { long mq_flags; /* message queue flags */ long mq_maxmsg; /* maximum number of messages */ long mq_msgsize; /* maximum message size */ long mq_curmsgs; /* number of messages currently queued */ ... };</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>mq_send() adds the message pointed to by <i>msg_ptr</i> to the message queue specified by <i>mqdes</i>. <i>msg_len</i> specifies the length of the message in bytes pointed to by <i>msg_ptr</i>. The value of <i>msg_len</i> must be less than or equal to the mq_msgsize attribute of the message queue, or mq_send() will fail.</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 <i>msg_prio</i>. 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 is inserted after other messages in the queue, if any, with equal <i>msg_prio</i> priority. The value of <i>msg_prio</i> must be greater than 0, and less than or equal to {MQ_PRIO_MAX}.</p> <p>If the specified message queue is full and if O_NONBLOCK is not set in the message queue description associated with <i>mqdes</i> (see mq_open(3R) and mq_setattr(3R)), mq_send() blocks, waiting until space becomes available to enqueue the message, or until mq_send() is interrupted by a signal. If more than one process (or thread) is waiting to send when space becomes available in the message queue, then the process of the highest priority which has been waiting the longest is unblocked to send its message. If the specified message queue is full and O_NONBLOCK 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 a value of 0; otherwise, no message is enqueued, the function returns -1, and sets errno to indicate the error condition.
ERRORS	<p>EAGAIN The O_NONBLOCK flag is set in the message queue description associated with <i>mqdes</i>, and the specified message queue is full.</p> <p>EBADF <i>mqdes</i> is not a valid message queue descriptor open for writing.</p> <p>EINTR A signal interrupted the call to mq_send()</p>

EMSGSIZE The specified message length, *msg_len*, exceeds the message size attribute of the message queue.

ENOSYS **mq_send()** is not supported by this implementation.

SEE ALSO **mq_open(3R)**, **mq_receive(3R)**, **mq_setattr(3R)**, **sysconf(3C)**

BUGS In Solaris 2.4, these functions always return **-1** and set **errno** to **ENOSYS**, because this release does not support the Message Passing option. It is our intention to provide support for these interfaces in future releases.

NAME	mq_setattr, mq_getattr – set/get message queue attributes
SYNOPSIS	<pre>cc [flag ...] file ... -lposix4 [library ...] #include <mqqueue.h> int mq_setattr(mqd_t mqdes, const struct mq_attr *mqstat, struct mq_attr* omqstat); int mq_getattr(mqd_t mqdes, struct mq_attr *mqstat); struct mq_attr { long mq_flags; /* message queue flags */ long mq_maxmsg; /* maximum number of messages */ long mq_msgsize; /* maximum message size */ long mq_curmsgs; /* number of messages currently queued */ ... };</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>mq_setattr() is used to set attributes associated with the message queue 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>mq_flags The value of this member is either 0 or O_NONBLOCK.</p> <p>The values of mq_maxmsg, mq_msgsize, and mq_curmsgs are ignored by mq_setattr(). 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. mq_getattr() is used to get status information and attributes associated with the message queue specified in <i>mqdes</i>. Upon return, the mq_flags member of the <i>mq_attr</i> structure referenced by <i>mqstat</i> has the value that was set when the message queue was created but also with modifications made by subsequent mq_setattr() calls.</p> <p>The following attributes were set at message queue creation:</p> <p>mq_maxmsg mq_msgsize</p> <p>Upon return, the mq_curmsgs (the number of messages currently on the queue) member of the <i>mq_attr</i> structure referenced by <i>mqstat</i> is set according to the current state of the message queue.</p>
RETURN VALUES	Upon successful completion, these function(s) return 0 ; otherwise, they return -1 , and set errno to indicate the error condition.

mq_setattr(), if successful, also changes the attributes of the message queue as specified.

ERRORS

EBADF *mqdes* is not a valid message queue descriptor.

ENOSYS **mq_setattr()** and **mq_getattr()** are not supported by this implementation.

SEE ALSO

mq_open(3R), **mq_receive(3R)**, **mq_send(3R)**

BUGS

In Solaris 2.4, these functions always return **-1** and set **errno** to **ENOSYS**, because this release does not support the Message Passing option. It is our intention to provide support for these interfaces in future releases.

NAME	mq_unlink – remove a message queue
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lposix4 [<i>library</i> ...] #include <mqqueue.h> int mq_unlink(const char *<i>name</i>);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>mq_unlink() removes the message queue named by <i>name</i>. After a successful call to mq_unlink() with <i>name</i>, a call to mq_open(3R) with the same <i>name</i> will fail if the flag O_CREAT is not set in <i>flags</i>. 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, mq_unlink() does not block (wait) until all references have been closed; it returns immediately.</p>
RETURN VALUES	<p>Upon successful completion, mq_unlink() returns a value of 0; otherwise, the named message queue is not changed by this function call, the function returns a value of -1 and sets errno to indicate the error condition.</p>
ERRORS	<p>EACCESS Permission is denied to unlink the named message queue.</p> <p>ENAMETOOLONG The length of the <i>name</i> string exceeds {PATH_MAX}, or a pathname component is longer than {NAME_MAX} while _POSIX_NO_TRUNC is in effect.</p> <p>ENOENT The named message queue, <i>name</i>, does not exist.</p> <p>ENOSYS mq_unlink() is not supported by this implementation.</p>
SEE ALSO	mq_close(3R) , mq_open(3R)
BUGS	<p>In Solaris 2.4, these functions always return -1 and set errno to ENOSYS, because this release does not support the Message Passing option. It is our intention to provide support for these interfaces in future releases.</p>

NAME	msync – synchronize memory with physical storage										
SYNOPSIS	<pre>#include <sys/types.h> #include <sys/mman.h> int msync(caddr_t addr, size_t len, int flags);</pre>										
MT-LEVEL	MT-Safe										
DESCRIPTION	<p>The function msync() writes all modified copies of pages over the range [<i>addr</i>, <i>addr + len</i>) to their backing storage locations. msync() optionally invalidates any copies so that further references to the pages will be obtained by the system from their backing storage locations. The backing storage for a modified MAP_SHARED mapping is the file the page is mapped to; the backing storage for a modified MAP_PRIVATE mapping is its swap area.</p> <p><i>flags</i> is a bit pattern built from the following values:</p> <table border="0"> <tr> <td style="padding-right: 20px;">MS_ASYNC</td> <td>perform asynchronous writes</td> </tr> <tr> <td>MS_SYNC</td> <td>perform synchronous writes</td> </tr> <tr> <td>MS_INVALIDATE</td> <td>invalidate mappings</td> </tr> </table> <p>If MS_ASYNC is set, msync() returns immediately once all write operations are scheduled; if MS_SYNC is set, msync() does not return until all write operations are completed.</p> <p>MS_INVALIDATE invalidates all cached copies of data in memory, so that further references to the pages will be obtained by the system from their backing storage locations.</p>	MS_ASYNC	perform asynchronous writes	MS_SYNC	perform synchronous writes	MS_INVALIDATE	invalidate mappings				
MS_ASYNC	perform asynchronous writes										
MS_SYNC	perform synchronous writes										
MS_INVALIDATE	invalidate mappings										
RETURN VALUES	Upon successful completion, the function msync() returns 0; otherwise, it returns -1 and sets errno to indicate the error.										
ERRORS	<table border="0"> <tr> <td style="padding-right: 20px;">EBUSY</td> <td>Some or all of the addresses in the range [<i>addr</i>, <i>addr + len</i>) are locked and MS_SYNC with the MS_INVALIDATE option is specified.</td> </tr> <tr> <td>EINVAL</td> <td><i>addr</i> is not a multiple of the page size as returned by sysconf(3C). <i>flags</i> is not some combination of MS_ASYNC and MS_INVALIDATE.</td> </tr> <tr> <td>EIO</td> <td>An I/O error occurred while reading from or writing to the file system.</td> </tr> <tr> <td>ENOMEM</td> <td>Addresses in the range [<i>addr</i>, <i>addr + len</i>) are outside the valid range for the address space of a process, or specify one or more pages that are not mapped.</td> </tr> <tr> <td>EPERM</td> <td>MS_INVALIDATE was specified and one or more of the pages is locked in memory.</td> </tr> </table>	EBUSY	Some or all of the addresses in the range [<i>addr</i> , <i>addr + len</i>) are locked and MS_SYNC with the MS_INVALIDATE option is specified.	EINVAL	<i>addr</i> is not a multiple of the page size as returned by sysconf(3C) . <i>flags</i> 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 [<i>addr</i> , <i>addr + len</i>) 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.
EBUSY	Some or all of the addresses in the range [<i>addr</i> , <i>addr + len</i>) are locked and MS_SYNC with the MS_INVALIDATE option is specified.										
EINVAL	<i>addr</i> is not a multiple of the page size as returned by sysconf(3C) . <i>flags</i> 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 [<i>addr</i> , <i>addr + len</i>) 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.										
SEE ALSO	mmap(2) , mementl(2) , sysconf(3C)										

NOTES

msync() should be used by programs that require a memory object to be in a known state, such as in building transaction facilities.

NAME	mutex, mutex_init, mutex_destroy, mutex_lock, mutex_trylock, mutex_unlock – mutual exclusion locks
SYNOPSIS	<pre>#include <synch.h> int mutex_init(mutex_t *mp, int type, void * arg); int mutex_destroy(mutex_t *mp); int mutex_lock(mutex_t *mp); int mutex_trylock(mutex_t *mp); int mutex_unlock(mutex_t *mp);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>Mutual exclusion locks (mutexes) are used to serialize the execution of threads. They are typically used to ensure that only one thread executes a critical section of code at any one time (mutual exclusion).</p> <p>Mutexes can be used to synchronize threads in this process and other processes if they are allocated in memory that is writable and shared among the cooperating processes (see mmap(2)) and have been initialized for this behavior.</p> <p>Mutexes must be initialized before use. mutex_init() initializes the mutex pointed to by <i>mp</i>. A mutex can potentially have several different types of behavior, specified by <i>type</i>. No current type uses <i>arg</i> although a future type may specify additional behavior parameters via <i>arg</i>. <i>type</i> may be one of the following:</p> <p>USYNC_PROCESS The mutex can be used to synchronize threads in this process and other processes. Only one process should initialize the mutex. <i>arg</i> is ignored.</p> <p>USYNC_THREAD The mutex can be used to synchronize threads in this process, only. <i>arg</i> is ignored.</p> <p>Mutexes may also be initialized by allocation in zeroed memory. In this case a <i>type</i> of USYNC_THREAD is assumed. Multiple threads must not initialize the same mutex simultaneously. A mutex lock must not be re-initialized while other threads may be using it.</p> <p>mutex_destroy() destroys any state associated with the mutex pointed to by <i>mp</i>. The space for storing the mutex is not freed.</p> <p>mutex_lock() locks the mutex pointed to by <i>mp</i>. If the mutex is already locked, the calling thread blocks until the mutex becomes available. When mutex_lock() returns, the mutex is locked and the calling thread is the owner.</p> <p>mutex_trylock() attempts to lock the mutex pointed to by <i>mp</i>. If the mutex is already locked it returns with an error. Otherwise the mutex is locked and the calling thread is the owner.</p> <p>mutex_unlock() unlocks the mutex pointed to by <i>mp</i>. The mutex must be locked and the calling thread must be the one that last locked the mutex (the owner). If any other threads are waiting for the mutex to become available, one of them is unblocked. If the</p>

calling thread is not the owner of the lock, no error status is returned, and the behavior of the program is undefined.

RETURN VALUES

Zero is returned when successful. A non-zero value indicates an error.

ERRORS

If any of the following conditions are detected, these functions fail and return the corresponding value:

EINVAL Invalid argument.

EFAULT *mp* or *arg* points to an illegal address.

If any of the following conditions are detected, **mutex_trylock()** fails and returns the corresponding value:

EBUSY The mutex pointed to by *mp* was already locked.

SEE ALSO

mmap(2), **condition(3T)**

NOTES

In the current implementation, **mutex_lock()**, **mutex_unlock()**, and **mutex_trylock()** do not validate the mutex type. Therefore, **EINVAL** is not returned for an uninitialized mutex or for a mutex with an invalid type. The behavior of these interfaces for mutexes containing an invalid type is unspecified.

For example, the following call to **mutex_lock()** might “hang”. Since **mutex** is allocated from the stack and is not initialized, it may have “junk” data in it. **mutex** needs to be initialized using **mutex_init()**.

```
int global;
```

```
main()
```

```
{
```

```
    mutex_t mutex;
```

```
    /*
```

```
     * The address of this mutex is passed to threads created from
```

```
     * main(). NOTE: this is not recommended style.
```

```
     */
```

```
    ...
```

```
    if (mutex_lock(&mutex) == 0) { /* this call may hang */
```

```
        global++;
```

```
        mutex_unlock(&mutex);
```

```
    } else
```

```
        printf("mutex_lock() failed\n");
```

```
    ...
```

```
}
```

Instead, **mutex** should first be initialized using **mutex_init()**.

```
int global;
```

```
main()
```

```
{  
    mutex_t mutex;  
  
    ...  
    mutex_init(&mutex, USYNC_THREAD, NULL);  
    mutex_lock(&mutex); /* This call is now guaranteed to work */  
    global++;  
    mutex_unlock(&mutex);  
}
```

By default, there is no defined order of acquisition if multiple threads are waiting for a mutex.

These interfaces are also available via:

```
#include <thread.h>
```

NAME	nanosleep – high resolution sleep
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lposix4 [<i>library</i> ...] #include <time.h> int nanosleep(const struct timespec *rqtp, struct timespec *rmtp); struct timespec { time_t tv_sec; /* seconds */ long tv_nsec; /* and nanoseconds */ };</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>nanosleep() suspends the current thread from execution until either the time interval specified by <i>rqtp</i> 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 thread. 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. 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, CLOCK_REALTIME. nanosleep() will not block nor effect the action of any signal.</p>
RETURN VALUES	<p>If nanosleep() returns because the requested time has elapsed, it returns 0. Otherwise, if it returns because it has been interrupted by a signal:</p> <ul style="list-style-type: none"> it returns -1 and sets errno to indicate the interruption. If <i>rmtp</i> is non-NULL, the timespec structure referenced by <i>rmtp</i> will be updated to contain the remaining amount of time between <i>rqtp</i> and the time actually slept. <p>If any of the following error conditions occur, nanosleep() returns -1 and sets errno to indicate the error condition.</p>
ERRORS	<p>EINTR nanosleep() was interrupted by a signal.</p> <p>EINVAL <i>rqtp</i> specified a nanosecond value less than zero or greater than or equal to 1,000,000,000.</p> <p>ENOSYS nanosleep() is not supported by this implementation.</p>
SEE ALSO	sleep(3C)

NAME	ndbm, dbm_clearerr, dbm_close, dbm_delete, dbm_error, dbm_fetch, dbm_firstkey, dbm_nextkey, dbm_open, dbm_store – data base subroutines
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(char *file, int flags, int mode); int dbm_store(DBM *db, datum key, datum content, int flags);</pre>
MT-LEVEL	Unsafe
DESCRIPTION	<p>These functions maintain <i>key/content</i> pairs in a data base. The functions will handle very large (a billion blocks) data base 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 data base.</p> <p><i>keys</i> and <i>contents</i> are described by the datum typedef. A datum 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 .dir as its suffix. The second file contains all data and has .pag as its suffix.</p> <p>Before a data base can be accessed, it must be opened by dbm_open(). This will open and/or create the files <i>file.dir</i> and <i>file.pag</i> depending on the flags parameter (see open(2)). A data base is closed by calling dbm_close().</p> <p>Once open, the data stored under a key is accessed by dbm_fetch() and data is placed under a key by dbm_store(). The <i>flags</i> field can be either DBM_INSERT or DBM_REPLACE. DBM_INSERT will only insert new entries into the data base and will not change an existing entry with the same key. DBM_REPLACE will replace an existing entry if it has the same key. A key (and its associated contents) is deleted by dbm_delete(). A linear pass through all keys in a data base may be made, in an (apparently) random order, by use of dbm_firstkey() and dbm_nextkey(). dbm_firstkey() will return the first key in the data base. dbm_nextkey() will return the next key in the data base. This code will traverse the data base:</p> <pre>for (key = dbm_firstkey(db); key.dptr != NULL; key = dbm_nextkey(db))</pre> <p>dbm_error() returns non-zero when an error has occurred reading or writing the data base. dbm_clearerr() resets the error condition on the named data base.</p>

RETURN VALUES

All functions that return an **int** indicate errors with negative values. A return value of **0** indicates no error. Routines that return a **datum** indicate errors with a **NULL (0) *dptr***. If **dbm_store()** is called with a *flags* value of **DBM_INSERT** and finds an existing entry with the same key, it returns **1**.

EXAMPLES

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);
}
```

SEE ALSO

ar(1), cat(1), cp(1), tar(1), open(2), dbm(3B), netconfig(4)

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.

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. **dbm_store()** will return an error in the event that a disk block fills with inseparable data.

dbm_delete() does not physically reclaim file space, although it does make it available for reuse.

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	netdir, netdir_getbyname, netdir_getbyaddr, netdir_free, netdir_options, taddr2uaddr, uaddr2taddr, netdir_perror, netdir_sperror, netdir_mergeaddr – generic transport name-to-address translation
SYNOPSIS	<pre>#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 *netconfig, 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);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>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.</p> <p>The <code>netdir_getbyname()</code> routine maps the machine name and service name in the <code>nd_hostserv</code> structure to a collection of addresses of the type understood by the transport identified in the <code>netconfig</code> structure. This routine returns all addresses that are valid for that transport in the <code>nd_addrlist</code> structure. The <code>netconfig</code> structure is described on the <code>netconfig(4)</code> manual page. The <code>nd_hostserv</code> structure contains the following members:</p> <pre>char *h_host; /* host name */ char *h_serv; /* service name */</pre> <p>The <code>nd_addrlist</code> structure contains the following members:</p> <pre>int n_cnt; /* number of addresses */ struct netbuf *n_addrs;</pre> <p><code>netdir_getbyname()</code> accepts some special-case host names. The host names are defined in <code><netdir.h></code>. The currently defined host names are:</p> <p>HOST_SELF Represents the address to which local programs will bind their endpoints. <code>HOST_SELF</code> differs from the host name provided by <code>gethostname(3B)</code>, which represents the address to which <i>remote</i> programs will bind their endpoints.</p>

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 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:

```

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. *files* is the associated file descriptor. *option*, *files*, 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. *fildev* 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. *fildev* 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 **0.0.0.1.12**, 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.

SEE ALSO

gethostname(3B), getnetconfig(3N), getnetpath(3N), netconfig(4)

NAME	nice – change priority of a process
SYNOPSIS	<pre>/usr/ucb/cc [<i>flag</i> ...] <i>file</i> ... 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 fork(2). For a privileged process to return to normal priority from an unknown state, nice() 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, nice() returns 0. Otherwise, a value of –1 is returned and errno 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	nice(1) , renice(1B) , fork(2) , pricntl(2) , getpriority(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	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	<pre>cc [flag ...] file... -lnisdb -lnsl [library...] #include <rpcsvc/nis.h> #include <rpcsvc/nis_db.h> bool db_initialize(const char *dictionary_pathname); db_status db_create_table(const char *table_name, const table_obj *table); db_status db_destroy_table(const char *table_name); db_result *db_first_entry(const char *table_name, const int numattrs, const nis_attr *attrs); db_result *db_next_entry(const char *table_name, const db_next_desc *next_handle); db_result *db_reset_next_entry(const char *table_name, const db_next_desc *next_handle); db_result *db_list_entries(const char *table_name, const int numattrs, const nis_attr *attrs); db_result *db_remove_entry(const char *table_name, const int numattrs, const nis_attr *attrs); db_result *db_add_entry(const char *table_name, const int numattrs, const nis_attr *attrs, const entry_obj *entry); db_status db_table_exists(const char *table_name); db_status db_unload_table(const char *table_name); db_status db_checkpoint(const char *table_name); db_status db_standby(const char *table_name); void db_free_result(db_result *);</pre>
MT-LEVEL	Unsafe
DESCRIPTION	<p>These functions describe the interface between the NIS+ server and the underlying database. They are defined in the shared library /usr/lib/libnisdb.so.</p> <p>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.</p> <p>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+.</p>

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 {
    db_status      status;                /* Result status */
    db_next_desc  nextinfo;              /* descriptor */
    struct {
        u_int      objects_len;
        entry_obj  *objects_val;
    } objects;                            /* A variable list
                                        of objects */
    long          ticks;                  /* execution time in
                                        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 {
    char          *zattr_ndx;
    struct {
        u_int      zattr_val_len;
        char       *zattr_val_val;
    } 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).

SEE ALSO

rpc.nisd(1M), **nis_objects(3N)**, **nisfiles(4)**

NAME	nis_error, nis_sperrno, nis_perror, nis_terror, nis_sperror, nis_sperror_r – display NIS+ error messages
SYNOPSIS	<pre>cc [flag ...] file... -lnsl [library...] #include <rpcsvc/nis.h> char *nis_sperrno(const nis_error status); void nis_perror(const nis_error status, const char *label); void nis_terror(const nis_error status, const char *label); char *nis_sperror_r(nis_error status, char *label, char *buf); char *nis_sperror(const nis_error status, const char *label);</pre>
MT-LEVEL	Safe
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: error message</i>” on standard error.</p> <p>nis_terror() 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, buf, large enough to hold the error string (a buffer size of 128 bytes is guaranteed to be sufficiently large).</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>
SEE ALSO	niserror(1) , string(3C) , syslog(3)
NOTES	When compiling multithreaded applications, see Intro(3) , <i>Notes On Multithread Applications</i> , 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_map_group, __nis_map_group – NIS+ group manipulation functions
SYNOPSIS	<pre>cc [flag ...] file ... -lnsl [library...] #include <rpcsvc/nis.h> bool_t nis_ismember(const nis_name principal, const nis_name group); nis_error nis_addmember(const nis_name member, const nis_name group); nis_error nis_removemember(const nis_name member, const nis_name group); nis_error nis_creategroup(const nis_name group, const u_long flags); nis_error nis_destroygroup(const nis_name group); void nis_print_group_entry(const nis_name group); nis_error nis_verifygroup(const nis_name group);</pre>
MT-LEVEL	MT-Safe
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 <i>recursive</i> ("group") member, written "@cowards.oz.", refers to another group; all principals that belong to that group are considered to belong here. <p>Any member may be made <i>negative</i> by prefixing it with a minus sign ('-'). A group may thus contain explicit, implicit, recursive, negative explicit, negative implicit, and negative recursive members.</p> <p>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.</p> <p>The nis_ismember() function returns TRUE if it can establish that <i>principal</i> belongs to <i>group</i>; otherwise it returns FALSE.</p> <p>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.</p>

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

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).

Implicit Memberships

Given a group **baddies.oz.**, with members **wickedwitch.west.oz.** and ***.monkeys.west.oz.**, the function call

```
bool_var = nis_ismember("hulkhogan.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("hulkhogan.big.monkeys.west.oz.", "baddies.oz.");
```

will return 0 (FALSE).

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.

SEE ALSO

nisgrpadm(1), **nis_objects(3N)**

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:

```
u_long      gr_flags;          /* Interpretation Flags
                               (currently unused) */
struct {
    u_int     gr_members_len;
    nis_name  *gr_members_val;
} gr_members;                /* 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 (see **nis_admin(3N)**).

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	nisl_local_names, nisl_local_directory, nisl_local_host, nisl_local_group, nisl_local_principal – NIS+ local names
SYNOPSIS	cc [<i>flag</i> ...] <i>file</i> ... -lnsl [<i>library</i> ...] #include <rpcsvc/nis.h> nisl_name nisl_local_directory(void); nisl_name nisl_local_host(void); nisl_name nisl_local_group(void); nisl_name nisl_local_principal(void);
MT-LEVEL	MT-Safe
DESCRIPTION	<p>These functions return several default NIS+ names associated with the current process.</p> <p>nisl_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>nisl_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(3B) 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>nisl_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>nisl_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 nisladdcred(1M)) in the table named <i>cred.org_dir</i> in the default domain.</p> <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.</p>
ENVIRONMENT	NIS_GROUP This variable contains the name of the local NIS+ group. If the name is not fully qualified, the value returned by nisl_local_directory() will be concatenated to it.
SEE ALSO	nisldefaults(1) , nisladdcred(1M) , sysinfo(2) , gethostname(3B) , nisl_names(3N) , nisl_objects(3N)

NAME	nis_names, nis_lookup, nis_add, nis_remove, nis_modify, nis_freeresult – NIS+ namespace functions
SYNOPSIS	<pre>cc [flag ...] file ... -lnsl [library ...] #include <rpcsvc/nis.h> nis_result *nis_lookup(const nis_name name, const u_long flags); nis_result *nis_add(const nis_name name, const nis_object *obj); nis_result *nis_remove(const nis_name name, const nis_object *obj); nis_result *nis_modify(const nis_name name, const nis_object *obj); void nis_freeresult(nis_result *result);</pre>
MT-LEVEL	MT-Safe
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 nis_result 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 nis_result 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 <i>name</i>, and a flags parameter, <i>flags</i>, which is defined below. The object name is expected to correspond to the syntax of a non-indexed NIS+ name (see nis_tables(3N)). The nis_lookup() function is the <i>only</i> function from this group that can use a non-fully qualified name. If the parameter <i>name</i> is not a fully qualified name, then the flag EXPAND_NAME <i>must</i> be specified in the call. If this flag is not specified, the function will fail with the error NIS_BADNAME.</p> <p>The <i>flags</i> parameter is constructed by logically ORing zero or more flags from the following list.</p> <p>FOLLOW_LINKS 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</p>

	either a object is found that is not a <i>LINK</i> type object, or the library has followed 16 links.
HARD_LOOKUP	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.
NO_CACHE	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.
MASTER_ONLY	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.
EXPAND_NAME	When specified, the client library will attempt to expand a partially qualified name by calling the function nis_getnames() (see nis_subr(3N)) which uses the environment variable NIS_PATH .

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.

Note: 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 {
    nis_error status;
    struct {
        u_int objects_len;
        nis_object *objects_val;
    } objects;
    netobj      cookie;
    u_long      zticks;
    u_long      dticks;
    u_long      aticks;
    u_long      cticks;
};
```

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)**).

<i>zticks</i>	The time spent in the NIS+ service itself. This count starts when the server receives the request and stops when it sends the reply.
<i>dticks</i>	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.
<i>aticks</i>	The time spent in any "accelerators" or caches. This includes the time required to locate the server needed to resolve the request.
<i>cticks</i>	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 NO_CACHE when you call the lookup function.
NIS_NOTFOUND	The named object does not exist in the namespace.
NIS_CACHEEXPIRED	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 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_NAMEUNREACHABLE	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 HARD_LOOKUP flag.
NIS_UNKNOWNOBJ	The object returned is of an unknown type.
NIS_TRYAGAIN	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 nis_list() if the client specifies a callback and the server does not have enough resources to handle the callback.
NIS_SYSTEMERROR	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 syslog(3) message indicating why the RPC request failed.

ENVIRONMENT **NIS_PATH** If the flag **EXPAND_NAME** is set, this variable is the search path used by **nis_lookup()**.

SEE ALSO **nis_error(3N)**, **nis_objects(3N)**, **nis_server(3N)**, **nis_subr(3N)**, **nis_tables(3N)**

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	<code>cc [<i>flag</i> ...] <i>file</i> ... -lnsl [<i>library</i> ...] /usr/include/rpcsvc/nis_objects.x</code>
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 nisl_object structure contains the following members:</p> <pre> typedef char *nisl_name; struct nisl_object { nisl_oid zo_oid; nisl_name zo_name; nisl_name zo_owner; nisl_name zo_group; nisl_name zo_domain; u_long zo_access; u_long zo_ttl; objdata zo_data; }; </pre> <p>In this structure, the first member zo_oid, 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, zo_name, 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>zo_domain 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 zo_name and zo_oid, 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> <p>The zo_owner and zo_group members contain the NIS+ names of the object's principal owner and group owner, respectively. Both names <i>must be</i> 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.</p>

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 {
        u_int                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 {
    u_long    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 {
        u_int    ep_len;
        endpoint *ep_val;
    } ep;
    u_long    key_type;
    netobj    pkey;
}
typedef struct nis_server nis_server;

struct directory_obj {
    nis_name  do_name;
    nstype    do_type;
    struct {
        u_int    do_servers_len;
        nis_server *do_servers_val;
    } do_servers;
    u_long    do_ttl;
    struct {

```

```

        u_int      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 and **NIS_PK_DH** for a Diffie-Hellman type 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 {
        u_int    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 {
    u_long    gr_flags;
    struct {
        u_int    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

struct table_col {
    char      *tc_name;
    u_long   tc_flags;
    u_long   tc_rights;
}
typedef struct table_col table_col;

struct table_obj {
    char      *ta_type;
    u_int     ta_maxcol;
    u_char    ta_sep;
    struct {
        u_int     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 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.

Entry Objects

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 {
    u_long    ec_flags;
    struct {
        u_int    ec_value_len;
        char     *ec_value_val;
    } ec_value;
}
typedef struct entry_col entry_col;

struct entry_obj {
    char     *en_type;
    struct {
        u_int    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_admin(3N)**, **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... -lnsl [library...] #include <rpcsvc/nis.h> void nis_ping(const nis_name dirname, const u_long utime, const nis_object *dobj); nis_result *nis_checkpoint(const nis_name dirname);</pre>
MT-LEVEL	MT-Safe
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>dobj</i> is NULL, 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>
SEE ALSO	nislog(1M) , nis_names(3N) , nis_tables(3N) , nisfiles(4)

NAME	nls_server, nls_mkdir, nls_rmdir, nls_servstate, nls_stats, nls_getservlist, nls_freeservlist, nls_freetags – miscellaneous NIS+ functions
SYNOPSIS	<pre>cc [flag ...] file... -lnsl [library...] #include <rpcsvc/nis.h> nis_error nls_mkdir(const nis_name dirname, const nis_server *machine); nis_error nls_rmdir(const nis_name dirname, const nis_server *machine); nis_error nls_servstate(const nis_server *machine, const nis_tag *tags, const int numtags, nis_tag **result); nis_error nls_stats(const nis_server *machine, const nis_tag *tags, const int numtags, nis_tag **result); void nls_freetags(nis_tag *tags, const int numtags); nis_server **nls_getservlist(const nis_name dirname); void nls_freeservlist(nis_server **machines);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>These functions provide a variety of services for NIS+ applications.</p> <p>nls_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 nls_server structure, refer to nls_objects(3N).</p> <p>nls_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 nls_server structure, refer to nls_objects(3N).</p> <p>nls_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 nls_stats() function is used to retrieve statistics about how the server is operating. Tracking these statistics can help administrators determine when they need to add additional replicas or to break up a domain into two or more subdomains. For more information on reading statistics, see nlsstat(1M).</p> <p>nls_servstate() and nls_stats() use the tag list. This tag list is a variable length array of <i>nls_tag</i> structures whose length is passed to the function in the <i>numtags</i> parameter. The set of legal tags are defined in the file <rpcsvc/nls_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 <i>tag_value</i> member being set to the string “unknown.” Both of these functions return their results in malloced tag structure, <i>*result</i>. If there is an error, <i>*result</i> is set to NULL. The <i>tag_value</i> pointers points to allocated string memory which contains the results. Use nls_freetags() to free the tag structure.</p>

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.

SEE ALSO

nisstat(1M), **nis_names(3N)**, **nis_objects(3N)**, **nis_subr(3N)**

NAME	nls_subr, nls_leaf_of, nls_name_of, nls_domain_of, nls_getnames, nls_freenames, nls_dir_cmp, nls_clone_object, nls_destroy_object, nls_print_object – NIS+ subroutines
SYNOPSIS	<pre>cc [flag ...] file ... -lnsl [library ...] #include <rpcsvc/nis.h> nis_name nls_leaf_of(const nis_name name); nis_name nls_name_of(const nis_name name); nis_name nls_domain_of(const nis_name name); nis_name *nls_getnames(const nis_name name); void nls_freenames(nis_name *namelist); name_pos nls_dir_cmp(const nis_name n1, const nis_name n2); nis_object *nls_clone_object(const nis_object *src, nis_object *dest); void nls_destroy_object(nis_object *obj); void nls_print_object(const nis_object *obj);</pre>
MT-LEVEL	Safe
DESCRIPTION	<p>These subroutines are provided to assist in the development of NIS+ applications. They provide several useful operations on both NIS+ names and objects.</p> <p>The first group, nls_leaf_of(), nls_domain_of(), and nls_name_of() provide the functions for parsing NIS+ names. nls_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.</p> <p>nls_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 nls_leaf_of() and nls_domain_of() it is possible to break a NIS+ name into its individual components.</p> <p>nls_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 nls_name_of() were passed a name bob.friends.foo.bar.baz., then nls_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.</p> <p>nls_getnames() will return a list of candidate names for the name passed in as <i>name</i>. If this name is not fully qualified, nls_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 nls_freenames().</p>

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

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 '\$'.

SEE ALSO

nis_names(3N), **nis_objects(3N)**, **nis_tables(3N)**

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... -lnsl [library...] #include <rpcsvc/nis.h> nis_result *nis_list(const nis_name name, const u_long flags, int (*callback)(const nis_name table_name, const nis_object *object, const void *userdata), const void *userdata); nis_result *nis_add_entry(const nis_name table_name, const nis_object *object, const u_long flags); nis_result *nis_remove_entry(const nis_name name, const nis_object *object, const u_long flags); nis_result *nis_modify_entry(const nis_name name, const nis_object *object, const u_long flags); nis_result *nis_first_entry(const nis_name table_name); nis_result *nis_next_entry(const nis_name table_name, const netobj *cookie); void nis_freeresult(nis_result *result);</pre>
MT-LEVEL	MT-Safe
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 nis_result 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> <p style="text-align: center;">[colname=value, ...],tablename</p> <p>The list function, nis_list(), takes an indexed name as the value for the <i>name</i> parameter. Here, the tablename should be a fully qualified NIS+ name unless the EXPAND_NAME flag (described below) is set. The second parameter, <i>flags</i>, defines how the function will respond to various conditions. The value for this parameter is created by logically ORing together one or more flags from the following list.</p> <p>FOLLOW_LINKS</p> <p style="padding-left: 40px;">If the table specified in <i>name</i> 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.</p> <p>FOLLOW_PATH This flag specifies that if the entry is not found within this table, the list</p>

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.

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. 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 {
    nis_error  status;
    struct {
        u_int      objects_len;
        nis_object *objects_val;
    } objects;
    netobj      cookie;
    u_long      zticks;
    u_long      dticks;
    u_long      aticks;
    u_long      cticks;
};

```

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() . 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.
NIS_NOTMASTER	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 /var/nis/NIS_SHARED_DIRCACHE file will need to have their cache managers restarted (use nis_cachemgr -i) to flush this cache.
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_NOTSEARCHABLE	The table name resolved to a NIS+ object that was not searchable.
NIS_PARTIAL	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.
NIS_RPCERROR	This fatal error indicates the RPC subsystem failed in some way. Generally there will be a syslog(3) message indicating why the RPC request failed.
NIS_S_NOTFOUND	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.
NIS_S_SUCCESS	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.
	NIS_SUCCESS	The request was successful.
	NIS_SYSTEMERROR	Some form of generic system error occurred while attempting the request. Check the syslog(3) 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. add_entry() , remove_entry() , and modify_entry() return this error when the master server is currently updating its internal state. It can be returned to nis_list() 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	NIS_PATH	When set, this variable is the search path used by nis_list() if the flag EXPAND_NAME is set.
SEE ALSO	niscat(1) , niserror(1) , nismatch(1) , nis_cachemgr(1M) , nis_error(3N) , nis_local_names(3N) , nis_names(3N) , nis_objects(3N) , syslog(3)	
WARNINGS	Use the flag HARD_LOOKUP carefully since it can cause the application to block indefinitely during a network partition.	
NOTES	<p>The path used when the flag FOLLOW_PATH is specified, is the one present in the <i>first</i> table searched. The path values in tables that are subsequently searched are ignored.</p> <p>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.</p> <p>There are currently no known methods for nis_first_entry() and nis_next_entry() to get their answers from only the master server.</p>	

NAME	nl_langinfo – language information
SYNOPSIS	<pre>#include <nl_types.h> #include <langinfo.h> char *nl_langinfo(nl_item item);</pre>
MT-LEVEL	Safe with exceptions
DESCRIPTION	<p>nl_langinfo() 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>.</p> <p>Since yes and no strings are implemented using gettext(3I), link with the -lintl library.</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>
EXAMPLES	<p>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>
SEE ALSO	gettext(3I) , localeconv(3C) , setlocale(3C) , strftime(3C) , 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> <p>This function is built upon the functions localeconv(), strftime(), and gettext(). Where possible users are advised to use these interfaces to the required data instead of using calls to nl_langinfo().</p>
NOTES	<p>nl_langinfo can be used safely in a multi-thread application, as long as setlocale(3C) is not being called to change the locale.</p>

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 nlist 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 n_name 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 n_name 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 n_value and n_type fields. If a symbol cannot be located, the corresponding n_type 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 -1 and sets all of the n_type fields in members of the array pointed to by <i>nl</i> to zero.
SEE ALSO	nlist(3E) , a.out(4)
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 n_value field is compatibly set. Other fields in the nlist structure are filled with the ELF (Executable and Linking Format) values (see nlist(3e) and a.out(4)).</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>
MT-LEVEL	Safe
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 -g option to cc(1B).</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 n_name are set to 0.</p> <p>This function is useful for examining the system name list kept in the file /dev/ksyms. 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>
SEE ALSO	cc(1B) , elf(3E) , kvm_nlist(3K) , kvm_open(3K) , a.out(4) , ksyms(7) , mem(7)

NAME	nlsgetcall – get client's data passed via the listener
SYNOPSIS	<pre>#include <sys/tiuser.h> struct t_call *nlsgetcall(int fildes);</pre>
MT-LEVEL	Unsafe
DESCRIPTION	<p>nlsgetcall() allows server processes started by the listener process to access the client's t_call structure, that is, the <i>sndcall</i> argument of t_connect(3N).</p> <p>The t_call structure returned by nlsgetcall() can be released using t_free(3N).</p> <p>nlsgetcall() returns the address of an allocated t_call structure or NULL if a t_call structure cannot be allocated. If the t_alloc() succeeds, undefined environment variables are indicated by a negative <i>len</i> field in the appropriate netbuf structure. A <i>len</i> field of zero in the netbuf structure is valid and means that the original buffer in the listener's t_call structure was NULL.</p>
WARNING	<p>The <i>len</i> field in the netbuf 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>
RETURN VALUES	A NULL pointer is returned if a t_call structure cannot be allocated by t_alloc() . t_errno can be inspected for further error information. Undefined environment variables are indicated by a negative length field (<i>len</i>) in the appropriate netbuf structure.
FILES	<pre>/usr/lib/libnsl_s.a /usr/lib/libslan.a /usr/lib/libnls.a</pre>
SEE ALSO	nlsadmin(1M) , getenv(3C) , t_alloc(3N) , t_connect(3N) , t_error(3N) , t_free(3N) , t_sync(3N)
NOTES	<p>Server processes must call t_sync(3N) before calling this routine.</p> <p>This interface is unsafe in multithreaded applications. Unsafe interfaces should be called only from the main thread.</p>

NAME	nlsprovider – get name of transport provider
SYNOPSIS	char *nlsprovider(void);
MT-LEVEL	Unsafe
DESCRIPTION	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. The environment variable is only available to server processes started by the listener process.
RETURN VALUES	If the variable is not defined in the environment, a NULL pointer is returned.
FILES	/usr/lib/libslan.a (7300) /usr/lib/libnls.a (3B2 Computer) /usr/lib/libnsl_s.a
SEE ALSO	nlsadmin(1M)
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>fildes</i>, char *<i>service_code</i>); extern int _nlslog, t_errno; extern char *_nlsrmsg;</pre>
MT-LEVEL	Unsafe
DESCRIPTION	<p>Given a virtual circuit to a listener process (<i>fildes</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 _nlsrmsg. 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 t_errno. <p>Positive values are error return codes from the <i>listener</i> process. Mnemonics for these codes are defined in <listen.h>.</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, _nlsrmsg contains a pointer to a static, null terminated character buffer containing the <i>service request response message</i>. Note that both _nlsrmsg and the data buffer are overwritten by each call to nlsrequest().</p> <p>If _nlslog is non-zero, nlsrequest() prints error messages on stderr. Initially, _nlslog is zero.</p>
FILES	<pre>/usr/lib/libnls.a /usr/lib/libslan.a /usr/lib/libnsl_s.a</pre>
SEE ALSO	nlsadmin(1M) , t_error(3N)
WARNINGS	<p>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 via a T_LOOK error before or during the first t_snd() or t_rcv() call.</p>

NOTES

These interfaces are unsafe in multithreaded applications. Unsafe interfaces should be called only from the main thread.

NAME	offsetof – offset of structure member
SYNOPSIS	#include <stddef.h> size_t offsetof(<i>type</i> , <i>member-designator</i>);
MT-LEVEL	MT-Safe
DESCRIPTION	offsetof() is a macro defined in <stddef.h> which expands to an integral constant expression that has type size_t , the value of which is the offset in bytes, to the structure member (designated by <i>member-designator</i>), from the beginning of its structure (designated by <i>type</i>).

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>
MT-LEVEL	Unsafe
DESCRIPTION	<p>p2open() forks and execs a shell running the command line pointed to by <i>cmd</i>. On return, fp[0] points to a FILE pointer to write the command's standard input and fp[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	<pre>#include <stdio.h> #include <libgen.h> main(argc,argv) int argc; char **argv; { FILE *fp[2]; pid_t pid; char buf[16]; pid=p2open("/usr/bin/cat", fp); if (pid == -1) { fprintf(stderr, "p2open failed\n"); exit(1); } write(fileno(fp[0]),"This is a test\n", 16); if(read(fileno(fp[1]), buf, 16) <=0) fprintf(stderr, "p2open failed\n"); else write(1, buf, 16); (void)p2close(fp); }</pre>

SEE ALSO `fclose(3S)`, `popen(3S)`, `setbuf(3S)`

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	panel_above, panel_below – panels deck traversal primitives
SYNOPSIS	#include <panel.h> PANEL *panel_above(PANEL *panel); PANEL *panel_below(PANEL *panel);
MT-LEVEL	Unsafe
DESCRIPTION	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. 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.
RETURN VALUES	NULL is returned if an error occurs.
SEE ALSO	curses(3X) , panels(3X)
NOTES	These routines allow traversal of the deck of currently visible panels. The header < panel.h > automatically includes the header < curses.h >.

NAME	panel_move, move_panel – move a panels window on the virtual screen
SYNOPSIS	#include <panel.h> int move_panel(PANEL *panel, int starty, int startx);
MT-LEVEL	Unsafe
DESCRIPTION	move_panel() moves the curses window associated with <i>panel</i> so that its upper left-hand corner is at <i>starty</i> , <i>startx</i> . See usage note, below.
RETURN VALUES	OK is returned if the routine completes successfully, otherwise ERR is returned.
SEE ALSO	curses(3X) , panel_update(3X) , panels(3X)
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	#include <panel.h> PANEL *new_panel(WINDOW *win); int del_panel(PANEL * panel);
MT-LEVEL	Unsafe
DESCRIPTION	new_panel() creates a new panel associated with <i>win</i> and returns the panel pointer. The new panel is placed on top of the panel deck. del_panel() destroys <i>panel</i> , but not its associated window.
RETURN VALUES	new_panel() returns NULL if an error occurs. del_win() returns OK if successful, ERR otherwise.
SEE ALSO	curses(3X) , panel_update(3X) , panels(3X)
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>#include <panel.h> int show_panel(PANEL *panel); int hide_panel(PANEL *panel); int panel_hidden(PANEL *panel);</pre>
MT-LEVEL	Unsafe
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.
SEE ALSO	curses(3X) , panel_update(3X) , panels(3X)
NOTES	The header <panel.h> automatically includes the header <curses.h> .

NAME	panel_top, top_panel, bottom_panel – panels deck manipulation routines
SYNOPSIS	<pre>#include <panel.h> int top_panel(PANEL *panel); int bottom_panel(PANEL *panel);</pre>
MT-LEVEL	Unsafe
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.
SEE ALSO	curses(3X) , panel_update(3X) , panels(3X)
NOTES	The header <panel.h> automatically includes the header <curses.h> .

NAME	panel_update, update_panels – panels virtual screen refresh routine
SYNOPSIS	#include <panel.h> void update_panels(void);
MT-LEVEL	Unsafe
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.
SEE ALSO	curs_refresh(3X) , curses(3X) , panels(3X)
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	<pre>#include <panel.h> int set_panel_userptr(PANEL *panel, char *ptr); char * panel_userptr(PANEL *panel);</pre>
MT-LEVEL	Unsafe
DESCRIPTION	Each panel has a user pointer available for maintaining relevant information. set_panel_userptr() sets the user pointer of <i>panel</i> to <i>ptr</i> . panel_userptr() returns the user pointer of <i>panel</i> .
RETURN VALUES	set_panel_userptr returns OK if successful, ERR otherwise. panel_userptr returns NULL if there is no user pointer assigned to <i>panel</i> .
SEE ALSO	curses(3X) , panels(3X)
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	<pre>#include <panel.h> WINDOW *panel_window(PANEL *panel); int replace_panel(PANEL *panel, WINDOW *win);</pre>
MT-LEVEL	Unsafe
DESCRIPTION	<p>panel_window() returns a pointer to the window of <i>panel</i>.</p> <p>replace_panel() replaces the current window of <i>panel</i> with <i>win</i>.</p>
RETURN VALUES	<p>panel_window() returns NULL on failure.</p> <p>replace_panel() returns OK on successful completion, ERR otherwise.</p>
SEE ALSO	curses(3X), panels(3X)
NOTES	The header <panel.h> automatically includes the header <curses.h>.

NAME panels – character based panels package

SYNOPSIS `#include <panel.h>`

MT-LEVEL Unsafe

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.

SEE ALSO **curses**(3X), and 3X pages whose names begin “panel_” for detailed routine descriptions.

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 *path, const char *name, const char *mode);</pre>																										
MT-LEVEL	MT-Safe																										
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 [rwxfbcdpugks]:</p> <table border="0" style="margin-left: 40px;"> <thead> <tr> <th style="text-align: left;">Letter</th> <th style="text-align: left;">Meaning</th> </tr> </thead> <tbody> <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> </tbody> </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>To find the ls command using the PATH environment variable:</p> <pre>pathfind (getenv ("PATH"), "ls", "rx")</pre>																										

RETURN VALUES If no match is found, **pathname** returns a null pointer, **((char *) 0)**.

SEE ALSO **sh(1)**, **test(1)**, **access(2)**, **mknod(2)**, **stat(2)**, **getenv(3C)**

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	perror, errno – print system error messages
SYNOPSIS	<pre>#include <stdio.h> void perror(const char *s); #include <errno.h> int errno;</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>perror() 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 <i>s</i> is printed first, then a colon and a blank, then the message and a newline. (However, if <i>s</i> is a null pointer or points to a null string, the colon is not printed.) To be of most use, the argument string should include the name of the program that incurred the error. The error number is taken from the external variable errno, (see intro(2)), which is set when errors occur but not cleared when non-erroneous calls are made.</p>
SEE ALSO	intro(2) , fmtmsg(3C) , gettext(3I) , setlocale(3C) , strerror(3C)
NOTES	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) .

NAME	pfmt – display error message in standard format
SYNOPSIS	#include <pfmt.h> int pfmt(FILE *stream, long flags, char *format, ... /* arg */);
MT-LEVEL	MT-safe
DESCRIPTION	<p>pfmt() retrieves a format string from a locale-specific message database (unless MM_NOGET is specified) and uses it for printf() style formatting of <i>args</i>. The output is displayed on <i>stream</i>.</p> <p>pfmt() encapsulates the output in the standard error message format (unless MM_NOSTD is specified, in which case the output is simply printf() like).</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> <p style="padding-left: 40px;"><i><catalog>:<msgnum>:<defmsg></i>.</p> <p>If MM_NOGET is specified, only the <i><defmsg></i> part must be specified.</p> <p><i><catalog></i> is used to indicate the message database that contains the localized version of the format string. <i><catalog></i> 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><i><msgnum></i> 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() using the LC_ALL or LC_MESSAGES 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> part of the <i>format</i> 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(). In this case, the <i>format</i> argument has the following structure:</p> <p style="padding-left: 40px;"><i>:<msgnum>:<defmsg></i>.</p> <p>pfmt() will output Message not found!!\n as format string if <i><catalog></i> is not a valid catalog name, if no catalog is specified (either explicitly or via 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> <p>The <i>flags</i> determine the type of output (i.e. whether the <i>format</i> should be interpreted as is or encapsulated in the standard message format), and the access to message catalogs to retrieve a localized version of <i>format</i>.</p>

The *flags* are 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 a **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>* part 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** generates a localized version of **HALT**, but does not halt the machine.
- MM_ERROR** generates a localized version of **ERROR** (default, value 0).
- MM_WARNING** generates a localized version of **WARNING**.
- MM_INFO** generates 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()**. The numeric value ORed with other *flags* will generate the specified severity.

If the severity is not defined, **pfmt()** used 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** specifies an action message. Any severity value is superseded and replaced by a localized version of **TO FIX**.

STANDARD ERROR MESSAGE FORMAT

pfmt() displays error messages in the following format:
label: severity: text

If no *label* was defined by a call to **setlabel()**, 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: text

label: TO FIX: text

RETURN VALUE

Upon success, **pfmt()** returns the number of bytes transmitted. Upon failure, it returns a negative value:

-1 write error to *stream*.

EXAMPLES

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 ...
```

NOTES

pfmt() uses **gettext(3C)**, it is recommended that **pfmt()** not be used.

SEE ALSO

addsev(3C), **gettext(3C)**, **lfmt(3C)**, **printf(3S)**, **setcat(3C)**, **setlabel(3C)**, **setlocale(3C)**, **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	plock() 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. plock() performs the function specified by <i>op</i> : <table border="0" style="margin-left: 40px;"> <tr> <td>PROCLOCK</td> <td>Lock text and data segments into memory (process lock).</td> </tr> <tr> <td>TXTLOCK</td> <td>Lock text segment into memory (text lock).</td> </tr> <tr> <td>DATLOCK</td> <td>Lock data segment into memory (data lock).</td> </tr> <tr> <td>UNLOCK</td> <td>Remove locks.</td> </tr> </table>	PROCLOCK	Lock text and data segments into memory (process lock).	TXTLOCK	Lock text segment into memory (text lock).	DATLOCK	Lock data segment into memory (data lock).	UNLOCK	Remove locks.				
PROCLOCK	Lock text and data segments into memory (process lock).												
TXTLOCK	Lock text segment into memory (text lock).												
DATLOCK	Lock data segment into memory (data lock).												
UNLOCK	Remove locks.												
RETURN VALUES	Upon successful completion, a value of 0 is returned to the calling process. Otherwise, a value of -1 is returned and errno is set to indicate the error.												
ERRORS	plock() fails and does not perform the requested operation if one or more of the following are true: <table border="0" style="margin-left: 40px;"> <tr> <td>EAGAIN</td> <td>Not enough memory.</td> </tr> <tr> <td>EINVAL</td> <td><i>op</i> is equal to PROCLOCK and a process lock, a text lock, or a data lock already exists on the calling process.</td> </tr> <tr> <td>EINVAL</td> <td><i>op</i> is equal to TXTLOCK and a text lock, or a process lock already exists on the calling process.</td> </tr> <tr> <td>EINVAL</td> <td><i>op</i> is equal to DATLOCK and a data lock, or a process lock already exists on the calling process.</td> </tr> <tr> <td>EINVAL</td> <td><i>op</i> is equal to UNLOCK and no lock exists on the calling process.</td> </tr> <tr> <td>EPERM</td> <td>The effective user of the calling process is not super-user.</td> </tr> </table>	EAGAIN	Not enough memory.	EINVAL	<i>op</i> is equal to PROCLOCK and a process lock, a text lock, or a data lock already exists on the calling process.	EINVAL	<i>op</i> is equal to TXTLOCK and a text lock, or a process lock already exists on the calling process.	EINVAL	<i>op</i> is equal to DATLOCK and a data lock, or a process lock already exists on the calling process.	EINVAL	<i>op</i> is equal to UNLOCK and no lock exists on the calling process.	EPERM	The effective user of the calling process is not super-user.
EAGAIN	Not enough memory.												
EINVAL	<i>op</i> is equal to PROCLOCK and a process lock, a text lock, or a data lock already exists on the calling process.												
EINVAL	<i>op</i> is equal to TXTLOCK and a text lock, or a process lock already exists on the calling process.												
EINVAL	<i>op</i> is equal to DATLOCK and a data lock, or a process lock already exists on the calling process.												
EINVAL	<i>op</i> is equal to UNLOCK and no lock exists on the calling process.												
EPERM	The effective user of the calling process is not super-user.												
SEE ALSO	exec(2) , exit(2) , fork(2) , memcntl(2) , mlock(3C) , mlockall(3C)												
NOTES	mlock(3C) and mlockall(3C) are the preferred interfaces to process locking.												

NAME	popen, pclose – initiate pipe to/from a process
SYNOPSIS	<pre>#include <stdio.h> FILE *popen(const char *command, const char *type); int pclose (FILE *stream);</pre>
MT-LEVEL	Unsafe
DESCRIPTION	<p>popen() creates a pipe between the calling program and the command to be executed. The arguments to popen() are pointers to null-terminated strings. <i>command</i> consists of a shell command line. <i>type</i> is an I/O mode, either r for reading or w 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 w, 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 r, by reading from the file <i>stream</i>. Because open files are shared, a type r command may be used as an input filter and a type w as an output filter.</p> <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 details on termination status.</p>
RETURN VALUES	<p>popen() returns a null pointer if files or processes cannot be created.</p> <p>pclose() returns the termination status of the command. pclose() returns -1 if <i>stream</i> is not associated with a popen() command and sets errno to indicate the error.</p>
EXAMPLES	<p>Here is an example of a typical call:</p> <pre>#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; }</pre> <p>This program will print on the standard output (see stdio(3S)) all the file names in the current directory that have a .c suffix.</p>

SEE ALSO `pipe(2)`, `wait(2)`, `waitpid(2)`, `fclose(3S)`, `fopen(3S)`, `stdio(3S)`, `system(3S)`, `wstat(5)`

NOTES 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").

NAME	printf, fprintf, sprintf, vprintf, fprintf, vsprintf – formatted output conversion
SYNOPSIS	<pre> /usr/ucb/cc [flag ...] file ... #include <stdio.h> int printf(format, ...) const char *format; int fprintf(stream, format, va_list) FILE *stream; char *format; va_dcl; char *sprintf(s, format, va_list) char *s, *format; va_dcl; int vprintf(format, ap) char *format; va_list ap; int fprintf(stream, format, ap) FILE *stream; char *format; va_list ap; char *vsprintf(s, format, ap) char *s, *format; va_list ap; </pre>
DESCRIPTION	<p>printf() places output on the standard output stream stdout. fprintf() places output on the named output <i>stream</i>. 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.</p> <p>vprintf(), fprintf(), 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).</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 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 <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 simply ignored.</p> <p>Each conversion specification is introduced by the character %. After the %, the following appear in sequence:</p>

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 *args* 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.ddde±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()`, `fprint()`, `vprintf()`, and `vfprintf()` return EOF.

EXAMPLES

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);
```

To print π to 5 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.

Very wide fields (>128 characters) fail.

NAME	printf, fprintf, sprintf – print formatted output
SYNOPSIS	<pre>#include <stdio.h> int printf(const char *format, /* args */ ...); int fprintf(FILE *strm, const char *format, /* args */ ...); int sprintf(char *s, const char *format, /* args */ ...);</pre>
MT-LEVEL	See the NOTES section of this page.
DESCRIPTION	<p>printf() places output on the standard output stream stdout.</p> <p>fprintf() places output on <i>strm</i>.</p> <p>sprintf() places output, followed by the null character (\0), in consecutive bytes starting at <i>s</i>. It is the user's responsibility to ensure that enough storage is available. Each function returns the number of characters transmitted (not including the \0 in the case of sprintf()) or a negative value if an output error was encountered.</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 that contains three types of objects defined below:</p> <ol style="list-style-type: none"> 1. plain characters that are simply copied to the output stream; 2. escape sequences that represent non-graphic characters; 3. conversion specifications. <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>\n 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> <p>All forms of the printf() functions allow for the insertion of a language-dependent decimal-point character. The decimal-point character is defined by the program's locale (category LC_NUMERIC). In the C locale, or in a locale where the decimal-point character is not defined, the decimal-point character defaults to a period (.).</p>

Each conversion specification is introduced by the character %. After the character %, the following appear in sequence:

An optional field, consisting of a decimal digit string followed by a \$, specifying the next *args* to be converted. If this field is not provided, the *args* following the last *args* converted will be used.

Zero or more *flags*, which modify the meaning of the conversion specification.

An optional string of decimal digits to specify 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. If the format is %s or %ws (wide-character string), then the field width should be interpreted as the minimum columns of screen display. E.g. %10s means if the converted value has a screen width of 7 columns, then 3 spaces would be padded on the right.

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 decimal-point 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 characters to be printed from a string in **s** or **ws** conversions. 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 specifier applies to a **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 specifier applies to a pointer to a **short int** argument. An optional **l** (ell) specifies that a following **d**, **i**, **o**, **u**, **x**, or **X** conversion specifier applies to a **long int** or **unsigned long int** argument; an optional **l** (ell) specifies that a following **n** conversion specifier applies to a pointer to **long int** argument. An optional **L** specifies that a following **e**, **E**, **f**, **g**, or **G** conversion specifier applies to a **long double** argument. If an **h**, **l**, or **L** appears before any other conversion specifier, the behavior is undefined.

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.

In format strings containing the **digits\$* form of a conversion specification, a field width or precision may also be indicated by the sequence **digits\$*, giving the position in the argument list of an integer *args* containing the field width or precision.

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, be specified in the format string.

The *flag* characters and their meanings are:

- 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 space and + flags both appear, the space flag will be ignored.
- # The value is to be converted to an alternate form. For **c**, **d**, **i**, **s**, and **u** conversions, the flag has no effect. For an **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**) prepended to it. For **e**, **E**, **f**, **g**, and **G** conversions, the result will always contain a decimal-point 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 **g** and **G** conversions, trailing zeros will not be removed from the result as they normally are.
- 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. For other conversions, the behavior is undefined.

Each conversion character results in fetching zero or more *args*. The results are undefined if there are insufficient *args* for the format. If the format is exhausted while *args* remain, the excess *args* are ignored.

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**). The **x** conversion uses the letters **abcdef** and the **X** conversion uses the letters **ABCDEF**. The precision specifies the minimum number of digits to appear. If the value being converted can be represented in fewer digits than the specified minimum, it will be expanded with leading zeros. The default precision is 1. The result of converting a zero value with a precision of zero is no characters.
- f** The double *args* is converted to decimal notation in the style **[-]ddd.ddd**, where the number of digits after the decimal-point 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 decimal-point character appears. If a

	decimal-point character appears, at least 1 digit appears before it. The value is rounded to the appropriate number of digits.
e,E	The double <i>args</i> is converted to the style <code>[-]d.ddde±dd</code> , where there is one digit before the decimal-point 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 decimal-point 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 <i>args</i> 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 decimal-point character appears only if it is followed by a digit.
c	The int <i>args</i> is converted to an unsigned char , and the resulting character is printed.
wc	The int <i>args</i> is converted to a wide character (wchar_t), and the resulting wide character is printed.
s	The <i>args</i> is taken to be a string (character pointer) and characters from the string are written up to (but not including) a terminating null character; if the precision is specified, no more than that many characters are written. If the precision is not specified, it is taken to be infinite, so all characters up to the first null character are printed. A null value for <i>args</i> will yield undefined results.
ws	The <i>args</i> 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, no more than that many wide characters are 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 <i>args</i> will yield undefined results.
p	The <i>args</i> should be a pointer to void . The value of the pointer is converted to an implementation-defined 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 function.
n	The argument should be a pointer to an integer into which is written the number of characters 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.

If the character after the % or *%digits\$* sequence is not a valid conversion character, the results of the conversion are undefined.

If a floating-point value is the internal representation for infinity, the output is $[\pm]Infinity$, 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 $[\pm]NaN$. 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()** routine had been called.

LC_NUMERIC

Determines how numeric formats are handled. In the "C" locale, numeric handling follows the U.S. rules.

RETURN VALUES

printf(), **fprintf()**, and **sprintf()** return the number of characters transmitted, or return a negative value if an error was encountered.

EXAMPLES

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);
```

To print π to 5 decimal places:

```
printf("pi = %.5f", 4 * atan(1.0));
```

To print a list of names in columns which are 20 characters wide:

```
printf("%20s%20s%20s", lastname, firstname, middlename );
```

FILES

/usr/lib/locale/locale/LC_NUMERIC/numeric
LC_NUMERIC database for *locale*

SEE ALSO

exit(2), **lseek(2)**, **write(2)**, **abort(3C)**, **ecvt(3C)**, **putc(3S)**, **scanf(3S)**, **setlocale(3C)**, **stdio(3S)**

NOTES

printf() and **fprintf()** are Async-Safe in mutli-thread applications. **sprintf()** is MT-Safe in multi-thread applications. qq.BR **printf** and **fprintf** can be used safely in a multi-thread application, as long as **setlocale(3C)** is not being called to change the locale.

NAME	psignal, sys_siglist – system signal messages
SYNOPSIS	<pre>/usr/ucb/cc [<i>flag ...</i>] <i>file ...</i> 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 <signal.h>.</p> <p>To simplify variant formatting of signal names, the vector of message strings sys_siglist is provided; the signal number can be used as an index in this table to get the signal name without the newline. The define NSIG defined in <signal.h> 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	perror(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.

NAME	psignal, psigninfo – system signal messages
SYNOPSIS	<pre>#include <siginfo.h> void psignal(int sig, const char *s); void psigninfo(siginfo_t *pinfo, char *s);</pre>
MT-LEVEL	Safe
DESCRIPTION	<p>psignal() and psigninfo() produce messages on the standard error output describing a signal. <i>sig</i> is a signal that may have been passed as the first argument to a signal handler. <i>pinfo</i> is a pointer to a siginfo structure that may have been passed as the second argument to an enhanced signal handler (see sigaction(2)). The argument string <i>s</i> is printed first, then a colon and a blank, then the message and a newline.</p>
SEE ALSO	sigaction(2) , gettext(3I) , perror(3C) , setlocale(3C) , siginfo(5) , signal(5)
NOTES	If the application is linked with -lintl , then messages printed from these functions are in the native language specified by the LC_MESSAGES locale category; see setlocale(3C) .

NAME	ptsname – get name of the slave pseudo-terminal device
SYNOPSIS	<pre>#include <stdio.h> char *ptsname(int fildes);</pre>
MT-LEVEL	Safe
DESCRIPTION	The ptsname() function returns the name of the slave pseudo-terminal device associated with a master pseudo-terminal device. <i>fildes</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 /dev/pts/N , where N 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 NULL . This could occur if <i>fildes</i> is an invalid file descriptor or if the slave device name does not exist in the file system.
SEE ALSO	open(2) , grantpt(3C) , ttyname(3C) , unlockpt(3C) <i>STREAMS Programmer's Guide</i>

NAME	putc, putc_unlocked, putchar, putchar_unlocked, fputc, putw – put character or word on a stream
SYNOPSIS	<pre>#include <stdio.h> int putc(int c, FILE *stream); int putc_unlocked(int c, FILE *stream); int putchar(int c); int putchar_unlocked(int c); int fputc(int c, FILE *stream); int putw(int w, FILE *stream);</pre>
MT-LEVEL	See the NOTES section of this page.
DESCRIPTION	<p>putc() writes <i>c</i> (converted to an unsigned char) onto the output <i>stream</i> (see intro(3)) at the position where the file pointer (if defined) is pointing, and advances the file pointer appropriately. If the file cannot support positioning requests, or <i>stream</i> was opened with append mode, the character is appended to the output <i>stream</i>. putchar(c) is defined as putc(c, stdout). putc() and putchar() are macros.</p> <p>putc_unlocked() and putchar_unlocked() are respectively variants of putc() and putchar() that do not lock the stream. It is the caller's responsibility to acquire the stream lock before calling these functions and releasing the lock afterwards; see flockfile(3S) and stdio(3S).</p> <p>fputc() behaves like putc(), but is a function rather than a macro. fputc() runs more slowly than putc(), but it takes less space per invocation and its name can be passed as an argument to a function.</p> <p>putw() writes the C int (word) <i>w</i> to the standard I/O output <i>stream</i> (at the position of the file pointer, if defined). The size of a word is the size of an integer and varies from machine to machine. putw() neither assumes nor causes special alignment in the file.</p>
RETURN VALUES	<p>On success, putc(), fputc(), and putchar() return the value that was written. On error, those functions return the constant EOF. putw() returns ferror(stream), so that it returns 0 on success and 1 on failure.</p> <p>Failure will occur, for example, if the file <i>stream</i> is not open for writing or if the output file cannot grow.</p>
SEE ALSO	write(2) , intro(3) , fclose(3S) , ferror(3S) , flockfile(3S) , fopen(3S) , printf(3S) , puts(3S) , setbuf(3S) , stdio(3S)
NOTES	Because it is implemented as a macro, putc() evaluates a <i>stream</i> argument more than once. In particular, putc(c, *f++) ; does not work sensibly. fputc() should be used instead.

Because of possible differences in word length and byte ordering, files written using **putw()** are machine-dependent, and may not be read using **getw()** on a different processor.

Functions exist for all the above defined macros. To get the function form, the macro name must be undefined (for example, **#undef putc**).

fputc(), **putc()**, **putchar()**, and **putw()** are MT-Safe in multi-thread applications. **putc_unlocked()** and **putchar_unlocked()** are unsafe in multi-thread applications.

NAME	putenv – change or add value to environment
SYNOPSIS	#include <stdlib.h> int putenv(char *string);
MT-LEVEL	Safe
DESCRIPTION	putenv() 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. <i>string</i> 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() .
RETURN VALUES	putenv() returns non-zero if it was unable to obtain enough space using malloc() for an expanded environment, otherwise zero is returned.
SEE ALSO	exec(2) , getenv(3C) , malloc(3C) , environ(5)
NOTES	This routine uses malloc(3C) to enlarge the environment. After putenv() is called, environment variables are not in alphabetical order. <i>string</i> should not be an automatic variable. <i>string</i> should be declared static if it is declared within a function because it cannot be automatically declared. A potential error is to call the function putenv() with a pointer to an automatic variable as the argument and to then exit the calling function while <i>string</i> is still part of the environment. putenv() can be safely called from a multi-thread program. However, care must still be taken when using putenv() and getenv(3C) in a multi-thread program. These routines examine and modify the environment list. This list is shared by all threads in a program. The system prevents the list from being accessed simultaneously by two different threads. However, it does not prevent two threads from successively accessing the environment list using putenv() or getenv(3C) .

NAME	putpwent – write password file entry
SYNOPSIS	#include <pwd.h> int putpwent(const struct passwd *p, FILE *f);
MT-LEVEL	Unsafe
DESCRIPTION	putpwent() is the inverse of getpwent() , (see getpwnam(3C)). Given a pointer to a passwd structure created by getpwent() (or getpwuid() or getpwnam()), putpwent() writes a line on the stream <i>f</i> , which matches the format of <i>/etc/passwd</i> .
RETURN VALUES	putpwent() returns non-zero if an error was detected during its operation, otherwise zero.
SEE ALSO	getpwnam(3C) , putspent(3C)
NOTES	Do not use without also using putspent() to update the shadow file. The use of this function is discouraged.
BUGS	This routine is of limited utility, since most password files are maintained as Network Information Service (NIS) files, and cannot be updated with this routine.

NAME	puts, fputs – put a string on a stream
SYNOPSIS	#include <stdio.h> int puts(const char *s); int fputs(const char *s, FILE *stream);
MT-LEVEL	MT-Safe
DESCRIPTION	puts() writes the string pointed to by <i>s</i> , followed by a new-line character, to the standard output stream stdout (see intro(3)). fputs() writes the null-terminated string pointed to by <i>s</i> to the named output <i>stream</i> . Neither function writes the terminating null character.
RETURN VALUES	On success both routines return the number of characters written; otherwise they return EOF.
SEE ALSO	write(2) , intro(3) , fclose(3S) , ferror(3S) , fopen(3S) , printf(3S) , putc(3S) , stdio(3S)
NOTES	puts() appends a new-line character while fputs() does not.

NAME	putspent – write shadow password file entry
SYNOPSIS	<pre>#include <shadow.h> int putsent(const struct spwd *p, FILE *fp);</pre>
MT-LEVEL	Unsafe
DESCRIPTION	<p>The putsent() routine is the inverse of getsent(). Given a pointer to a spwd structure created by the getsent() routine (or the getspnam() routine), the putsent() routine writes a line on the stream <i>fp</i>, which matches the format of /etc/shadow. The spwd 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 sp_min, sp_max, sp_lstchg, sp_warn, sp_inact, or sp_expire field of the spwd structure is -1, or if sp_flag is 0, the corresponding /etc/shadow field is cleared.</p>
RETURN VALUES	The putsent() routine returns non-zero if an error was detected during its operation, otherwise zero.
SEE ALSO	getpwnam(3C) , getspnam(3C) , putpwent(3C)
NOTES	<p>This routine is for internal use only, compatibility is not guaranteed.</p> <p>Do not use without also using putpwent() to update the password file.</p> <p>The use of this function is discouraged.</p>

NAME	putwc, putwchar, fputwc – convert Process Code character to EUC and put on a stream
SYNOPSIS	<pre>cc [flag ...] file ... -lw [library ...] #include <stdio.h> #include <wchar.h> wchar_t putwc(int c, FILE *stream); wchar_t putwchar(int c); wchar_t fputwc(int c, FILE *stream);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>putwc() and fputwc() convert the Process Code (<i>wchar_t</i>) character <i>c</i> to Extended Unix Code (EUC) and write it onto the named output <i>stream</i> (at the position where the file pointer, if defined, is pointing). It returns the character written.</p> <p>putwchar(c) is defined as fputwc(c,stdout). putwc() and putwchar() are macros. See stdio(3S) for a discussion of output streams.</p>
RETURN VALUES	On success, these functions each return the value passed. On error, these functions return the constant EOF .
SEE ALSO	fclose(3S) , ferror(3S) , fopen(3S) , fread(3S) , getwc(3I) , printf(3S) , putc(3S) , putws(3I) , setbuf(3S) , stdio(3S)

NAME	putws, fputws – convert a string of Process Code characters to EUC characters and put it on a stream
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lw [<i>library</i> ...] #include <stdio.h> #include <wdec.h> int putws(wchar_t *s); int fputws(wchar_t *s, FILE *stream);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>putws() converts the Process Code string (terminated by a <i>wchar_t</i>NULL) pointed to by <i>s</i>, to an Extended Unix Code (EUC) string followed by a NEWLINE character, and writes it to the standard output stream stdout.</p> <p>fputws() writes the <i>wchar_t</i>NULL-terminated string pointed to by <i>s</i> to the named output stream, and does <i>not</i> append a NEWLINE.</p> <p>Neither function writes the terminal NULL character.</p>
RETURN VALUES	Both routines return the number of Process Code characters transformed and written. Both routines return EOF on error. This will happen if the routines try to write on a file that has not been opened for writing.
SEE ALSO	ferror(3S), fopen(3S), fread(3S), getws(3I), printf(3S), putwc(3I)

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>
MT-LEVEL	Safe
DESCRIPTION	<p>qsort() 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><i>base</i> points to the element at the base of the table. <i>nel</i> is the number of elements in the table. <i>width</i> specifies the size of each element in bytes. <i>compar</i> is the name of the comparison function, which is called with two arguments that point to the elements being compared. 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>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;</pre>

```
qsort((char *) a, 10, sizeof(int), intcompare);  
for (i=0; i<10; i++) printf(" %d",a[i]);  
printf("\n");  
}
```

SEE ALSO [sort\(1\)](#), [bsearch\(3C\)](#), [lsearch\(3C\)](#), [string\(3C\)](#)

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	#include <signal.h> int raise(int sig);
MT-LEVEL	MT-Safe
DESCRIPTION	raise() sends the signal <i>sig</i> to the executing program. raise() uses kill() to send the signal to the executing program: kill(getpid(), sig); See kill(2) for a detailed list of failure conditions. See signal(3C) for a list of signals.
RETURN VALUES	raise() returns zero if the operation succeeds. Otherwise, raise() returns -1 and errno is set to indicate the error.
SEE ALSO	getpid(2) , kill(2) , signal(3C)

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(3B)
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(3B) provide much better, though more elaborate, random-number generators. 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>
MT-LEVEL	See the NOTES section of this page.
DESCRIPTION	<p>rand() uses a multiplicative congruential random-number generator with period 2^{32} that returns successive pseudo-random numbers in the range from 0 to RAND_MAX (defined in <stdlib.h>).</p> <p>The function srand() uses the argument <i>seed</i> as a seed for a new sequence of pseudo-random numbers to be returned by subsequent calls to the function rand(). If the function srand() is then called with the same <i>seed</i> value, the sequence of pseudo-random numbers will be repeated. If the function 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>rand_r() 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>
SEE ALSO	drand48(3C)
NOTES	<p>The rand_r() 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> <p>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.</p> <p>The spectral properties of rand() are limited. drand48(3C) provides a much better, though more elaborate, random-number generator.</p> <p>rand() is unsafe in multi-thread applications. rand_r() is MT-Safe, and should be used instead. srand() is unsafe in multi-thread applications.</p>

NAME	random, srandom, initstate, setstate – better random number generator; routines for changing generators
SYNOPSIS	<pre> /usr/ucb/cc [flag ...] file ... long random() int srandom(seed) unsigned seed; char *initstate(seed, state, n) unsigned seed; char *state; int n; char *setstate(state) char *state; </pre>
DESCRIPTION	<p>random() uses a non-linear additive feedback random number generator employing a default table of size 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 very large, approximately $16 \times (2^{31}-1)$.</p> <p>random() and srandom() 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. Like rand(3C), however, random() will, by default, produce a sequence of numbers that can be duplicated by calling srandom() with 1 as the seed.</p> <p>The initstate() routine allows a state array, passed in as an argument, to be initialized for future use. <i>n</i> specifies the size of <i>state</i> in bytes. initstate() uses <i>n</i> to decide how sophisticated a random number generator it should use—the more state, the better the random numbers will be. Current “optimal” values for the amount of state information are 32, 64, 128, and 256 bytes. If the amount of state information is less than 32 bytes, a simple linear congruential random number generator is used. Using less than 8 bytes causes an error. The seed for the initialization (which specifies a starting point for the random number sequence, and provides for restarting at the same point) is also an argument. initstate() returns a pointer to the previous state information array.</p> <p>Once a state has been initialized, the setstate() routine provides for rapid switching between states. setstate() returns a pointer to the previous state array; its argument state array is used for further random number generation until the next call to initstate() or setstate().</p>

Once a state array has been initialized, it may be restarted at a different point either by calling `initstate()` (with the desired seed, the state array, and its size) or by calling both `setstate()` (with the state array) and `srandom()` (with the desired seed). The advantage of calling both `setstate()` and `srandom()` is that the size of the state array does not have to be remembered after it is initialized.

With 256 bytes of state information, the period of the random number generator is greater than 2^{69} , which should be sufficient for most purposes.

RETURN VALUES

If `initstate()` is called with less than 8 bytes of state information, or if `setstate()` detects that the state information has been garbled, error messages are printed on the standard error output.

EXAMPLES

```
/* 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("%d0,random());
}
```

SEE ALSO

`drand48(3C)`, `rand(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.

`random()` and `srandom()` function 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 [flag ...] file ... -lsocket -lnsl [library ...]</pre> <pre>int rcmd(char **ahost, unsigned short inport, const char *luser, const char *ruser, const char *cmd, int *fd2p);</pre> <pre>int rresvport(int *port);</pre> <pre>int ruserok(const char *rhost, int suser, const char *ruser, const char *luser);</pre>
MT-LEVEL	Unsafe
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 SOCK_STREAM 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 process, although you may be able to get its attention by using out-of-band data.</p> <p>The protocol is described in detail in in.rshd(1M).</p> <p>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 <i>port</i>, 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 <i>port</i> until either it succeeds or port number 512 is reached.</p> <p>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</p>

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

SEE ALSO

rlogin(1), **rsh(1)**, **in.rexecd(1M)**, **in.rshd(1M)**, **intro(2)**, **gethostbyname(3N)**, **rexec(3N)**

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	read_vtoc, write_vtoc – read and write a disk's VTOC								
SYNOPSIS	<pre>#include <sys/vtoc.h> cc [flag ...] -ladm file ... int read_vtoc(int fd, struct vtoc *vtoc); int write_vtoc(int fd, struct vtoc *vtoc);</pre>								
MT-LEVEL	Unsafe								
DESCRIPTION	<p>read_vtoc() returns the VTOC 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 at 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	<p>read_vtoc returns:</p> <table border="0"> <tr> <td style="padding-left: 2em;">positive number</td> <td>Success. The positive number is the slice index associated with the open file descriptor.</td> </tr> <tr> <td style="padding-left: 2em;">negative number</td> <td>There are two possible error returns. VT_EIO indicates an I/O error occurred and VT_ERROR indicates an unknown error.</td> </tr> </table> <p>write_vtoc returns:</p> <table border="0"> <tr> <td style="padding-left: 2em;">0</td> <td>Success</td> </tr> <tr> <td style="padding-left: 2em;">negative number</td> <td>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.</td> </tr> </table>	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.	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.
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.								
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.								
SEE ALSO	format(1M) , fmthard(1M) , prtvtoc(1M) , ioctl(2) , dkio(7)								
BUGS	write_vtoc cannot write a VTOC on an unlabeled disk. Use format(1M) for this purpose.								

NAME	realpath – returns the real file name
SYNOPSIS	<pre>#include <stdlib.h> #include <sys/param.h> char *realpath(char *file_name, char *resolved_name);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>realpath() resolves all links and references to “.” and “..” in <i>file_name</i> and stores it in <i>resolved_name</i>.</p> <p>It 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, <i>../relative_dir</i>), it returns the <i>resolved absolute</i> name. For the other relative path names, it returns the <i>resolved relative</i> name.</p> <p><i>resolved_name</i> must be big enough (MAXPATHLEN) to contain the fully resolved path name.</p>
RETURN VALUES	If there is no error, realpath() returns a pointer to the <i>resolved_name</i> . Otherwise it returns a null pointer and places the name of the offending file in <i>resolved_name</i> . The global variable errno is set to indicate the error.
SEE ALSO	getcwd(3C) , sysconf(3C)
NOTES	<p>realpath() operates on null-terminated strings.</p> <p>One should have execute permission on all the directories in the given and the resolved path.</p> <p>realpath() may fail to return to the current directory if an error occurs.</p>

NAME	reboot – reboot system or halt processor
SYNOPSIS	<pre> /usr/ucb/cc [flag ...] file ... #include <sys/reboot.h> int reboot (howto, bootargs) int howto; char *bootargs; </pre>
DESCRIPTION	<p>reboot() reboots the system. <i>howto</i> is a mask of options passed to the bootstrap program. The function interface permits only RB_HALT or RB_AUTOBOOT to be passed to the reboot program; the other flags are used in scripts stored on the console storage media, or used in manual bootstrap procedures. When none of these options (for instance RB_AUTOBOOT) is given, the system is rebooted from file /kernel/unix. An automatic consistency check of the disks is then normally performed.</p> <p>The bits of <i>howto</i> that are used are:</p> <p>RB_HALT the processor is simply halted; no reboot takes place. RB_HALT should be used with caution.</p> <p>RB_ASKNAME Interpreted by the bootstrap program itself, causing it to inquire as to what file should be booted. Normally, the system is booted from the file /kernel/unix without asking.</p> <p><i>bootargs</i> is an obsolete pointer that is now ignored, even though reboot() still requires a second argument of type char *.</p>
RETURN VALUES	If successful, this call never returns. Otherwise, a -1 is returned and an error is returned in the global variable errno .
ERRORS	EPERM The caller is not the super-user.
FILES	/kernel/unix system namelist
SEE ALSO	intro(1M) , crash(1M) , halt(1M) , init(1M) , reboot(1M) , uadmin(2)
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>Any other <i>howto</i> argument causes /kernel/unix to boot.</p> <p>Only the super-user may reboot() a machine.</p> <p><i>bootargs</i> is ignored in SunOS 5.X releases.</p>

NAME	recv, recvfrom, recvmsg – receive a message from a socket
SYNOPSIS	<pre>cc [<i>flag ...</i>] <i>file ...</i> -lsocket -lnsl [<i>library ...</i>] #include <sys/types.h> #include <sys/socket.h> #include <sys/uio.h> int recv(int <i>s</i>, char *<i>buf</i>, int <i>len</i>, int <i>flags</i>); int recvfrom(int <i>s</i>, char *<i>buf</i>, int <i>len</i>, int <i>flags</i>, struct sockaddr *<i>from</i>, int *<i>fromlen</i>); int recvmsg(int <i>s</i>, struct msghdr *<i>msg</i>, int <i>flags</i>);</pre>
MT-LEVEL	Safe
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> <p>MSG_OOB Read any “out-of-band” data present on the socket rather than the regular “in-band” data.</p> <p>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.</p> <p>The recvmsg() call uses a msghdr structure to minimize the number of directly supplied parameters. This structure is defined in <sys/socket.h> and includes the following members:</p> <pre> 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;</pre>

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	<i>s</i> 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	<i>s</i> is not a socket.
ESTALE	A stale NFS file handle exists.
EWOULDBLOCK	The socket is marked non-blocking and the requested operation would block.

SEE ALSO

fcntl(2), **ioctl(2)**, **read(2)**, **connect(3N)**, **getsockopt(3N)**, **send(3N)**, **socket(3N)**

NAME	regcmp, regex – compile and execute regular expression
SYNOPSIS	<pre>cc [flag ...] file ... -lgen [library ...] #include <libgen.h> char *regcmp(const char *string1, /* char *string2 */ ..., int /*(char *)0*/); char *regex(const char *re, const char *subject, /* char *ret0 */ ...); extern char *__loc1;</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>regcmp() compiles a regular expression (consisting of the concatenated arguments) and returns a pointer to the compiled form. malloc(3C) 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>regex() executes a compiled pattern against the subject string. Additional arguments are passed to receive values back. regex() returns NULL on failure or a pointer to the next unmatched character on success. A global character pointer __loc1 points to where the match began. regcmp() and regex() 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>[] * . ^ These symbols retain their meaning in ed(1).</p> <p>\$ Matches the end of the string; \n 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>{m} {m,} {m,u} 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, {m}), it indicates the exact number of times the regular expression is to be applied. The value {m,} is analogous to {m, infinity}. The plus (+) and star (*) operations are equivalent to {1,} and {0,} respectively.</p> <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. regex() makes its assignments unconditionally.</p>

(...) 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

The following example matches a leading newline in the subject string pointed at by cursor.

```
char *cursor, *newcursor, *ptr;
...
newcursor = regex((ptr = regcmp("^\\n", (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;
...
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;
...
newcursor = regex(name, string);
```

FILES /usr/ccs/lib/libgen.a

SEE ALSO **ed(1)**, **regcmp(1)**, **malloc(3C)**

NOTES The user program may run out of memory if **regcmp()** is called iteratively without freeing the vectors no longer required.

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	regex, re_comp, re_exec – regular expression handler
SYNOPSIS	<pre>/usr/ucb/cc [flag ...] file ... char *re_comp(sp) register char *sp; int re_exec(p1) register char *p1;</pre>
DESCRIPTION	<p>re_comp() compiles a string into an internal form suitable for pattern matching. re_exec() checks the argument string against the last string passed to re_comp().</p> <p>re_comp() returns a NULL pointer if the string <i>sp</i> was compiled successfully; otherwise a string containing an error message is returned. If re_comp() is passed 0 or a NULL string, it returns without changing the currently compiled regular expression.</p> <p>re_exec() returns 1 if the string <i>p1</i> matches the last compiled regular expression, 0 if the string <i>p1</i> failed to match the last compiled regular expression, and -1 if the compiled regular expression was invalid (indicating an internal error).</p> <p>The strings passed to both re_comp() and re_exec() may have trailing or embedded NEWLINE characters; they are terminated by NULL characters. The regular expressions recognized are described in the manual entry for ed(1), given the above difference.</p>
RETURN VALUES	<p>re_exec() returns -1 for an internal error.</p> <p>re_comp() returns one of the following strings if an error occurs:</p> <ul style="list-style-type: none"> No previous regular expression Regular expression too long unmatched \(missing] too many \(\) pairs unmatched \)
SEE ALSO	ed(1) , ex(1) , grep(1) , regcmp(1) , regcmp(3G) , regexpr(3G)
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 ed regular expressions of the form <code>\{m\}</code>, <code>\{m,\}</code>, or <code>\{m,n\}</code> are not supported.</p>

NAME	regexr, compile, step, advance – regular expression compile and match routines
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lgen [<i>library</i> ...] #include <regexr.h> char *compile(const char *instring, char *expbuf, char *endbuf); int step(const char *string, char *expbuf); int advance(const char *string, char *expbuf); extern char *loc1, *loc2, *locs; extern int nbra, regerrno, reglength; extern char *braslist[], *braelist[];</pre>
MT-LEVEL	MT-Safe
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 ed(1).</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 NULL, compile() uses malloc(3C) 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 NULL. If the compiled expression cannot fit in (<i>endbuf</i>–<i>expbuf</i>) bytes, compile() returns NULL and regerrno (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 compile().</p> <p>The function step() 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 step(). The variables set in step() are loc1 and loc2. loc1 is a pointer to the first character that matched the regular expression. The variable loc2 points to the character after the last character that matches the regular expression. Thus if the regular expression matches the entire line, loc1 points to the first character of <i>string</i> and loc2 points to the null at the end of <i>string</i>.</p> <p>The purpose of step() is to step through the <i>string</i> argument until a match is found or until the end of <i>string</i> is reached. If the regular expression begins with ^, step() tries to match the regular expression at the beginning of the string only.</p>

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

The following is similar to the regular expression code from **grep**:

```
#include <regexpr.h>
...
if(compile(*argv, (char *)0, (char *)0) == (char *)0)
    regerr(regerrno);
...
if (step(linebuf, expbuf))
    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**.

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 search string.
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.

ENVIRONMENT

If any of the `LC_*` variables (`LC_CTYPE`, `LC_MESSAGES`, `LC_TIME`, `LC_COLLATE`, `LC_NUMERIC`, and `LC_MONETARY`) (see `environ(5)`) are not set in the environment, the operational behavior of `tar` for each corresponding locale category is determined by the value of the `LANG` environment variable. If `LC_ALL` is set, its contents are used to override both the `LANG` and the other `LC_*` variables. If none of the above variables is set in the environment, the "C" (U.S. style) locale determines how `tar` behaves.

LC_CTYPE

Determines how `tar` handles characters. When `LC_CTYPE` is set to a valid value, `tar` can display and handle text and filenames containing valid characters for that locale. `tar` can display and handle Extended Unix code (EUC) characters where any individual character can be 1, 2, or 3 bytes wide. `tar` can also handle EUC characters of 1, 2, or more column widths. In the "C" locale, only characters from ISO 8859-1 are valid.

SEE ALSO

`ed(1)`, `grep(1)`, `sed(1)`, `malloc(3C)`, `regex(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	remove – remove file
SYNOPSIS	<pre>#include <stdio.h> int remove(const char *path);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>remove() 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.</p> <p>For files, remove() is identical to unlink(). For directories, remove() is identical to rmdir().</p> <p>See rmdir(2) and unlink(2) for a detailed list of failure conditions.</p>
RETURN VALUES	Upon successful completion, remove() returns a value of 0; otherwise, it returns a value of -1 and sets errno to indicate an error.
SEE ALSO	rmdir(2) , unlink(2)

NAME	resolver, res_mkquery, res_send, res_init, dn_comp, dn_expand – resolver routines																				
SYNOPSIS	<pre>cc [flag ...] file ... -lresolv -lsocket -lnsl [library ...] #include <sys/types.h> #include <netinet/in.h> #include <arpa/nameser.h> #include <resolv.h> int res_mkquery(int op, char *dname, int class, int type, char *data, int datalen, struct rrec *newrr, char *buf, int buflen); int res_send(char *msg, int msglen, char *answer, int anslen); int res_init(void); int dn_comp(char *exp_dn, char *comp_dn, int length, char **dnptrs, char **lastdnptr); int dn_expand(char *msg, char *comp_dn, char exp_dn, int msglen, int length);</pre>																				
MT-LEVEL	Unsafe																				
DESCRIPTION	<p>These routines are used for making, sending and interpreting packets to Internet domain name servers. Global information that is used by the resolver routines is kept in the variable <code>_res</code>. Most of the values have reasonable defaults and can be ignored. Options are a simple bit mask and are OR-ed in to enable. Options stored in <code>_res.options</code> are defined in <code><resolv.h></code> and are as follows.</p> <table border="0"> <tr> <td style="padding-right: 20px;">RES_INIT</td> <td>True if the initial name server address and default domain name are initialized (that is, <code>res_init()</code> has been called).</td> </tr> <tr> <td>RES_DEBUG</td> <td>Print debugging messages.</td> </tr> <tr> <td>RES_AAONLY</td> <td>Accept authoritative answers only. <code>res_send()</code> will continue until it finds an authoritative answer or finds an error. Currently this is not implemented.</td> </tr> <tr> <td>RES_USEVC</td> <td>Use TCP connections for queries instead of UDP.</td> </tr> <tr> <td>RES_PRIMARY</td> <td>Query primary server only.</td> </tr> <tr> <td>RES_IGNTC</td> <td>Unused currently (ignore truncation errors, that is, do not retry with TCP).</td> </tr> <tr> <td>RES_RECURSE</td> <td>Set the recursion desired bit in queries. This is the default. <code>res_send()</code> does not do iterative queries and expects the name server to handle recursion.</td> </tr> <tr> <td>RES_DEFNAMES</td> <td>Append the default domain name to single label queries. This is the default.</td> </tr> <tr> <td>RES_STAYOPEN</td> <td>Used with <code>RES_USEVC</code> to keep the TCP connection open between queries. This is useful only in programs that regularly do many queries. UDP should be the normal mode used.</td> </tr> <tr> <td>RES_DNSRCH</td> <td>Search up local domain tree.</td> </tr> </table>	RES_INIT	True if the initial name server address and default domain name are initialized (that is, <code>res_init()</code> has been called).	RES_DEBUG	Print debugging messages.	RES_AAONLY	Accept authoritative answers only. <code>res_send()</code> will continue until it finds an authoritative answer or finds an error. Currently this is not implemented.	RES_USEVC	Use TCP connections for queries instead of UDP.	RES_PRIMARY	Query primary server only.	RES_IGNTC	Unused currently (ignore truncation errors, that is, do not retry with TCP).	RES_RECURSE	Set the recursion desired bit in queries. This is the default. <code>res_send()</code> does not do iterative queries and expects the name server to handle recursion.	RES_DEFNAMES	Append the default domain name to single label queries. This is the default.	RES_STAYOPEN	Used with <code>RES_USEVC</code> to keep the TCP connection open between queries. This is useful only in programs that regularly do many queries. UDP should be the normal mode used.	RES_DNSRCH	Search up local domain tree.
RES_INIT	True if the initial name server address and default domain name are initialized (that is, <code>res_init()</code> has been called).																				
RES_DEBUG	Print debugging messages.																				
RES_AAONLY	Accept authoritative answers only. <code>res_send()</code> will continue until it finds an authoritative answer or finds an error. Currently this is not implemented.																				
RES_USEVC	Use TCP connections for queries instead of UDP.																				
RES_PRIMARY	Query primary server only.																				
RES_IGNTC	Unused currently (ignore truncation errors, that is, do not retry with TCP).																				
RES_RECURSE	Set the recursion desired bit in queries. This is the default. <code>res_send()</code> does not do iterative queries and expects the name server to handle recursion.																				
RES_DEFNAMES	Append the default domain name to single label queries. This is the default.																				
RES_STAYOPEN	Used with <code>RES_USEVC</code> to keep the TCP connection open between queries. This is useful only in programs that regularly do many queries. UDP should be the normal mode used.																				
RES_DNSRCH	Search up local domain tree.																				

res_init() reads the initialization file to get the default domain name and the Internet address of the initial hosts running the name server. If this line does not exist, the host running the resolver is tried.

res_mkquery() makes a standard query message and places it in *buf*. **res_mkquery()** will return the size of the query or **-1** if the query is larger than *buflen*. *op* is usually **QUERY** but can be any of the query types defined in `<arpa/nameser.h>`. *dname* is the domain name. If *dname* consists of a single label and the **RES_DEFNAMES** flag is enabled (the default), *dname* will be appended with the current domain name. The current domain name is defined in a system file and can be overridden by the environment variable **LOCALDOMAIN**. *newrr* is currently unused but is intended for making update messages. *class* and *type* define the class and type of query. **data* is the resource record; and *datalen* is the length of the record.

res_send() sends a query to name servers and returns an answer. It will call **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. *anslen* returns the length of the response or **-1** if there were errors.

dn_expand() expands the compressed domain name *comp_dn* to a full domain name. Expanded names are converted to upper case. *msg* is a pointer to the beginning of the message, *exp_dn* is a pointer to a buffer of size *length* for the result. The size of compressed name is returned or **-1** if there was an error.

dn_comp() compresses the domain name *exp_dn* and stores it in *comp_dn*. The size of the compressed name is returned or **-1** if there were errors. *length* is the size of the array pointed to by *comp_dn*. *dnptrs* is a list of pointers to previously compressed names in the current message. The first pointer points to the beginning of the message and the list ends with **NULL**. *lastdnptr* is a pointer to the end of the array pointed to *dnptrs*. A side effect is to update the list of pointers for labels inserted into the message by **dn_comp()** as the name is compressed. If *dnptr* is **NULL**, do not try to compress names. If *lastdnptr* is **NULL**, do not update the list.

FILES /etc/resolv.conf

SEE ALSO in.named(1M), nstest(1M), resolv.conf(4)

NOTES These interfaces are unsafe in multithreaded applications. Unsafe interfaces should be called only from the main thread.

NAME	rexec – return stream to a remote command
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lsocket -lnsl [<i>library</i> ...]</pre> <pre>int rexec(char **<i>ahost</i>, unsigned short <i>inport</i>, const char *<i>user</i>, const char *<i>passwd</i>, const char *<i>cmd</i>, int *<i>fd2p</i>);</pre>
MT-LEVEL	Unsafe
DESCRIPTION	<p>rexec() looks up the host <i>ahost</i> using gethostbyname(3N), returning <code>-1</code> if the host does not exist. Otherwise <i>ahost</i> 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.</p> <p>The port <i>inport</i> specifies which well-known DARPA Internet port to use for the connection. The protocol for connection is described in detail in in.rexecd(1M).</p> <p>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 <i>fd2p</i> is non-zero, then an auxiliary channel to a control process will be setup, and a file descriptor for it will be placed in <i>fd2p</i>. 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 <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 process, although you may be able to get its attention by using out-of-band data.</p>
RETURN VALUES	<p>If rexec() succeeds, a file descriptor number, which is a socket of type SOCK_STREAM, is returned by the routine. <i>ahost</i> is set to the standard name of the host, and if <i>fd2p</i> is not NULL, a file descriptor number is placed in <i>fd2p</i> which represents the command's standard error stream.</p> <p>If rexec() fails, <code>-1</code> is returned.</p>
SEE ALSO	in.rexecd(1M) , gethostbyname(3N) , getservbyname(3N)
NOTES	<p>There is no way to specify options to the socket() call that rexec() makes.</p> <p>This interface is unsafe in multithreaded applications. Unsafe interfaces should be called only from the main thread.</p>

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>
MT-LEVEL	MT-Safe with exceptions
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 <rpc/rpc.h>. Routines that take a netconfig structure also require that <netconfig.h> be included. Applications using RPC and XDR routines should be linked with the libnsl library.</p>
Multithread Considerations	<p>In the case of multithreaded applications, the _REENTRANT flag must be defined on the command line at compilation time (-D_REENTRANT). Defining this flag enables a thread-specific version of rpc_clnt_create (see rpc_clnt_create(3N)).</p> <p>Client-side routines are MT-Safe. CLIENT handles (see rpc_clnt_create(3N)) 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).</p> <p>Server-side routines are mostly MT-Unsafe. In this implementation the service transport handle, SVCXPRT (see rpc_svc_create(3N)), 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. Routines that are affected by this restriction are marked as unsafe for MT applications (see rpc_svc_calls(3N)).</p>
Nettype	<p>Some of the high-level RPC interface routines take a <i>nettype</i> string as one of the parameters (for example, clnt_create(), svc_create(), rpc_reg(), rpc_call()). 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> <ul style="list-style-type: none"> netpath 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 visible. netpath 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.

Data Structures

Some of the data structures used by the RPC package are shown below.

The AUTH Structure

```

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 */
    u_int    oa_length; /* not to exceed MAX_AUTH_BYTES */
};

/*
 * Auth handle, interface to client 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 */
    };
};

```

The CLIENT
Structure

```

        void (*ah_destroy)(); /* destroy this structure */
    } *ah_ops;
    caddr_t ah_private;
} AUTH;

/*
 * Client rpc handle.
 * Created by individual implementations.
 * Client is responsible for initializing
 */
typedef struct {
    AUTH          *cl_auth;          /* authenticator */
    struct clnt_ops {
        enum clnt_stat (*cl_call)(); /* call remote procedure */
        void (*cl_abort)(); /* abort a call */
        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 */
    } *cl_ops;
    caddr_t cl_private; /* private stuff */
    char *cl_netid; /* network identifier */
    char *cl_tp; /* device name */
} CLIENT;

```

The SVCXPRT
Structure

```

enum xpirt_stat {
    XPRT_DIED,
    XPRT_MOREREQS,
    XPRT_IDLE
};

/*
 * Server side transport handle
 */
typedef struct {
    int xp_fd; /* file descriptor for the
                server handle */
    u_short xp_port; /* obsolete */
    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 {
    u_long      rq_prog;        /* service program number */
    u_long      rq_vers;       /* service protocol version */
    u_long      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 */
};

```

The XDR Structure

```

/*
 * 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 a long from underlying stream */
        bool_t (*x_putlong)(); /* put a long to underlying stream */
        bool_t (*x_getbytes)(); /* get bytes from underlying stream */
        bool_t (*x_putbytes)(); /* put bytes to underlying stream */
        u_int (*x_getpostn)(); /* returns bytes off from beginning */
        bool_t (*x_setpostn)(); /* lets you reposition the stream */
        long * (*x_inline)(); /* buf quick ptr to buffered data */
        void (*x_destroy)(); /* free privates of this xdr_stream */
    } *x_ops;
    caddr_t x_public; /* users' data */
    caddr_t x_private; /* 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_screateerror	rpc_clnt_create(3N)
clnt_sperrno	rpc_clnt_calls(3N)
clnt_sperror	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_noprogram	rpc_svc_err(3N)
svcerr_progvers	rpc_svc_err(3N)
svcerr_systemerr	rpc_svc_err(3N)

svcerr_weakauth	rpc_svc_err(3N)
svdfd_create	rpc_soc(3N)
svcrawl_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

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)**, **environ(5)**

NAME	rpc_clnt_auth, auth_destroy, authnone_create, authsys_create, authsys_create_default – library routines for client side remote procedure call authentication
MT-LEVEL	MT-Safe
DESCRIPTION	<p>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.</p> <p>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.</p> <p>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).</p> <p>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.</p>
Routines	<p>The following routines require that the header <rpc/rpc.h> be included (see rpc(3N) for the definition of the AUTH data structure).</p> <pre data-bbox="422 840 682 871">#include <rpc/rpc.h></pre> <pre data-bbox="422 882 812 913">void auth_destroy(AUTH *auth);</pre> <p data-bbox="519 924 1445 1018">A function macro that destroys the authentication information associated with <i>auth</i>. Destruction usually involves deallocation of private data structures. The use of <i>auth</i> is undefined after calling auth_destroy().</p> <pre data-bbox="422 1060 795 1092">AUTH *authnone_create(void);</pre> <p data-bbox="519 1102 1461 1197">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.</p> <pre data-bbox="422 1239 1282 1302">AUTH *authsys_create(const char *host, const uid_t uid, const gid_t gid, const int len, const gid_t *aup_gids);</pre> <p data-bbox="519 1312 1477 1438">Create and return an RPC authentication handle that contains AUTH_SYS authentication information. The parameter <i>host</i> is the name of the machine on which the information was created; <i>uid</i> is the user's user ID; <i>gid</i> is the user's current group ID; <i>len</i> and <i>aup_gids</i> refer to a counted array of groups to which the user belongs.</p> <pre data-bbox="422 1480 876 1512">AUTH *authsys_create_default(void);</pre> <p data-bbox="519 1522 1169 1554">Call authsys_create() with the appropriate parameters.</p>

SEE ALSO | **kerberos_rpc(3N), rpc(3N), rpc_clnt_calls(3N), rpc_clnt_create(3N), secure_rpc(3N)**

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
MT-LEVEL	MT-Safe
DESCRIPTION	<p>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.</p> <p>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.</p> <p>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)).</p> <p>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).</p>
Routines	<p>See rpc(3N) for the definition of the CLIENT data structure.</p> <p>#include <rpc/rpc.h></p> <p>enum clnt_stat clnt_call(CLIENT *clnt, const u_long procnum, const xdrproc_t inproc, const caddr_t in, const xdrproc_t outproc, caddr_t out, const struct timeval tout);</p> <p>A function macro that calls the remote procedure <i>procnum</i> associated with the client handle, <i>clnt</i>, which is obtained with an RPC client creation routine such as clnt_create() (see rpc_clnt_create(3N)). The parameter <i>inproc</i> is the XDR function used to encode the procedure's parameters, and <i>outproc</i> is the XDR function used to decode the procedure's results; <i>in</i> is the address of the procedure's argument(s), and <i>out</i> is the address of where to place the result(s). <i>tout</i> 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).</p> <p>If the remote call succeeds, the status returned is RPC_SUCCESS, otherwise an appropriate status is returned.</p> <p>bool_t clnt_freeres(CLIENT *clnt, const xdrproc_t outproc, caddr_t out);</p> <p>A function macro that frees any data allocated by the RPC/XDR system when it decoded the results of an RPC call. The parameter <i>out</i> is the address of the results, and <i>outproc</i> is the XDR routine describing the results. This routine returns 1 if the results were successfully freed, and 0 otherwise.</p>

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_spcreaterror()** (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 u_long prognum, const u_long versnum, const u_long 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 u_long prognum, const u_long versnum,
const u_long procnum, const xdrproc_t xargs, caddr_t argsp,
const xdrproc_t xresults, caddr_t resultsp, const int inittime, const int waittime,
const resultproc_t eachresult, 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 u_long prognum,
const u_long versnum, const u_long 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.

SEE ALSO

printf(3S), **rpc(3N)**, **rpc_clnt_auth(3N)**, **rpc_clnt_create(3N)**

NAME	rpc_clnt_create, clnt_control, clnt_create, clnt_create_vers, clnt_destroy, clnt_dg_create, clnt_pcreateerror, clnt_raw_create, clnt_spccreateerror, clnt_tli_create, clnt_tp_create, clnt_vc_create, rpc_createerr – library routines for dealing with creation and manipulation of CLIENT handles																					
MT-LEVEL	MT-Safe																					
DESCRIPTION	<p>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.</p> <p>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.</p>																					
Routines	<p>See rpc(3N) for the definition of the CLIENT data structure.</p> <p>#include <rpc/rpc.h></p> <p>bool_t clnt_control(CLIENT *clnt, const u_int req, char *info);</p> <p>A function macro to change or retrieve various information about a client object. <i>req</i> indicates the type of operation, and <i>info</i> is a pointer to the information. For both connectionless and connection-oriented transports, the supported values of <i>req</i> and their argument types and what they do are:</p> <table border="0"> <tr> <td>CLSET_TIMEOUT</td> <td>struct timeval *</td> <td>set total timeout</td> </tr> <tr> <td>CLGET_TIMEOUT</td> <td>struct timeval *</td> <td>get total timeout</td> </tr> </table> <p>Note: if you set the timeout using clnt_control(), the timeout argument passed by clnt_call() is ignored in all subsequent calls.</p> <p>Note: If you set the timeout value to 0 clnt_control() immediately returns an error (RPC_TIMEDOUT). Set the timeout parameter to 0 for batching calls.</p> <table border="0"> <tr> <td>CLGET_FD</td> <td>int *</td> <td>get the associated file descriptor</td> </tr> <tr> <td>CLGET_SVC_ADDR</td> <td>struct netbuf *</td> <td>get servers address</td> </tr> <tr> <td>CLSET_FD_CLOSE</td> <td>void</td> <td>close the file descriptor when destroying the client handle (see clnt_destroy())</td> </tr> <tr> <td>CLSET_FD_NCLOSE</td> <td>void</td> <td>do not close the file descriptor when destroying the client handle</td> </tr> <tr> <td>CLGET_VERS</td> <td>unsigned long *</td> <td>get the RPC program's version number associated with the client handle</td> </tr> </table>	CLSET_TIMEOUT	struct timeval *	set total timeout	CLGET_TIMEOUT	struct timeval *	get total timeout	CLGET_FD	int *	get the associated file descriptor	CLGET_SVC_ADDR	struct netbuf *	get servers address	CLSET_FD_CLOSE	void	close the file descriptor when destroying the client handle (see clnt_destroy())	CLSET_FD_NCLOSE	void	do not close the file descriptor when destroying the client handle	CLGET_VERS	unsigned long *	get the RPC program's version number associated with the client handle
CLSET_TIMEOUT	struct timeval *	set total timeout																				
CLGET_TIMEOUT	struct timeval *	get total timeout																				
CLGET_FD	int *	get the associated file descriptor																				
CLGET_SVC_ADDR	struct netbuf *	get servers address																				
CLSET_FD_CLOSE	void	close the file descriptor when destroying the client handle (see clnt_destroy())																				
CLSET_FD_NCLOSE	void	do not close the file descriptor when destroying the client handle																				
CLGET_VERS	unsigned long *	get the RPC program's version number associated with the client handle																				

CLSET_VERS	unsigned long *	set the RPC program's version number associated with the client handle. This assumes that the RPC server for this new version is still listening at the address of the previous version.
CLGET_XID	unsigned long *	get the XID of the previous remote procedure call
CLSET_XID	unsigned long *	set the XID of the next remote procedure call

The following operations are valid for connectionless 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.

CLIENT *clnt_create(const char *host, const u_long prognum, const u_long versnum, const char *nettype);

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 **rpc_clnt_calls(3N)**).

CLIENT *clnt_create_vers(const char *host, const u_long prognum, u_long *vers_outp, const u_long vers_low, const u_long 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.

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 u_long *prognum*, const u_long *versnum*, const u_int *sendsz*,
const u_int *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 u_long prognum, const u_long 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_spcreateerror(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 fildes, const struct netconfig *netconf, const struct netbuf *svcaddr, const u_long prognum, const u_long versnum, const u_int sendsz, const u_int recvsz);

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. *fildes* 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 *fildes* is **RPC_ANYFD** and *netconf* is NULL, a **RPC_UNKNOWNPROTO** error is set. If *fildes* is unbound, then it will attempt to bind the descriptor. The user may specify the size of the buffers with the parameters *sendsz* and *recvsz*; 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 u_long prognum, const u_long 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_vc_create(const int fildes, const struct netbuf *svcaddr,  
const u_long prognum, const u_long versnum, const u_int sendsz,  
const u_int recvsz);
```

This routine creates an RPC client for the remote program *prog*num and version *vers*num; the client uses a connection-oriented transport. The remote program is located at address *sv*caddr. The parameter *fil*des is an open and bound file descriptor. The user may specify the size of the send and receive buffers with the parameters *send*sz and *recv*sz; values of 0 choose suitable defaults. This routine returns NULL if it fails.

The address *sv*caddr 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`.

SEE ALSO `rpc(3N)`, `rpc_clnt_auth(3N)`, `rpc_clnt_calls(3N)`, `rpcbind(1M)`

NAME	rpc_control – library routine for manipulating global RPC attributes for client and server applications																														
SYNOPSIS	bool_t rpc_control(int op, void *info);																														
MT-LEVEL	MT-Safe																														
DESCRIPTION	<p>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. A function that allows applications to set and modify global attributes that apply to client as well as server functions. <i>op</i> indicates the type of operation, and <i>info</i> is a pointer to the operation specific information. The supported values of <i>op</i> and their argument types, and what they do are:</p> <table border="0"> <tr> <td>RPC_SVC_MTMODE_SET</td> <td>int *</td> <td>set multithread mode</td> </tr> <tr> <td>RPC_SVC_MTMODE_GET</td> <td>int *</td> <td>get multithread mode</td> </tr> <tr> <td>RPC_SVC_THRMAX_SET</td> <td>int *</td> <td>set maximum number of threads</td> </tr> <tr> <td>RPC_SVC_THRMAX_GET</td> <td>int *</td> <td>get maximum number of threads</td> </tr> <tr> <td>RPC_SVC_THRTOTAL_GET</td> <td>int *</td> <td>get number of active threads</td> </tr> <tr> <td>RPC_SVC_THRCREATES_GET</td> <td>int *</td> <td>get number of threads created</td> </tr> <tr> <td>RPC_SVC_THRERRORS_GET</td> <td>int *</td> <td>get number of thread create errors</td> </tr> </table> <p>There are three multithread (MT) modes. These are:</p> <table border="0"> <tr> <td>RPC_SVC_MT_NONE</td> <td>Single threaded mode</td> <td>(default)</td> </tr> <tr> <td>RPC_SVC_MT_AUTO</td> <td>Automatic MT mode</td> <td></td> </tr> <tr> <td>RPC_SVC_MT_USER</td> <td>User MT mode</td> <td></td> </tr> </table> <p>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.</p> <p>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.</p> <p>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.</p>	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_MT_NONE	Single threaded mode	(default)	RPC_SVC_MT_AUTO	Automatic MT mode		RPC_SVC_MT_USER	User MT mode	
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_MT_NONE	Single threaded mode	(default)																													
RPC_SVC_MT_AUTO	Automatic MT mode																														
RPC_SVC_MT_USER	User MT mode																														

RETURN VALUES

This routine returns **TRUE** if the operation was successful, and **FALSE** otherwise.

SEE ALSO

rpc(3N), **rpc_svc_calls(3N)**, **rpcbind(1M)**

NAME	rpc_rac, rac_drop, rac_poll, rac_recv, rac_send – remote asynchronous calls
SYNOPSIS	<pre>cc [flag ...] file ... -lrac -lnsl [library ...] #include <rpc/rpc.h> #include <rpc/rac.h></pre>
MT-LEVEL	Unsafe
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> <pre>#include <rpc/rpc.h> void rac_drop(CLIENT *cl, void *h);</pre> <p>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> <p>After a call to rac_drop() the handle referred to by <i>h</i> is invalid. It may no longer be used in any asynchronous operation.</p> <pre>enum clnt_stat rac_poll(CLIENT *cl, void *h);</pre> <p>rac_poll() returns the status of the call currently in progress on the <CLIENT, asynchronous handle> tuple referred to by <i>cl</i> and <i>h</i>.</p> <p>rac_poll() return values are:</p> <ul style="list-style-type: none"> RPC_SUCCESS A reply has been received and is available for reading by rac_recv(). 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 *h* 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.

enum clnt_stat rac_rcv(CLIENT *cl, void * h);

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.

void *rac_send(client *cl, unsigned long proc, xdrproc_t xargs, void *argsp, xdrproc_t xresults, void *resultsp, struct timeval timeout);

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_recv()** 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_recv()**), 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_sends** 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.

SEE ALSO

poll(2), **rpc(3N)**, **rpc_clnt_create(3N)**, **rpc_clnt_calls(3N)**, **xdr(3N)**

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 **libnsl** 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	rpc_soc, authdes_create, authunix_create, authunix_create_default, callrpc, clnt_broadcast, clntraw_create, clnttcp_create, clntudp_bufcreate, clntudp_create, get_myaddress, pmap_getmaps, pmap_getport, pmap_rmtcall, pmap_set, pmap_unset, registerrpc, svc_fds, svc_getcaller, svc_getreq, svc_register, svc_unregister, svcsfd_create, svcraw_create, svctcp_create, svcudp_bufcreate, svcudp_create, xdr_authunix_parms – obsolete library routines for RPC
MT-LEVEL	Unsafe
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. 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 librpcsoc and libnsl. If the user passed RPC_ANYSOCK as the file descriptor, and the application is linked with libnsl only, then the routine will return a TLI file descriptor and not a socket.</p>
Routines	<p>The following routines require that the header <rpc/rpc.h> be included. The symbol PORTMAP 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 * authdes_create(char *name, unsigned window, struct sockaddr *syncaddr, des_block *ckey);</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 keyserv(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 parameter <i>name</i> is the network name, or <i>netname</i>, 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</p>

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 *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 it is supplied, however, then it will be used instead.

Warning: this routine exists for backward compatibility only, and is obsoleted by **authdes_seccreate()** (see **secure_rpc(3N)**).

AUTH * authunix_create(char *host, int uid, int gid, int grouplen, int gidlistp);

Create and return an RPC authentication handle that contains 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 obsoleted 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 obsoleted by **authsys_create_default()** (see **rpc_clnt_auth(3N)**).

callrpc(char *host, u_long prognum, u_long versnum, u_long 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_pereno()** (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(u_long prognum, u_long versnum, u_long 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 caller's 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 * clntraw_create(u_long prognum, u_long 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 **svcrow_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, u_long prognum, u_long versnum,
int *fdp, u_int sendsz, u_int 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, u_long prognum, u_long versnum, struct timeval wait, int *fdp, u_int sendsz, u_int 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, u_long prognum, u_long 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*→*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.

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, u_long prognum, u_long versnum, u_long 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, u_long prognum, u_long versnum, u_long procnum, char *in, xdrproct_t inproc, char *out, xdrproct_t outproc, struct timeval tout, u_long *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 portmap_set(u_long prognum, u_long versnum, u_long 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 portmap_unset(u_long prognum, u_long 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 obsoleted by **svc_fdset** (see **rpc_svc_calls(3N)**).

struct sockaddr_in * svc_getcaller(SVCXPRT *xpirt);

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_gettrpcaller()` (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 * svcfd_create(int fd, u_int sendsz, u_int 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, u_int sendsz, u_int 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->xp_fd` is the transport's file descriptor, and `xprt->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, u_int sendsz, u_int 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*→*xp_fd* is the transport's file descriptor, and *xprt*→*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*→*xp_fd* is the transport's file descriptor, and *xprt*→*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(u_long prognum, u_long versnum, u_long 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()** (see **rpc_svc_calls(3N)**).

```
svc_register(SVCXPRT *xpirt, u_long prognum, u_long versnum, void (*dispatch)(),  
u_long 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 *xpirt*→*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 *xpirt);
```

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()** (see **rpc_svc_calls(3N)**).

```
void svc_unregister(u_long prognum, u_long 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()** (see **rpc_svc_calls(3N)**).

```
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)**).

SEE ALSO

keyserv(1M), **rpcbind(1M)**, **rpcinfo(1M)**, **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)**, **rpcbind(3N)**, **secure_rpc(3N)**, **select(3C)**

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_getrpcaller, svc_pollset, svc_run, svc_sendreply – library routines for RPC servers
MT-LEVEL	See the NOTES section of this page.
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 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.</p> <p>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.</p>
Routines	<p>#include <rpc/rpc.h></p> <p>int svc_dg_enablecache(SVCXPRT *xpvt, const unsigned long cache_size);</p> <p>This function allocates a duplicate request cache for the service endpoint <i>xpvt</i>, large enough to hold <i>cache_size</i> 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.</p> <p>This function is safe in MT applications.</p> <p>int svc_done(SVCXPRT *xpvt);</p> <p>This function frees resources allocated to service a client request directed to the service endpoint <i>xpvt</i>. 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 <i>xpvt</i> should not be referenced by the service procedure. Server multithreading modes and parameters can be set using the rpc_control() call.</p> <p>This function is safe in MT applications. It will have no effect if invoked in modes other than the User MT mode.</p>

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_getreqset()** 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.

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; *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 function macro is unsafe in MT applications.

struct netbuf *svc_getrpcaller(const SVCXPRT *xpvt);

The approved way of getting the network address of the caller of a procedure associated with the RPC service transport handle *xpvt*.

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.

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)

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
MT-LEVEL	MT-Safe
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	<p>See rpc(3N) for the definition of the SVCXPRT data structure.</p> <p>#include <rpc/rpc.h></p> <p>bool_t</p> <p>svc_control(SVCXPRT *svc, const u_int 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> <p>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.</p> <p>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 – an RPC_PROGVERSMISMATCH error will be returned), or 1 (indicating that the out of range request should be silently ignored).</p> <p>int svc_create(const void (*dispatch)(const struct svc_req *, const SVCXPRT *), const u_long prognum, const u_long versnum, const char *nettype);</p> <p>svc_create() creates server handles for all the transports belonging to the class <i>nettype</i>.</p> <p><i>nettype</i> defines a class of transports which can be used for a particular</p>

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 *xpirt);

A function macro that destroys the RPC service handle `xpirt`. Destruction usually involves deallocation of private data structures, including `xpirt` itself. Use of `xpirt` is undefined after calling this routine.

SVCXPRT *svc_dg_create(const int fildes, const u_int sendsz, const u_int 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 `fildes` 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 fildes, const u_int sendsz, const u_int 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 *fildes*, const struct netconfig **netconf*,
const struct t_bind **bindaddr*, const u_int *sendsz*, const u_int *recvsz*);**

This routine creates an RPC server handle, and returns a pointer to it. *fildes* is the file descriptor on which the service is listening. If *fildes* 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 *fildes* is bound to the address specified by *bindaddr*, otherwise *fildes* 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 u_long *prognum*, const u_long *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 u_int *sendsz*, const u_int *recvsz*);

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 *recvsz*; 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.

SEE ALSO

rpcbind(1M), rpc(3N), rpc_svc_calls(3N), rpc_svc_err(3N), rpc_svc_reg(3N)

NAME	rpc_svc_err, svcerr_auth, svcerr_decode, svcerr_noproc, svcerr_noprog, svcerr_progvers, svcerr_systemerr, svcerr_weakauth – library routines for server side remote procedure call errors
MT-LEVEL	MT-Safe
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> <p>#include <rpc/rpc.h></p> <p>void svcerr_auth(const SVCXPRT *xpvt, const enum auth_stat why); 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 *xpvt); 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 *xpvt); Called by a service dispatch routine that does not implement the procedure number that the caller requests.</p> <p>void svcerr_noprog(const SVCXPRT *xpvt); 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 *xpvt, u_long low_vers, u_long high_vers); 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 *xpvt); 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.</p>

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)**.

SEE ALSO

rpc(3N), rpc_svc_calls(3N), rpc_svc_create(3N), rpc_svc_reg(3N)

NAME	rpc_svc_reg, rpc_reg, svc_reg, svc_unreg, svc_auth_reg, xpirt_register, xpirt_unregister – library routines for registering servers
MT-LEVEL	MT-Safe
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	See rpc(3N) for the definition of the SVCXPRT data structure.

#include <rpc/rpc.h>

bool_t rpc_reg(const u_long prognum, const u_long versnum, const u_long procnum, const char *(*procname), const xdrproc_t inproc, const xdrproc_t outproc, const char *nettype);

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 the XDR function used to decode the parameters while *outproc* is the XDR function used to encode the results. Procedures are registered on all available transports of the class *nettype*. See **rpc(3N)**. This routine returns **0** if the registration succeeded, **-1** otherwise.

int svc_reg(const SVCXPRT *xpirt, const u_long prognum, const u_long versnum, const void (*dispatch), const struct netconfig *netconf);

Associates *prognum* and *versnum* with the service dispatch procedure, *dispatch*. If *netconf* is **NULL**, the service is not registered with the *rpcbind* service. For example, if a service has already been registered using some other means, such as **inetd** (see **inetd(1M)**), it will not need to be registered again. If *netconf* is non-zero, then a mapping of the triple [*prognum*, *versnum*, *netconf*→*nc_netid*] to *xpirt*→*xp_ltaddr* is established with the local **rpcbind** service.

The **svc_reg()** routine returns **1** if it succeeds, and **0** otherwise.

void svc_unreg(const u_long prognum, const u_long versnum);

Remove from the **rpcbind** service, all mappings of the triple [*prognum*, *versnum*, *all-transports*] to network address and all mappings within the RPC service package of the double [*prognum*, *versnum*] to dispatch routines.

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.

SEE ALSO

inetd(1M), **rpcbind(1M)**, **rpc(3N)**, **rpc_svc_calls(3N)**, **rpc_svc_create(3N)**, **rpc_svc_err(3N)**, **rpcbind(3N)**, **select(3C)**

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
MT-LEVEL	Safe
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	See rpc(3N) for the definition of the XDR data structure.
	<pre>#include <rpc/rpc.h></pre>
	<pre>bool_t xdr_accepted_reply(XDR *xdrs, const struct accepted_reply *ar);</pre> <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>
	<pre>bool_t xdr_authsys_parms(XDR *xdrs, struct authsys_parms *aupp);</pre> <p>Used for describing UNIX operating system credentials. It includes machine-name, uid, gid list, etc.</p>
	<pre>void xdr_callhdr(XDR *xdrs, struct rpc_msg *chdr);</pre> <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>
	<pre>bool_t xdr_callmsg(XDR *xdrs, struct rpc_msg *cmsg);</pre> <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>
	<pre>bool_t xdr_opaque_auth(XDR *xdrs, struct opaque_auth *ap);</pre> <p>Used for describing RPC opaque authentication information messages.</p>
	<pre>bool_t xdr_rejected_reply(XDR *xdrs, const struct rejected_reply *rr);</pre> <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.

SEE ALSO [rpc\(3N\)](#), [xdr\(3N\)](#)

NAME	rpcbind, rpcb_getmaps, rpcb_getaddr, rpcb_gettime, rpcb_rmtcall, rpcb_set, rpcb_unset – library routines for RPC bind service
MT-LEVEL	MT-Safe
DESCRIPTION	These routines allow client C programs to make procedure calls to the RPC binder service. rpcbind (see rpcbind (1M)) maintains a list of mappings between programs and their universal addresses.
Routines	<p>#include <rpc/rpc.h></p> <p>struct rpcblist *rpcb_getmaps(const struct netconfig *netconf, const char *host); An interface to the rpcbind 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 rpcbind service on <i>host</i>. This routine will return NULL, if the remote rpcbind could not be contacted.</p> <p>bool_t rpcb_getaddr(const u_long prognum, const u_long versnum, const struct netconfig *netconf, struct netbuf *svcaddr, const char *host); An interface to the rpcbind 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 rpcbind service. In the latter case, the global variable rpc_createerr (see rpc_clnt_create(3N)) contains the RPC status.</p> <p>bool_t rpcb_gettime(const char *host, time_t *timep); This routine returns the time on <i>host</i> in <i>timep</i>. If <i>host</i> 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.</p> <p>enum clnt_stat rpcb_rmtcall(const struct netconfig *netconf, const char *host, const u_long prognum, const u_long versnum, const u_long procnum, const xdrproc_t inproc, const caddr_t in, const xdrproc_t outproc, caddr_t out, const struct timeval tout, struct netbuf *svcaddr); An interface to the rpcbind service, which instructs rpcbind on <i>host</i> 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 <i>svcaddr</i> 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).</p>

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.

```
bool_t rpcb_set(const u_long prognum, const u_long versnum,  
               const struct netconfig *netconf, const struct netbuf *svcaddr);
```

An interface to the **rpcbind** service, which establishes a mapping between the triple [*prognum*, *versnum*, *netconf*→*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.

```
bool_t rpcb_unset(const u_long prognum, const u_long versnum,  
                  const struct netconfig *netconf);
```

An interface to the **rpcbind** service, which destroys the mapping between the triple [*prognum*, *versnum*, *netconf*→*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-transports*] 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)**).

SEE ALSO **rpcbind(1M)**, **rpcinfo(1M)**, **rpc_clnt_calls(3N)**, **rpc_svc_calls(3N)**

NAME	rstat, havedisk – get performance data from remote kernel
PROTOCOL	/usr/include/rpcsvc/rstat.x
SYNOPSIS	<pre>cc [<i>flag ...</i>] <i>file ...</i> -lrpcsvc [<i>library ...</i>] #include <rpc/rpc.h> #include <rpcsvc/rstat.h> enum clnt_stat rstat(char *host, struct statstime *statp); havedisk(char *host);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>These routines require that the rpc.rstatd(1M) daemon be configured and available on the remote system indicated by <i>host</i>. The rstat() protocol is used to gather statistics from remote kernel. Statistics will be available on items such as paging, swapping, and cpu utilization.</p> <p>rstat() fills in the statstime structure <i>statp</i> for <i>host</i>. <i>statp</i> must point to an allocated statstime structure. rstat() returns RPC_SUCCESS if it was successful; otherwise a enum clnt_stat is returned which can be displayed using clnt_perrno(3N).</p> <p>havedisk() returns 1 if <i>host</i> has disk, 0 if it does not, and -1 if this cannot be determined.</p> <p>The following XDR routines are available in librpcsvc:</p> <pre> xdr_statstime xdr_statsvar</pre>
SEE ALSO	rpc.rstatd (1M), rup (1), rpc_clnt_calls (3N)

NAME	rusers, rnusers – return information about users on remote machines
PROTOCOL	/usr/include/rpcsvc/rusers.x
SYNOPSIS	<pre>cc [<i>flag ...</i>] <i>file ...</i> -lrpcsvc [<i>library ...</i>] #include <rpc/rpc.h> #include <rpcsvc/rusers.h> enum clnt_stat rusers(char *host, struct utmpidlearr *up); int rnusers(char *host);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>These routines require that the rpc.rusersd(1M) daemon be configured and available on the remote system indicated by <i>host</i>. The rusers() protocol is used to retrieve information about users logged in on the remote system.</p> <p>rusers() fills the utmpidlearr structure with data about <i>host</i>, and returns 0 if successful. <i>up</i> must point to an allocated utmpidlearr structure. If rusers() returns successful it will have allocated data structures within the <i>up</i> structure, which should be freed with xdr_free(3N) 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 enum clnt_stat and can be displayed with clnt_perror(3N) or clnt_sperrno(3N).</p> <p>See the header <rpcsvc/rusers.h> for a definition of struct utmpidlearr.</p> <p>rnusers() returns the number of users logged on to <i>host</i> (-1 if it cannot determine that number).</p> <p>The following XDR routines are available in librpcsvc:</p> <pre> xdr_utmpidlearr.</pre>
SEE ALSO	rusers(1) , rpc.rusersd(1M) , rpc_clnt_calls(3N) , xdr_free(3N)

NAME	rwall – write to specified remote machines
PROTOCOL	<code>/usr/include/rpcsvc/rwall.x</code>
SYNOPSIS	<pre>cc [<i>flag ...</i>] <i>file ...</i> -lrpcsvc [<i>library ...</i>] #include <rpc/rpc.h> #include <rpcsvc/rwall.h> enum clnt_stat rwall(char *host, char *msg);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>These routines require that the rpc.rwalld(1M) daemon be configured and available on the remote system indicated by <i>host</i>.</p> <p>rwall() executes wall(1M) on <i>host</i>. The rpc.rwalld process on <i>host</i> prints <i>msg</i> 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).</p>
SEE ALSO	rpc.rwalld(1M) , wall(1M) , rpc_clnt_calls(3N)

NAME	rwlock, rwlock_init, rwlock_destroy, rw_rdlock, rw_wrlock, rw_tryrdlock, rw_trywrlock, rw_unlock – multiple readers, single writer locks
SYNOPSIS	<pre>#include <synch.h> int rwlock_init(rwlock_t *rwlp, int type, void * arg); int rwlock_destroy(rwlock_t *rwlp); int rw_rdlock(rwlock_t *rwlp); int rw_wrlock(rwlock_t *rwlp); int rw_unlock(rwlock_t *rwlp); int rw_tryrdlock(rwlock_t *rwlp); int rw_trywrlock(rwlock_t *rwlp);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>Multiple readers, single writer (readers/writer) locks allow many threads to have simultaneous read-only access to data while allowing only one thread to have write access at any given time. They are typically used to protect data that is searched more frequently than it is changed.</p> <p>Readers/writer locks can be used to synchronize threads in this process and other processes if they are allocated in memory that is writable and shared among the cooperating processes (see mmap(2)) and have been initialized for this behavior.</p> <p>Readers/writer locks must be initialized before use. rwlock_init() initializes the readers/writer lock pointed to by <i>rwlp</i>. A readers/writer lock can potentially have several different types of behavior, specified by <i>type</i>. No current type uses <i>arg</i> although a future type may specify additional behavior parameters via <i>arg</i>. <i>type</i> may be one of the following:</p> <p>USYNC_PROCESS The readers/writer lock can be used to synchronize threads in this process and other processes. Only one process should initialize the readers/writer lock. <i>arg</i> is ignored.</p> <p>USYNC_THREAD The readers/writer lock can be used to synchronize threads in this process, only. <i>arg</i> is ignored.</p> <p>Readers/writer locks may also be initialized by allocation in zeroed memory. In this case a <i>type</i> of USYNC_THREAD is assumed. Multiple threads must not initialize the same readers/writer lock simultaneously. A readers/writer lock must not be re-initialized while other threads may be using it.</p> <p>rwlock_destroy() destroys any state associated with the readers/writer lock pointed to by <i>rwlp</i>. The space for storing the readers/writer lock is not freed.</p> <p>rw_rdlock() acquires a read lock on the readers/writer lock pointed to by <i>rwlp</i>. If the readers/writer lock is already locked for writing, the calling thread blocks until the write lock is released. More than one thread may hold a read lock on a readers/writer lock at any one time.</p>

rw_tryrdlock() attempts to acquire a read lock on the readers/writer lock pointed to by *rwlp*. If the readers/writer lock is already locked for writing, it returns an error. Otherwise the read lock is acquired.

rw_wrlock() acquires a write lock on the readers/writer lock pointed to by *rwlp*. If the readers/writer lock is already locked for reading or writing, the calling thread blocks until all the read locks and write locks are released. Only one thread may hold a write lock on a readers/writer lock at any one time.

rw_trywrlock() attempts to acquire a write lock on the readers/writer lock pointed to by *rwlp*. If the readers/writer lock is already locked for reading or writing, it returns an error.

rw_unlock() unlocks a readers/writer lock pointed to by *rwlp*. The readers/writer lock must be locked and the calling thread must hold the lock either for reading or writing. If any other threads are waiting for the readers/writer lock to become available, one of them is unblocked. If the calling thread does not hold the lock for either reading or writing, no error status is returned, and the behavior of the program is undefined.

RETURN VALUES

Zero is returned when successful. A non-zero value indicates an error.

ERRORS

If any of the following conditions are detected, these functions fail and return the corresponding value:

EINVAL Invalid argument.

EFAULT *rwlp* or *arg* point to an illegal address.

If any of the following conditions are detected, **rw_tryrdlock()** or **rw_trywrlock()** fails and returns the corresponding value:

EBUSY The readers/writer lock pointed to by *rwlp* was already locked.

SEE ALSO

mmap(2)

NOTES

These interfaces also available via:

#include <thread.h>

By default, there is no defined order of acquisition if multiple threads are waiting for a readers/writer lock. However, implementations usually bias acquisition order in some way so as to avoid writer starvation. The current implementation tends to favor writers over readers.

NAME	scandir, alphasort – scan a directory
SYNOPSIS	<pre>/usr/ucb/cc [flag ...] file ... #include <sys/types.h> #include <sys/dir.h> int scandir(<i>dirname</i>, <i>namelist</i>, <i>select</i>, <i>dcomp</i>) char *<i>dirname</i>; struct direct *(*<i>namelist</i>[]); int (*<i>select</i>)(.), (*<i>dcomp</i>)(); int alphasort(<i>d1</i>, <i>d2</i>) struct direct **<i>d1</i>, **<i>d2</i>;</pre>
DESCRIPTION	<p>scandir() 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. alphasort() is a routine that sorts the array alphabetically.</p> <p>scandir() returns the number of entries in the array and a pointer to the array through the parameter <i>namelist</i>.</p>
RETURN VALUES	Returns -1 if the directory cannot be opened for reading or if malloc(3C) cannot allocate enough memory to hold all the data structures.
SEE ALSO	getdents(2) , directory(3C) , malloc(3C) , qsort(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	scanf, fscanf, sscanf – convert formatted input
SYNOPSIS	<pre>#include <stdio.h> int scanf(const char *format, ...); int fscanf(FILE *strm, const char *format, ...); int sscanf(const char *s, const char *format, ...);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>scanf() reads from the standard input stream, stdin.</p> <p>fscanf() reads from the stream <i>strm</i>.</p> <p>sscanf() reads from the character string <i>s</i>.</p> <p>Each function reads characters, 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 pointer arguments indicating where the converted input should be stored. If there are insufficient arguments for the format, the behavior is undefined. If the format is exhausted while arguments remain, the excess arguments are simply ignored.</p> <p>The control string usually contains conversion specifications, which are used to direct interpretation of input sequences. The control string may contain:</p> <ol style="list-style-type: none"> 1. White-space characters (blanks, tabs, new-lines, or form-feeds) that, except in two cases described below, cause input to be read up to the next non-white-space character. 2. An ordinary character (not %) that must match the next character of the input stream. 3. Conversion specifications consisting of the character % or the character sequence %<i>digits</i>%, an optional assignment suppression character *, a decimal digit string that specifies an optional numerical maximum field width, an optional letter I (ell), L, or h indicating the size of the receiving object, and a conversion code: % or <i>digit</i>, *, <i>decimal digit string</i>, h or l or L, <i>conversion code</i> <p>The following defines which size indicators can be used with which conversion codes, and the size they indicate.</p>

Conversion Code	Size Indicator	Size
d, i, n	none	int
	h	short int
	l	long int
o, u, x	none	unsigned int
	h	unsigned short int
	l	unsigned long int
e, f, g	none	float
	l	double
	L	long double

The **h**, **l**, or **L** modifier is ignored with any other conversion codes.

A conversion specification directs the conversion of the next input field; the result is placed in the variable pointed to by the corresponding argument unless assignment suppression was indicated by the character `*`. The suppression of assignment provides a way of describing an input field that is to be skipped. An input field is defined as a string of non-space characters; it extends to the next inappropriate character or until the maximum field width, if one is specified, is exhausted. For all descriptors except the character `[` and the character `c`, white space leading an input field is ignored.

Conversions can be applied to the *n*th argument in the argument list, rather than to the next unused argument. In this case, the conversion character `%` (see above) is replaced by the sequence `%digits$` where *digits* is a decimal integer *n*, giving the position of the argument in the argument list. The first such argument, `%1$`, immediately follows *format*. The control string can contain either form of a conversion specification, that is, `%` or `%digits$`, although the two forms cannot be mixed within a single control string.

The conversion code indicates the interpretation of the input field; the corresponding pointer argument must usually be of a restricted type. For a suppressed field, no pointer argument is given. The following conversion codes are valid:

- %** A single `%` is expected in the input at this point; no assignment is done.
- d** Matches an optionally signed decimal integer, whose format is the same as expected for the subject sequence of the `strtol()` function with the value 10 for the *base* argument. The corresponding argument should be a pointer to integer.
- u** Matches an optionally signed decimal integer, whose format is the same as expected for the subject sequence of the `strtoul()` function (see `strtol(3C)`) with the value 10 for the *base* argument. The corresponding argument should be a pointer to unsigned integer.
- o** Matches an optionally signed octal integer, whose format is the same as expected for the subject sequence of the `strtoul()` function with the value 8 for the *base* argument. The corresponding argument should be a pointer to unsigned integer.
- x** Matches an optionally signed hexadecimal integer, whose format is the

same as expected for the subject sequence of the **strtoul()** function with the value 16 for the *base* argument. The corresponding argument should be a pointer to unsigned integer.

- i** Matches an optionally signed integer, whose format is the same as expected for the subject sequence of the **strtol()** function with the value 0 for the *base* argument. The corresponding argument should be a pointer to integer.
- n** No input is consumed. The corresponding argument should be a pointer to integer into which is to be written the number of characters read from the input stream so far by the call to the function. Execution of a **%n** directive does not increment the assignment count returned at the completion of execution of the function.
- e,f,g** Matches an optionally signed floating point number, whose format is the same as expected for the subject string of the **strtod** function. The corresponding argument should be a pointer to floating.
- s** A character string is expected; the corresponding argument should be a character pointer pointing to an array of characters large enough to accept the string and a terminating **\0**, which will be added automatically. The input field is terminated by a white-space character.
- ws** A wide character string is expected; the corresponding argument should be a wide character pointer pointing to an array of wide characters large enough to accept the wide character string and a terminating **\0**, which will be added automatically. The input field is terminated by a white-space character.
- c** Matches a sequence of characters of the number specified by the field width (1 if no field width is present in the directive). The corresponding argument should be a pointer to the initial character of an array large enough to accept the sequence. No null character is added. The normal skip over white space is suppressed.
- wc** Matches a sequence of wide characters of the number specified by the field width (1 if no field width is present in the directive). The corresponding argument should be a pointer to the initial character of an array large enough to accept the sequence. No null character is added. The normal skip over white space is suppressed.
- [** Matches a nonempty sequence of characters from a set of expected characters (the *scanset*). The corresponding argument should be a pointer to the initial character of an array large enough to accept the sequence and a terminating null character, which will be added automatically. The conversion specifier includes all subsequent characters in the *format* string, up to and including the matching right bracket (**]**). The characters between the brackets (the *scanlist*) comprise the scanset, unless the character after the left 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 bracket. If the conversion specifier begins with **[** or **[^**, the right bracket

character is in the scanlist and the next right bracket character is the matching right bracket that ends the specification; otherwise the first right bracket character is the one that ends the specification.

A range of characters in the scanset may be represented by the construct *first – last*; thus **[0123456789]** may be expressed **[0–9]**. Using this convention, *first* must be lexically less than or equal to *last*, or else the dash will stand for itself. The character – will also stand for itself whenever it is the first or the last character in the scanlist. To include the right bracket as an element of the scanset, it must appear as the first character (possibly preceded by a circumflex) of the scanlist and in this case it will not be syntactically interpreted as the closing bracket. At least one character must match for this conversion to be considered successful.

- p** Matches the set of implementation-defined sequences produced as output by the **%p** conversion of the **printf(3S)** function. The corresponding argument should be a pointer to **void**. If the input item is a value converted earlier during the same program execution, the pointer that results compares equal to that value; otherwise, the behavior of the **%p** conversion is undefined.

If an invalid conversion character follows the %, the results of the operation may not be predictable.

The conversion specifiers **E**, **G**, and **X** are also valid and, under the **-Xa** and **-Xc** compilation modes (see **cc(1B)**), behave the same as **e**, **g**, and **x**, respectively. Under the **-Xt** compilation mode, **E**, **G**, and **X** behave the same as **le**, **lg**, and **lx**, respectively.

Each function allows for detection of a language-dependent decimal point character in the input string. The decimal point character is defined by the program's locale (category **LC_NUMERIC**). In the "C" locale, or in a locale where the decimal point character is not defined, the decimal point character defaults to a period (.).

The **scanf()** conversion terminates at end of file, at the end of the control string, or when an input character conflicts with the control string.

If end-of-file is encountered during input, conversion is terminated. If end-of-file occurs before any characters matching the current directive have been read (other than leading white space, where permitted), execution of the current directive terminates with an input failure; otherwise, unless execution of the current directive is terminated with a matching failure, execution of the following directive (if any) is terminated with an input failure.

If conversion terminates on a conflicting input character, the offending input character is left unread in the input stream. Trailing white space (including new-line characters) is left unread unless matched by a directive. The success of literal matches and suppressed assignments is not directly determinable other than via the **%n** directive.

LC_NUMERIC

Determines how numeric formats are handled. In the "C" locale, numeric

handling follows the U.S. rules.

RETURN VALUES

These routines return the number of successfully matched and assigned input items; this number can be 0 in the event of an early matching failure between an input character and the control string. If the input ends before the first matching failure or conversion, EOF is returned.

EXAMPLES

The call to the function `scanf()`:

```
int i, n; float x; char name[50];
n = scanf ("%d%f%s", &i, &x, name);
```

with the input line:

```
25 54.32E-1 thompson
```

will assign to `n` the value 3, to `i` the value 25, to `x` the value 5.432, and `name` will contain `thompson\0`.

The call to the function `scanf()`:

```
int i; float x; char name[50];
(void) scanf ("%2d%f%*d %[0-9]", &i, &x, name);
```

with the input line:

```
56789 0123 56a72
```

will assign 56 to `i`, 789.0 to `x`, skip 0123, and place the characters 56\0 in `name`. The next character read from `stdin` will be a.

FILES

`/usr/lib/locale/locale/LC_NUMERIC/numeric`
LC_NUMERIC database for *locale*

SEE ALSO

`cc(1B)`, `strtod(3C)`, `strtol(3C)`, `printf(3S)`

NAME	sched_get_priority_max, sched_get_priority_min, sched_rr_get_interval – get scheduling parameter limits
SYNOPSIS	<pre>cc [flag ...] file ... -lposix4 [library ...] #include <sched.h> int sched_get_priority_max(int policy); int sched_get_priority_min(int policy); int sched_rr_get_interval(pid_t pid, struct timespec *interval); struct timespec { time_t tv_sec; /* seconds */ long tv_nsec; /* and nanoseconds */ };</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>sched_get_priority_max() and sched_get_priority_min() return the appropriate maximum or minimum values, respectively, for the scheduling policy specified by <i>policy</i>.</p> <p>sched_rr_get_interval() updates the timespec structure referenced by <i>interval</i> to contain the current execution time limit (i.e., time quantum) for the process specified by <i>pid</i> under the SCHED_RR policy. After that time limit expires, when another process at the same priority is ready to execute, a scheduling decision will be made. If <i>pid</i> is zero, the current execution time limit for the calling process is stored in <i>interval</i>.</p> <p>The value of <i>policy</i> must be one of the scheduling policy values defined in <sched.h>: SCHED_FIFO, SCHED_RR, or SCHED_OTHER.</p>
RETURN VALUES	<p>If successful, sched_get_priority_max() or sched_get_priority_min() returns the appropriate maximum or minimum values, respectively.</p> <p>If successful, sched_rr_get_interval() returns 0.</p> <p>If unsuccessful, these functions return -1, and set errno to indicate the error condition.</p>
ERRORS	<p>EINVAL The value of <i>policy</i> does not represent a defined scheduling policy.</p> <p>ENOSYS sched_get_priority_max(), sched_get_priority_min(), and sched_rr_get_interval() are not supported by this implementation.</p> <p>ESRCH No process can be found corresponding to that specified by <i>pid</i>.</p>
SEE ALSO	sched_setparam(3R) , sched_setscheduler(3R)
BUGS	In Solaris 2.4, these functions always return -1 and set errno to ENOSYS , because this release does not support the Priority Scheduling option. It is our intention to provide support for these interfaces in future releases.

NAME	sched_setparam, sched_getparam – set/get scheduling parameters
SYNOPSIS	<pre>cc [flag ...] file ... -lposix4 [library ...] #include <sched.h> int sched_setparam(pid_t pid, const struct sched_param *param); int sched_getparam(pid_t pid, struct sched_param *param); struct sched_param { int sched_priority; /* process execution scheduling priority */ ... }</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>sched_setparam() sets the scheduling parameters of the process specified by <i>pid</i> to the values specified by the sched_param structure referenced by <i>param</i>.</p> <p>sched_getparam() stores the scheduling parameters of a process, specified by <i>pid</i>, in the sched_param structure pointed to by <i>param</i>.</p> <p>If the target process has as its scheduling policy, SCHED_FIFO or SCHED_RR:</p> <p>If <i>pid</i> is zero, the scheduling parameters are set/stored for the calling process. Otherwise, if a process specified by <i>pid</i> exists and if the calling process has permission, the scheduling parameters are set/stored 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>The target process, <i>pid</i>, 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, <i>pid</i>, is set higher than that of the lowest priority running process, and if process <i>pid</i> is ready to run, then process <i>pid</i> 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 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>The value of <i>param</i>->sched_priority must be an 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.</p>
RETURN VALUES	If successful, sched_setparam() and sched_getparam() returns 0; otherwise, the priority remains unchanged, the function returns -1, and sets errno to indicate the error condition.

ERRORS

- EINVAL** One or more of **sched_setparam()**'s requested scheduling parameters is outside the range defined for the specified *pid*'s scheduling policy.
- ENOSYS** **sched_setparam()** and **sched_getparam()** are not supported by this implementation.
- EPERM** The requesting process does not have permission to set/get 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 *pid*.

SEE ALSO

intro(2), **exec(2)**, **sched_setscheduler(3R)**

BUGS

In Solaris 2.4, these functions always return **-1** and set **errno** to **ENOSYS**, because this release does not support the Priority Scheduling option. It is our intention to provide support for these interfaces in future releases.

NAME	sched_setscheduler, sched_getscheduler – set/get scheduling policy and scheduling parameters
SYNOPSIS	<pre>cc [flag ...] file ... -lposix4 [library ...] #include <sched.h> int sched_setscheduler(pid_t pid, int policy, const struct sched_param *param); int sched_getscheduler(pid_t pid); struct sched_param { int sched_priority; /* process execution scheduling priority */ ... } </pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>sched_setscheduler() 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 sched_param structure pointed to by <i>param</i>, respectively. The value of <i>param->sched_priority</i> must be any integer within the inclusive priority range for the scheduling policy specified by <i>policy</i>. The possible values for the <i>policy</i> parameter are defined in the header file <sched.h>: SCHED_FIFO, SCHED_RR, or SCHED_OTHER.</p> <p>If <i>pid</i> is zero, the scheduling policy and scheduling parameters are set for the calling process. Otherwise, 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 super-user. See intro(2).</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>sched_getscheduler() returns the scheduling policy of the process specified by <i>pid</i>. If <i>pid</i> is zero, the scheduling policy is returned for the calling process. Otherwise, if a process specified by <i>pid</i> exists and if the calling process has permission, the scheduling policy is returned for the process whose process ID is equal to <i>pid</i>.</p>
RETURN VALUES	<p>If successful, sched_setscheduler() returns the former scheduling policy of the specified process (<i>pid</i>), which will be one of the following values:</p> <ul style="list-style-type: none"> 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.

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.

or

SCHED_OTHER (time-sharing).

Otherwise, the policy and scheduling parameters remain unchanged, **sched_setscheduler()** returns **-1**, and sets **errno** to indicate the error condition.

If successful, **sched_getscheduler()** returns the scheduling policy of the specified process; otherwise, it returns **-1**, and sets **errno** to indicate the error condition.

ERRORS

- EINVAL** The value of *policy* is invalid, or one or more of the parameters contained in **param** is outside the valid range for the specified scheduling policy.
- ENOSYS** **sched_setscheduler()** and **sched_getscheduler()** are not supported by this implementation.
- EPERM** **sched_setscheduler()** does not have permission to set either or both of the scheduling parameters or the scheduling policy of the specified process.
- sched_getscheduler()** 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*.

SEE ALSO

priocntl(1), **intro(2)**, **exec(2)**, **priocntl(2)**, **sched_get_priority_max(3R)**, **sched_setparam(3R)**

BUGS

In Solaris 2.4, these functions always return **-1** and set **errno** to **ENOSYS**, because this release does not support the Priority Scheduling option. It is our intention to provide support for these interfaces in future releases.

NAME	sched_yield – yield processor
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lposix4 [<i>library</i> ...] #include <sched.h> int sched_yield(void);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	sched_yield() forces the running process to relinquish the processor until the process again becomes the head of its process list.
RETURN VALUES	If successful, sched_yield() returns 0 , otherwise, it returns -1 , and sets errno to indicate the error condition.
ERRORS	ENOSYS sched_yield() is not supported by this implementation.
BUGS	In Solaris 2.4, these functions always return -1 and set errno to ENOSYS , because this release does not support the Priority Scheduling option. It is our intention to provide support for these interfaces in future releases.

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
MT-LEVEL	MT-Safe
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> <ul style="list-style-type: none"> AUTH_NONE (none) no authentication. AUTH_SYS Traditional UNIX-style authentication. AUTH_DES DES encryption-based authentication. AUTH_KERB Kerberos encryption-based authentication. <p>The authdes_getucred() and authdes_seccreate() routines implement the AUTH_DES authentication flavor. The keyserver daemon keyerv (see keyerv(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> int authdes_getucred(const struct authdes_cred *adc, uid_t *uidp, gid_t *gidp, short *gidlenp, gid_t *gidlist);</pre> <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 the other groups to which the user belongs. <i>*gidlenp</i> is set to the number of valid group ID entries in <i>*gidlist</i> (see netname2user(), below).</p>

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 unsigned int 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.

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[NGROUPS]);
```

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()**.

SEE ALSO

chkey(1), keyserv(1M), newkey(1M), kerberos_rpc(3N), rpc(3N), rpc_clnt_auth(3N)

NAME	select – synchronous I/O multiplexing
SYNOPSIS	<pre>#include <sys/time.h> #include <sys/types.h> int select(int nfds, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, 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>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>select() examines the I/O file descriptor sets whose addresses are passed in <i>readfds</i>, <i>writefds</i>, and <i>exceptfds</i> to see if any of their file descriptors are ready for reading, are ready for writing, or have an exceptional condition pending, respectively. Out-of-band data is the only exceptional condition. <i>nfds</i> is the number of bits to be checked in each bit mask that represents a file descriptor; the file descriptors from 0 to <i>nfds</i> – 1 in the file descriptor sets are examined. On return, select() replaces the given file descriptor sets with subsets consisting of those file descriptors that are ready for the requested operation. The return value from the call to select() is the number of ready file descriptors.</p> <p>The file descriptor sets are stored as bit fields in arrays of integers. The following macros are provided for manipulating such file descriptor sets: FD_ZERO() initializes a file descriptor set <i>fdset</i> to the null set. FD_SET() includes a particular file descriptor <i>fd</i> in <i>fdset</i>. FD_CLR() removes <i>fd</i> from <i>fdset</i>. FD_ISSET() is nonzero if <i>fd</i> is a member of <i>fdset</i>, zero otherwise. The behavior of these macros is undefined if a file descriptor value is less than zero or greater than or equal to FD_SETSIZE. FD_SETSIZE is a constant defined in <code><sys/select.h></code>.</p> <p>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 timeval structure.</p> <p>Any of <i>readfds</i>, <i>writefds</i>, and <i>exceptfds</i> may be given as NULL pointers if no file descriptors are of interest.</p>
RETURN VALUES	select() returns the number of ready file descriptors contained in the file descriptor sets or -1 if an error occurred. If the time limit expires, then select() returns 0 .
ERRORS	<p>The call fails if:</p> <p>EBADF One of the I/O file descriptor sets specified an invalid I/O file descriptor.</p>

EINTR A signal was delivered before any of the selected events occurred, or the time limit expired.

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 .

SEE ALSO **poll(2)**, **read(2)**, **write(2)**

NOTES The default value for **FD_SETSIZE** (currently 1024) is larger than the default limit on the number of open files. In order to accommodate programs that may use a larger number of open files with **select()**, it is possible to increase this size within a program by providing a larger definition of **FD_SETSIZE** before the inclusion of **<sys/types.h>**.
The file descriptor sets are always modified on return, even if the call returns as the result of a timeout.

NAME	sem_close – close a named semaphore
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lposix4 [<i>library</i> ...] #include <semaphore.h> int sem_close(sem_t *sem); typedef struct { ... } sem_t; /*opaque POSIX.4 semaphore*/</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>sem_close() is used to indicate that the calling process is finished using the named semaphore <i>sem</i>. sem_close() deallocates any system resources for use by this process for this semaphore. 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 sem_unlink(3R) 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 will no longer be accessible. sem_close() should not be called for an unnamed semaphore initialized by sem_init(3R).</p>
RETURN VALUES	If successful, sem_close() returns 0 , otherwise it returns -1 and sets errno to indicate the error condition.
ERRORS	<p>EINVAL <i>sem</i> is not a valid semaphore descriptor. ENOSYS sem_close() is not supported by this implementation.</p>
SEE ALSO	sem_init(3R) , sem_open(3R) , sem_unlink(3R)
BUGS	In Solaris 2.4, these functions always return -1 and set errno to ENOSYS , because this release does not support the Semaphores option. It is our intention to provide support for these interfaces in future releases.

NAME	sem_destroy – destroy an unnamed semaphore
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lposix4 [<i>library</i> ...] #include <semaphore.h> int sem_destroy(sem_t *sem); typedef struct { ... } sem_t; /*opaque POSIX.4 semaphore*/</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	sem_destroy() is used to destroy the unnamed semaphore, <i>sem</i> , which was initialized by sem_init(3R) .
RETURN VALUES	If successful, sem_destroy() returns 0 , otherwise it returns -1 and sets errno to indicate the error condition.
ERRORS	EINVAL <i>sem</i> is not a valid semaphore. ENOSYS sem_destroy() is not supported by this implementation. EBUSY Other processes (or LWPs or threads) are currently blocked on the semaphore.
SEE ALSO	sem_init(3R) , sem_open(3R)
BUGS	In Solaris 2.4, these functions always return -1 and set errno to ENOSYS , because this release does not support the Semaphores option. It is our intention to provide support for these interfaces in future releases.

NAME	sem_getvalue – get the value of a semaphore
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lposix4 [<i>library</i> ...] #include <semaphore.h> int sem_getvalue(sem_t *sem, int *sval); typedef struct { ... } sem_t; /*opaque POSIX.4 semaphore*/</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>sem_getvalue() updates the location referenced by <i>sval</i> to have the value of the semaphore referenced by <i>sem</i> without affecting the state of the semaphore. The updated value represents an actual semaphore value that occurred at some unspecified time during the call to sem_getvalue(), but may not be the actual value of the semaphore when sem_getvalue() is returned to the caller.</p> <p>The value set in <i>sval</i> may be zero or positive. If <i>sval</i> is zero, there may be other processes (or LWPs or threads) waiting for the semaphore; if <i>sval</i> is positive, no one is waiting.</p>
RETURN VALUES	If successful, sem_getvalue() returns 0, otherwise, it returns -1, and sets errno to indicate the error condition.
ERRORS	EINVAL <i>sem</i> does not refer to a valid semaphore. ENOSYS sem_getvalue() is not supported by this implementation.
SEE ALSO	sem_post(3R) , sem_wait(3R)
BUGS	In Solaris 2.4, these functions always return -1 and set errno to ENOSYS , because this release does not support the Semaphores option. It is our intention to provide support for these interfaces in future releases.

NAME	sem_init – initialize an unnamed semaphore
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lposix4 [<i>library</i> ...] #include <semaphore.h> int sem_init(sem_t *sem, int pshared, unsigned int value); typedef struct { ... } sem_t; /*opaque POSIX.4 semaphore*/</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>sem_init() is used to initialize the unnamed semaphore, referred to by <i>sem</i>, to <i>value</i>. This 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 <i>pshared</i> is non-zero, then the semaphore is sharable between processes. If the semaphore is not being shared between processes, the application should set <i>pshared</i> to 0.</p>
RETURN VALUES	If successful, sem_init() returns 0 and initializes the semaphore in <i>sem</i> ; otherwise it returns -1 and sets errno to indicate the error condition.
ERRORS	<p>EINVAL <i>value</i> exceeds SEM_VALUE_MAX.</p> <p>ENOSPC A resource required to initialize the semaphore has been exhausted.</p> <p> The resources have reached the limit on semaphores, SEM_NSEMS_MAX.</p> <p>ENOSYS sem_init() is not supported by this implementation.</p> <p>EPERM The calling process lacks the appropriate privileges to initialize the semaphore.</p>
SEE ALSO	sem_destroy(3R) , sem_post(3R) , sem_wait(3R)
BUGS	In Solaris 2.4, these functions always return -1 and set errno to ENOSYS , because this release does not support the Semaphores option. It is our intention to provide support for these interfaces in future releases.

NAME	sem_open – initialize/open a named semaphore
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lposix4 [<i>library</i> ...] #include <semaphore.h> sem_t *sem_open(const char *name, int oflag, /* unsigned long mode, unsigned int value */ ...); typedef struct { ... } sem_t; /*opaque POSIX.4 semaphore*/</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>sem_open() establishes a connection to a semaphore, <i>name</i>, returning the address of the semaphore to the calling process (or LWP or thread) for 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.</p> <p><i>name</i> points to a string naming a semaphore object. The <i>name</i> argument should conform to the construction rules for a pathname. If a process makes multiple successful calls to sem_open() with the same value for <i>name</i>, the same semaphore address will be returned for each such successful call, provided that there have been no calls to sem_unlink(3R) for this semaphore.</p> <p><i>oflag</i> determines whether the semaphore is created or merely accessed by the call to sem_open(). The three valid values for <i>oflag</i> are 0, O_CREAT, or the bitwise inclusive OR of O_CREAT and O_EXCL. Setting the <i>oflag</i> bits to O_CREAT will create the semaphore if it does not already exist. Setting both O_CREAT and O_EXCL will fail if the semaphore already exists. The check for the existence of the semaphore and the creation of the semaphore if it does not exist is atomic with respect to other processes executing sem_open(). After the semaphore named <i>name</i> has been created by sem_open() with the O_CREAT flag, other processes can connect to this semaphore by calling sem_open() with the same value of <i>name</i>, and nobits set in <i>oflag</i>.</p> <p>Using the O_CREAT flag requires a third and a fourth argument: <i>mode</i> and <i>value</i>. The semaphore is created with an initial count of <i>value</i>. <i>value</i> must be less than or equal to {SEM_VALUE_MAX}. The semaphore's user ID acquires the effective user ID of the process; the semaphore's group ID is set to a system default group ID or to the effective group ID of the process. The semaphore's permission bits is set to the value of <i>mode</i>, modified by clearing all bits set in the file creation mask of the process (see umask(2)).</p>
RETURN VALUES	If successful, sem_open() returns the address of the semaphore, otherwise it returns -1 and sets errno to indicate the error condition.
ERRORS	EACCES The named semaphore exists and the O_RDWR permissions are denied, or the named semaphore does not exist and permission to create the named semaphore is denied.

EEXIST	O_CREAT and O_EXCL are set and the named semaphore already exists.
EINTR	sem_open() was interrupted by a signal.
EINVAL	<i>name</i> is not a valid name. O_CREAT was set in <i>offlag</i> and <i>value</i> is greater than {SEM_VALUE_MAX} .
EMFILE	The number of open semaphore descriptors in this process exceeds {SEM_NSEMS_MAX} . The number of open file descriptors in this process exceeds {OPEN_MAX} .
ENAMETOOLONG	The string-length of <i>name</i> exceeds {PATH_MAX} , or a pathname component is longer than {NAME_MAX} while _POSIX_NO_TRUNC is in effect.
ENFILE	The system file table is full.
ENOENT	O_CREAT is not set and the named semaphore does not exist.
ENOSPC	There is insufficient space for the creation of the new named semaphore.
ENOSYS	sem_open() is not supported by this implementation.

SEE ALSO **exec(2)**, **exit(2)**, **umask(2)**, **sysconf(3C)**, **sem_close(3R)**, **sem_post(3R)**, **sem_unlink(3R)**, **sem_wait(3R)**

BUGS In Solaris 2.4, these functions always return **-1** and set **errno** to **ENOSYS**, because this release does not support the Semaphores option. It is our intention to provide support for these interfaces in future releases.

NAME	sem_post – increment the count of a semaphore
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lposix4 [<i>library</i> ...] #include <semaphore.h> int sem_post(sem_t *sem); typedef struct { ... } sem_t /*opaque POSIX.4 semaphore*/</pre>
MT-LEVEL	Async-Safe
DESCRIPTION	<p>If, prior to the call to sem_post(), the value of <i>sem</i> was 0, and other processes (or LWPs or threads) were blocked waiting for the semaphore, then one of them will be allowed to return successfully from its call to sem_wait(3R). The process to be unblocked will be chosen in a manner appropriate to the scheduling policies and parameters in effect for the blocked processes. In the case of the policies SCHED_FIFO and SCHED_RR, the highest priority waiting process is unblocked, and if there is more than one highest-priority process blocked waiting for the semaphore, then the highest priority process which has been waiting the longest is unblocked.</p> <p>If, prior to the call to sem_post(), no other processes (or LWPs or thread) were blocked for the semaphore, then its value is incremented by one.</p> <p>sem_post() is reentrant with respect to signals (ASYNC-SAFE), and may be invoked from a signal-catching function.</p>
RETURN VALUES	If successful, sem_post() returns 0 , otherwise it returns -1 , and sets errno to indicate the error condition.
ERRORS	<p>EINVAL <i>sem</i> does not refer to a valid semaphore.</p> <p>ENOSYS sem_post() is not supported by this implementation.</p>
EXAMPLES	(see sem_wait(3R))
SEE ALSO	sched_setscheduler(3R) , sem_wait(3R)
NOTES	sem_wait(3R) and sem_trywait(3R) decrement the semaphore upon their successful return.
BUGS	In Solaris 2.4, these functions always return -1 and set errno to ENOSYS , because this release does not support the Semaphores option. It is our intention to provide support for these interfaces in future releases.

NAME	sem_unlink – remove a named semaphore
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lposix4 [<i>library</i> ...] #include <semaphore.h> int sem_unlink(const char *name);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>sem_unlink() removes the semaphore named by the string <i>name</i>. If the semaphore, <i>name</i>, 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 exec(2). Calls to sem_open(3R) to re-create or re-connect to the semaphore will refer to a new semaphore after sem_unlink() is called. sem_unlink() does not block until all references have been destroyed; rather, it returns immediately.</p>
RETURN VALUES	If successful, sem_unlink() returns 0 ; otherwise, the function returns -1 , sets errno to indicate the error condition, and the semaphore is left unchanged.
ERRORS	<p>EACCES Permission is denied to unlink the named semaphore.</p> <p>ENAMETOOLONG The string-length of <i>name</i> exceeds {PATH_MAX}, or a pathname component is longer than {NAME_MAX} while _POSIX_NO_TRUNC is in effect.</p> <p>ENOENT The named semaphore does not exist.</p> <p>ENOSYS sem_unlink() is not supported by this implementation.</p>
SEE ALSO	exec(2) , exit(2) , sem_close(3R) , sem_open(3R)
BUGS	In Solaris 2.4, these functions always return -1 and set errno to ENOSYS , because this release does not support the Semaphores option. It is our intention to provide support for these interfaces in future releases.

NAME	sem_wait, sem_trywait – acquire or wait for a semaphore
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lposix4 [<i>library</i> ...] #include <semaphore.h> int sem_wait(sem_t *sem); int sem_trywait(sem_t *sem); typedef struct { ... } sem_t; /*opaque POSIX.4 semaphore*/</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>sem_wait() and sem_trywait() are the functions by which a calling thread waits or proceeds depending upon the state of a semaphore. A synchronizing process can proceed only if the value of the semaphore it accesses is currently greater than 0.</p> <p>If at the time of a call to either sem_wait() or sem_trywait(), the value of <i>sem</i> is positive, these functions decrement the value of the semaphore, return immediately, and allow the calling process to continue.</p> <p>If the semaphore's value is 0:</p> <ul style="list-style-type: none"> sem_wait() blocks, awaiting the semaphore to be released by another process (or LWP or thread). sem_trywait() fails, returning immediately.
RETURN VALUES	<p>If at the time of a call to either sem_wait() or sem_trywait(), the value of <i>sem</i> is positive, these functions return 0 on success. If the call was unsuccessful, the state of the semaphore is unchanged, the calling function returns -1, and sets errno to indicate the error condition.</p>
ERRORS	<p>EAGAIN The value of <i>sem</i> was 0 when sem_trywait() was called.</p> <p>EINVAL <i>sem</i> does not refer to a valid semaphore.</p> <p>EINTR sem_wait() was interrupted by a signal.</p> <p>ENOSYS sem_wait() and sem_trywait() are not supported by this implementation.</p> <p>EDEADLK A deadlock condition was detected; i.e., two separate processes are waiting for an available resource to be released via a semaphore "held" by the other process.</p>
EXAMPLES	<p>The customer waiting-line in a bank may be analogous to the synchronization scheme of a semaphore utilizing sem_wait() and sem_trywait():</p>

```

/* cc [ flag ... ] file ... -lposix4 -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_post(&bank_line); /* increment bank_line;
                          this_customer's teller
                          is now available */
}

```

SEE ALSO [sem_post\(3R\)](#)

NOTES [sem_wait\(\)](#) can be interrupted by a signal, which may result in its premature return. [sem_post\(3R\)](#) increments the semaphore upon its successful return.

BUGS

In Solaris 2.4, these functions always return `-1` and set `errno` to `ENOSYS`, because this release does not support the Semaphores option. It is our intention to provide support for these interfaces in future releases.

NAME	semaphore, sema_init, sema_destroy, sema_wait, sema_trywait, sema_post – semaphores
SYNOPSIS	<pre>#include <synch.h> int sema_init(sema_t *sp, unsigned int count, int type, void * arg); int sema_destroy(sema_t *sp); int sema_wait(sema_t *sp); int sema_trywait(sema_t *sp); int sema_post(sema_t *sp);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>Conceptually, a semaphore is a non-negative integer count. Semaphores are typically used to coordinate access to resources. The semaphore count is initialized to the number of free resources. Threads then atomically increment the count when resources are added and atomically decrement the count when resources are removed. When the semaphore count becomes zero, indicating no more resources are present, threads trying to decrement the semaphore will block until the count becomes greater than zero.</p> <p>Semaphores can be used to synchronize threads in this process and other processes if they are allocated in memory that is writable and is shared among the cooperating processes (see mmap(2)) and have been initialized for this behavior.</p> <p>Semaphores must be initialized before use. sema_init() initializes the semaphore pointed to by <i>sp</i> to <i>count</i>. A semaphore can potentially have several different types of behavior, specified by <i>type</i>. No current type uses <i>arg</i> although a future type may specify additional behavior parameters via <i>arg</i>. <i>type</i> may be one of the following:</p> <p>USYNC_PROCESS The semaphore can be used to synchronize threads in this process and other processes. Only one process should initialize the semaphore. <i>arg</i> is ignored.</p> <p>USYNC_THREAD The semaphore can be used to synchronize threads in this process, only. <i>arg</i> is ignored.</p> <p>Multiple threads must not initialize the same semaphore simultaneously. A semaphore must not be re-initialized while other threads may be using it.</p> <p>sema_destroy() destroys any state associated with the semaphore pointed to by <i>sp</i>. The space for storing the semaphore is not freed.</p> <p>sema_wait() blocks the calling thread until the count in the semaphore pointed to by <i>sp</i> becomes greater than zero and then atomically decrements it.</p> <p>sema_trywait() atomically decrements the count in the semaphore pointed to by <i>sp</i> if the count is greater than zero. Otherwise it returns an error.</p> <p>sema_post() atomically increments the count semaphore pointed to by <i>sp</i>. If there are any threads blocked on the semaphore, one is unblocked.</p>

RETURN VALUES

Zero is returned when successful. A non-zero value indicates an error.

ERRORS

If any of the following conditions are detected, these functions fail and return the corresponding value:

EINVAL Invalid argument.

EFAULT *sp* or *arg* points to an illegal address.

If any of the following conditions are detected, **sema_wait()** fails and returns the corresponding value:

EINTR The wait was interrupted by a signal or **fork()**.

If any of the following conditions are detected, **sema_trywait()** fails and returns the corresponding value:

EBUSY The semaphore pointed to by *sp* has a zero count.

SEE ALSO

mmap(2)

NOTES

These interfaces also available via:

#include <thread.h>

By default, there is no defined order of unblocking if multiple threads are waiting for a semaphore.

NAME	send, sendto, sendmsg – send a message from a socket
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lsocket -lnsl [<i>library</i> ...] #include <sys/types.h> #include <sys/socket.h> int send(int <i>s</i>, const char *<i>msg</i>, int <i>len</i>, int <i>flags</i>); int sendto(int <i>s</i>, const char *<i>msg</i>, int <i>len</i>, int <i>flags</i>, const struct sockaddr *<i>to</i>, int <i>tolen</i>); int sendmsg(int <i>s</i>, const struct msghdr *<i>msg</i>, int <i>flags</i>);</pre>
MT-LEVEL	Safe
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). 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. 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> <p>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.</p> <p>MSG_DONTROUTE The SO_DONTROUTE option is turned on for the duration of the operation. It is used only by diagnostic or routing programs.</p> <p>See recv(3N) for a description of the msghdr structure.</p>
RETURN VALUES	These calls return the number of bytes sent, or -1 if an error occurred.
ERRORS	<p>The calls fail if:</p> <p>EBADF <i>s</i> is an invalid file descriptor.</p> <p>EINTR The operation was interrupted by delivery of a signal before any data could be buffered to be sent.</p> <p>EINVAL <i>tolen</i> is not the size of a valid address for the specified address family.</p>

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.

SEE ALSO [fcntl\(2\)](#), [poll\(2\)](#), [write\(2\)](#), [connect\(3N\)](#), [getsockopt\(3N\)](#), [recv\(3N\)](#), [select\(3C\)](#), [socket\(3N\)](#)

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>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>setbuf() may be used after a <i>stream</i> (see intro(3)) has been 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 BUFSIZ, defined in the <code><stdio.h></code> header, indicates how large the array pointed to by <i>buf</i> should be.</p> <pre>char buf[BUFSIZ];</pre> <p>setvbuf() may be used after a stream has been opened but before it is read or written. <i>type</i> determines how <i>stream</i> will be buffered. Legal values for <i>type</i> (defined in <code><stdio.h></code>) are:</p> <p>_IOFBF causes input/output to be fully buffered.</p> <p>_IOLBF causes output to be line buffered; the buffer will be flushed when a newline is written, the buffer is full, or input is requested.</p> <p>_IONBF causes 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. <i>size</i> 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 zero.
SEE ALSO	fopen(3S) , getc(3S) , malloc(3C) , putc(3S) , stdio(3S)
NOTES	<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 BUFSIZ. If the array pointed to by <i>buf</i> is larger than BUFSIZ, a portion of <i>buf</i> will not be used. If <i>buf</i> is smaller than BUFSIZ, other memory may be unexpectedly overwritten.</p> <p>Parts of buf will be used for internal bookkeeping of the stream and, therefore, buf 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>

NAME	setbuffer, setlinebuf – assign buffering to a stream
SYNOPSIS	<pre> /usr/ucb/cc [flag ...] file ... #include <stdio.h> void setbuffer(<i>iop</i>, <i>abuf</i>, <i>asize</i>) register FILE *<i>iop</i>; char *<i>abuf</i>; int <i>asize</i>; int setlinebuf(<i>iop</i>) register FILE *<i>iop</i>; </pre>
DESCRIPTION	<p>setbuffer, setlinebuf – 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 up and written as a block; when it is line buffered characters are saved up until a NEWLINE is encountered or input is read from stdin. fflush(3S) 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) on the file. If the standard stream stdout refers to a terminal it is line buffered. The standard stream stderr is unbuffered by default.</p> <p>setbuffer() 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 BUFSIZ, defined in the <stdio.h> header, tells how big an array is needed:</p> <pre>char buf[BUFSIZ];</pre> <p>setlinebuf() 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 NULL.</p>
SEE ALSO	malloc(3C) , fclose(3S) , fopen(3S) , fread(3S) , getc(3S) , printf(3S) , putc(3S) , puts(3S) , setbuf(3S)
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>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>

NAME	setcat – define default catalog
SYNOPSIS	<pre>#include <pfmt.h> char *setcat(const char *catalog);</pre>
MT-LEVEL	MT-safe
DESCRIPTION	<p>The routine setcat() defines the default message catalog to be used by subsequent calls to pfmt(), pfmt() or gettext() which do not explicitly specify a message catalog. <i>catalog</i> 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). setcat() 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(), pfmt() or lfmt() that do not explicitly specify a catalog name will use Message not found!!\n as default string.</p>
RETURN VALUE	Upon success, setcat() returns a pointer to the catalog name. Upon failure, setcat() returns a NULL pointer.
EXAMPLE	<pre>setcat("test"); gettext(":10", "hello world\n")</pre>
SEE ALSO	gettext(3C) , lfmt(3C) , pfmt(3C) , setlocale(3C) , environ(5)

NAME	setjmp, longjmp, _setjmp, _longjmp – non-local goto
SYNOPSIS	<pre>/usr/ucb/cc [flag ...] file ... #include <setjmp.h> int setjmp(<i>env</i>) jmp_buf <i>env</i>; void longjmp(<i>env</i>, <i>val</i>) jmp_buf <i>env</i>; int <i>val</i>; int _setjmp(<i>env</i>) jmp_buf <i>env</i>; void _longjmp(<i>env</i>, <i>val</i>) jmp_buf <i>env</i>; int <i>val</i>;</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> <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 <i>val</i> to the function that invoked setjmp(); however, if <i>val</i> 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.</p> <p>setjmp() and longjmp() save and restore the signal mask while _setjmp() and _longjmp() manipulate only the C stack and registers.</p> <p>None of these functions save or restore any floating-point status or control registers.</p>

EXAMPLES

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 returning_from_longjump = 0;
    unsigned int alarmclock = 4;
    (void) signal(SIGINT, signal_handler);
    (void) signal(SIGALRM, signal_handler);
    alarm(alarmclock);
    while(1) {
        if((returning_from_longjump = setjmp(env)) != 0)
            switch(returning_from_longjump) {
                case SIGINT:
                    printf("longjumped with interrupt %d\n",SIGINT);
                    break;
                case SIGALRM:
                    printf("longjumped with alarm %d\n",SIGALRM);
                    break;
            }
        printf(" waiting for you to INTERRUPT (cntrl-C) ... \n");
        sleep(1);
    }
    /* end while forever loop */
}

static void signal_handler(sig)
int sig; {
    switch(sig) {
        case SIGINT:
            ... /* process for interrupt */
            longjmp(env,sig);
            /* break never reached */

        case SIGALRM:
            ... /* process for alarm */
            longjmp(env,sig);
            /* break never reached */

        default:
            exit(sig);
    }
}

```

If the **a.out** resulting from this C language code is run, and the user sends an interrupt signal in the first 4 seconds of execution, the output will be:

longjumped with interrupt

Otherwise, after approximately 4 seconds the output will be:

longjumped with 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>
MT-LEVEL	Unsafe
DESCRIPTION	<p>These functions 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().</p> <p>sigsetjmp() 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 prctl(2)) are also saved.</p> <p>longjmp() restores the environment saved by the last call of setjmp() with the corresponding <i>env</i> argument. After longjmp() is completed, program execution continues as if the corresponding call of setjmp() had just returned the value <i>val</i>. The caller of setjmp() must not have returned in the interim. longjmp() 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 example).</p> <p>siglongjmp() restores the environment saved by the last call of sigsetjmp() with the corresponding <i>env</i> argument. After siglongjmp() is completed, program execution continues as if the corresponding call of sigsetjmp() had just returned the value <i>val</i>. siglongjmp() 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() is called.</p> <p>If a signal-catching function interrupts sleep() 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> <p>The function siglongjmp() restores the saved signal mask if and only if the <i>env</i> argument was initialized by a call to the sigsetjmp() function with a non-zero <i>savemask</i> 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.</p>

EXAMPLES

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 returning_from_longjump = 0;
    unsigned int alarmclock = 4;
    (void) signal(SIGINT, signal_handler);
    (void) signal(SIGALRM, signal_handler);
    alarm(alarmclock);
    while(1) {
        if((returning_from_longjump = setjmp(env)) != 0)
            switch(returning_from_longjump) {
                case SIGINT:
                    printf("longjumped with interrupt %d\n",SIGINT);
                    break;
                case SIGALRM:
                    printf("longjumped with alarm %d\n",SIGALRM);
                    break;
            }
        printf(" waiting for you to INTERRUPT (cntrl-C) ... \n");
        sleep(1);
    }
    /* end while forever loop */
}

static void signal_handler(sig)
int sig; {
    switch(sig) {
        case SIGINT:
            ... /* process for interrupt */
            longjmp(env,sig);
            /* break never reached */

        case SIGALRM:
            ... /* process for alarm */
            longjmp(env,sig);
            /* break never reached */

        default:
            exit(sig);
    }
}

```

If the **a.out** resulting from this C language code is run, and the user sends an interrupt signal in the first 4 seconds of execution, the output will be:

longjumped with interrupt

Otherwise, after approximately 4 seconds the output will be:

longjumped with alarm

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.

SEE ALSO

getcontext(2), **priocntl(2)**, **sigaction(2)**, **sigaltstack(2)**, **sigprocmask(2)**, **signal(3C)**

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, absolute chaos is guaranteed.

NAME	setlabel – define the label for pfmt() and lfmt().
MT-LEVEL	MT-safe
SYNOPSIS	<pre>#include <pfmt.h> int setlabel(const char *label);</pre>
DESCRIPTION	<p>The routine setlabel() defines the label for messages produced in standard format by subsequent calls to pfmt() and lfmt().</p> <p><i>label</i> is a character string no more than 25 characters in length.</p> <p>No label is defined before setlabel() is called. A NULL pointer or an empty string passed as argument will reset the definition of the label.</p>
RETURN VALUE	setlabel() returns 0 in case of success, non-zero otherwise.
EXAMPLE	<p>The following code (without previous call to setlabel()):</p> <pre>pfmt(stderr, MM_ERROR, "test:2:Cannot open file\n"); setlabel("UX:test"); pfmt(stderr, MM_ERROR, "test:2:Cannot open file\n");</pre> <p>will produce the following output:</p> <pre>ERROR: Cannot open file UX:test: ERROR: Cannot open file</pre>
USAGE	The label should be set once at the beginning of a utility and remain constant.
SEE ALSO	getopt(3C) , lfmt(3C) , pfmt(3C)

NAME	setlocale – modify and query a program's locale																												
SYNOPSIS	#include <locale.h> char *setlocale(int category, const char *locale);																												
MT-LEVEL	Safe with exceptions																												
DESCRIPTION	<p>setlocale() 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 <locale.h> header. LC_ALL names all of a program's locale categories.</p> <p>LC_CTYPE affects the behavior of character handling functions such as isdigit() and tolower(), and multibyte character functions such as mbtowc() and wctomb().</p> <p>LC_NUMERIC affects the decimal point character and thousands separator character for the formatted input/output functions and string conversion functions.</p> <p>LC_TIME affects the date and time format as delivered by ascftime(), cftime(), getdate(), and strftime().</p> <p>LC_COLLATE affects the sort order produced by strcoll() and strxfrm().</p> <p>LC_MONETARY affects the monetary formatted information returned by localeconv().</p> <p>LC_MESSAGES affects the behavior of dgettext(), gettext(), and gettextt().</p> <p>Each category corresponds to a set of databases which contain the relevant information for each defined locale. The location of a database is given by the following path, /usr/lib/locale/locale/category, where <i>locale</i> and <i>category</i> are the names of locale and category, respectively. For example, the database for the LC_CTYPE category of the de (Deutsch or German) locale would be found in /usr/lib/locale/de/LC_CTYPE.</p> <p>A value of "C" for <i>locale</i> specifies the traditional UNIX system behavior. At program startup, the equivalent of</p> <pre style="margin-left: 40px;">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> <table border="0" style="margin-left: 40px;"> <thead> <tr> <th>Category</th> <th>1st Env. Var.</th> <th>2nd Env. Var.</th> <th>3rd Env. Var.</th> </tr> </thead> <tbody> <tr> <td>LC_CTYPE:</td> <td>LC_ALL</td> <td>LC_CTYPE</td> <td>LANG</td> </tr> <tr> <td>LC_COLLATE:</td> <td>LC_ALL</td> <td>LC_COLLATE</td> <td>LANG</td> </tr> <tr> <td>LC_TIME:</td> <td>LC_ALL</td> <td>LC_TIME</td> <td>LANG</td> </tr> <tr> <td>LC_NUMERIC:</td> <td>LC_ALL</td> <td>LC_NUMERIC</td> <td>LANG</td> </tr> <tr> <td>LC_MONETARY:</td> <td>LC_ALL</td> <td>LC_MONETARY</td> <td>LANG</td> </tr> <tr> <td>LC_MESSAGES:</td> <td>LC_ALL</td> <td>LC_MESSAGES</td> <td>LANG</td> </tr> </tbody> </table>	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_TIME:	LC_ALL	LC_TIME	LANG	LC_NUMERIC:	LC_ALL	LC_NUMERIC	LANG	LC_MONETARY:	LC_ALL	LC_MONETARY	LANG	LC_MESSAGES:	LC_ALL	LC_MESSAGES	LANG
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_TIME:	LC_ALL	LC_TIME	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. A composite locale is a string returned by a prior call to **setlocale(LC_ALL,0)**. This string will restore each category to the previous locale. If **setlocale()** fails to set the locale for any category, a null pointer is returned and the program's locale for all categories is not changed. Otherwise, *locale* is returned.

A null pointer for *locale* causes **setlocale()** to return the current locale associated with the *category*. The program's locale is not changed.

FILES	<i>/usr/lib/locale/locale/LC_CTYPE</i>	character type database for <i>locale</i>
	<i>/usr/lib/locale/locale/LC_NUMERIC</i>	numeric format data for <i>locale</i>
	<i>/usr/lib/locale/locale/LC_TIME</i>	date and time formats for <i>locale</i>
	<i>/usr/lib/locale/locale/LC_COLLATE</i>	sort ordering information for <i>locale</i>
	<i>/usr/lib/locale/locale/LC_MESSAGES</i>	message catalogs for <i>locale</i>
	<i>/usr/lib/locale/locale/LC_MONETARY</i>	currency format data for <i>locale</i>

SEE ALSO **ctype(3C)**, **localeconv(3C)**, **mbchar(3C)**, **strcoll(3C)**, **strftime(3C)**, **gettext(3I)**, **environ(5)**

NOTES To change locale in a multi-thread 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 multi-thread 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.

NAME	setregid – set real and effective group IDs
SYNOPSIS	<pre>/usr/ucb/cc [flag ...] file ...</pre> <p>int setregid(rgid, egid) int rgid, egid;</p>
DESCRIPTION	<p>setregid() is used to set the real and effective group IDs of the calling process. If <i>rgid</i> is -1, the real GID is not changed; if <i>egid</i> is -1, the effective GID is not changed. The real and effective GIDs may be set to different values in the same call.</p> <p>If the effective user ID of the calling process is super-user, the real GID and the effective GID can be set to any legal value.</p> <p>If the effective user ID of the calling process is not super-user, either the real GID can be set to the saved setGID from execv, or the effective GID can either be set to the saved setGID or the real GID. Note: if a setGID process sets its effective GID to its real GID, it can still set its effective GID back to the saved setGID.</p> <p>In either case, if the real GID is being changed (that is, if <i>rgid</i> is not -1), or the effective GID is being changed to a value not equal to the real GID, the saved setGID is set equal to the new effective GID.</p> <p>If the real GID is changed from its current value, the old value is removed from the groups access list (see getgroups(2)) if it is present in that list, and the new value is added to the groups access list if it is not already present and if this would not cause the number of groups in that list to exceed NGROUPS, as defined in /usr/include/sys/param.h.</p>
RETURN VALUES	Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and errno is set to indicate the error.
ERRORS	<p>setregid() will fail and neither of the group IDs will be changed if:</p> <p>EPERM The calling process's effective UID is not the super-user and a change other than changing the real GID to the saved setGID, or changing the effective GID to the real GID or the saved GID, was specified.</p>
SEE ALSO	exec(2) , getgroups(2) , getuid(2) , setuid(2) , setreuid(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	setreuid – set real and effective user IDs
SYNOPSIS	<pre>/usr/ucb/cc [flag ...] file ... int setreuid(ruid, euid) int ruid, euid;</pre>
DESCRIPTION	<p>setreuid() emulates the BSD setreuid() system call. However, exact BSD semantics are not maintained. Also, this routine cannot implement all cases that are possible with the BSD system call, and returns an error in those cases.</p> <p>If setreuid() is executed by root (the effective user ID is 0), and the application is either trying to change both the real user ID and effective user ID to the same value, or if the application is trying to change the real user ID to root, then setreuid() calls the setuid(2) system call with the <i>ruid</i> parameter, to change the real, effective, and saved user IDs to the new value.</p> <p>setreuid() detects this case in the following manner:</p> <p style="padding-left: 40px;">If the special cases for root (described above) are not met, setreuid() maps an <i>ruid</i> and/or <i>euid</i> parameter equaling -1 to the current real user ID or current effective user ID, respectively.</p> <p style="padding-left: 40px;">If the <i>ruid</i> parameter equals the current effective user ID, and the <i>euid</i> parameter equals the current real user ID; or if the <i>ruid</i> parameter equals the current real user ID, then this case is detected.</p> <p>setreuid() returns EPERM error in cases it cannot implement. These cases include: non-root user attempts to change the real user ID to the current effective user ID (that is, setreuid(y, -1) where <i>y</i> equals the current effective user ID).</p> <p>It is recommended that applications use one of the base system calls in place of setreuid(). If the application is executed as root, and intends to change its real and effective user IDs to give up root privilege, the recommended interface is setuid(2).</p>
RETURN VALUES	Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and errno is set to indicate the error.
ERRORS	<p>setreuid() will fail and neither of the user IDs will be changed if:</p> <p>EPERM The calling process's effective user ID is not the super-user and a change other than changing the real user ID to the effective user ID, or changing the effective user ID to the real user ID or the saved set-user ID, was specified.</p> <p>EPERM setreuid() detected it cannot handle this case.</p>
SEE ALSO	exec(2) , getuid(2) , setuid(2) , setregid(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	shm_open – open a shared memory object
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lposix4 [<i>library</i> ...] #include <sys/mman.h> int shm_open(const char *name, int oflag, mode_t mode);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>shm_open() either opens a file descriptor for the shared memory object with the name referenced by <i>name</i>. 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. Since the open file description is new, the new file descriptor is not as yet shared with any other processes.</p> <p>The file status flags and file access modes of the open file descriptor are set according to the value of <i>oflag</i>: the bitwise inclusive OR of the following flags, defined in the header <fcntl.h>. (Applications must specify exactly one of the first two values below in the value of <i>oflag</i>):</p> <ul style="list-style-type: none"> O_RDONLY Open for read access only. O_RDWR Open for read or write access. <p>Any combination of the remaining flags may be bitwise inclusive OR- ed with the value of <i>oflag</i>:</p> <ul style="list-style-type: none"> O_CREAT If <i>name</i> does not exist, the shared memory object is created, it's user ID is set to the effective user ID of the process, and it's group ID is set to a system default group ID or to the effective group ID of the process. The shared memory object's permission bits will be set to the value of <i>mode</i>, modified by clearing all bits set in the file mode creation mask of the process (see umask(2)). <p><i>mode</i> does not affect whether the shared memory object is opened for reading, for writing, or for both. The new shared memory object has a size of zero.</p> <p>If the shared memory object does exist, this flag will have no effect, except as specified under O_EXCL below.</p> <ul style="list-style-type: none"> O_EXCL If both OEXCL and O_CREAT are set, shm_open() fails if the shared memory object, <i>name</i>, 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 OEXCL and O_CREAT set.

	O_TRUNC	If the shared memory object exists, and it is successfully opened O_RDWR , the object is truncated to zero length and the mode and ownership are unchanged by this function call.
RETURN VALUES		If successful, shm_open() returns a nonnegative integer representing the lowest numbered unused file descriptor, otherwise it returns -1 and sets errno to indicate the error condition.
ERRORS	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	<i>name</i> is an invalid file description.
	EMFILE	The number of open file descriptors in this process exceeds {OPEN_MAX} .
	ENAMETOOLONG	The length of the <i>name</i> string exceeds {PATH_MAX} , or a pathname component is longer than {NAME_MAX} while _POSIX_NO_TRUNC is in effect.
	ENFILE	The system file table is full
	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	shm_open() is not supported by this implementation.
FILES		/usr/include/fcntl.h
SEE ALSO		close(2) , dup(2) , exec(2) , fcntl(2) , mmap(2) , umask(2) , sysconf(3C) , shm_unlink(3R) , fcntl(5)
NOTES		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.
BUGS		In Solaris 2.4, these functions always return -1 and set errno to ENOSYS , because this release does not support the Shared Memory Objects option. It is our intention to provide support for these interfaces in future releases.

NAME	shm_unlink – remove a shared memory object
SYNOPSIS	cc [<i>flag</i> ...] <i>file</i> ... -lposix4 [<i>library</i> ...] int shm_unlink(const char *name);
MT-LEVEL	MT-Safe
DESCRIPTION	shm_unlink() 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	If successful, shm_unlink() returns 0 , otherwise it returns -1 and sets errno to indicate the error condition, and the named shared memory object is not affected by this function.
ERRORS	EACCES Permission is denied to unlink the named shared memory object. ENAMETOOLONG The length of the <i>name</i> string exceeds {PATH_MAX} , or a pathname component is longer than {NAME_MAX} while _POSIX_NO_TRUNC is in effect. ENOENT The named shared memory object does not exist. ENOSYS shm_unlink() is not supported by this implementation.
SEE ALSO	close(2) , mmap(2) , mlock(3C) , shm_open(3R)
BUGS	In Solaris 2.4, these functions always return -1 and set errno to ENOSYS , because this release does not support the Shared Memory Objects option. It is our intention to provide support for these interfaces in future releases.

NAME	shutdown – shut down part of a full-duplex connection
SYNOPSIS	<code>cc [<i>flag</i> ...] <i>file</i> ... -lsocket -lnsl [<i>library</i> ...]</code> <code>int shutdown(int <i>s</i>, int <i>how</i>);</code>
MT-LEVEL	Safe
DESCRIPTION	The shutdown() call shuts down all or part of a full-duplex connection on the socket associated with <i>s</i> . If <i>how</i> is 0 , then further receives will be disallowed. If <i>how</i> is 1 , then further sends will be disallowed. If <i>how</i> 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: <ul style="list-style-type: none"> EBADF <i>s</i> 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 <i>s</i> is not a socket.
SEE ALSO	connect(3N) , socket(3N)
NOTES	The <i>how</i> values should be defined constants.

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 sigmask 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 errno to EINTR.</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 sigmask is provided to construct the mask for a given <i>signum</i>.</p> <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().</p> <p>It is not possible to block SIGKILL, SIGSTOP, or SIGCONT, this restriction is silently imposed by the system.</p>
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> sigfpe_handler_type sigfpe(sigfpe_code_type code, sigfpe_handler_type hdl);</pre>										
MT-LEVEL	Safe										
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> <table border="0"> <tr> <td style="padding-right: 10px;">FPE_FLTRES</td> <td>fp_inexact – floating-point inexact result</td> </tr> <tr> <td>FPE_FLTDIV</td> <td>fp_division – floating-point division by zero</td> </tr> <tr> <td>FPE_FLTUND</td> <td>fp_underflow – floating-point underflow</td> </tr> <tr> <td>FPE_FLTOVF</td> <td>fp_overflow – floating-point overflow</td> </tr> <tr> <td>FPE_FLTINV</td> <td>fp_invalid – floating-point invalid operation</td> </tr> </table> <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>	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
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										

EXAMPLES

A user-specified signal handler might look like this:

```
#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/signinfo.h

SEE ALSO sigaction(2), abort(3C), signal(3C), 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-thread applications is unsupported.</p> <p>This library routine uses an extension of the sigvec(3B) function that is not available in 4.2BSD, 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(7)). 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 †. Signals marked with ● are discarded if the action is SIG_DFL; signals marked with † 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. 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	<p>signal() will fail and no action will take place if the following occurs:</p> <p>EINVAL <i>sig</i> is not a valid signal number, or is SIGKILL or SIGSTOP.</p>

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(7)

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.

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>signal() and sigset() are used to modify signal dispositions. <i>sig</i> specifies the signal, which may be any signal except SIGKILL and SIGSTOP. <i>disp</i> 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>sighold() adds <i>sig</i> to the calling process's signal mask.</p> <p>sigrelse() removes <i>sig</i> from the calling process's signal mask.</p> <p>sigignore() sets the disposition of <i>sig</i> to SIG_IGN.</p> <p>sigpause() 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>On success, signal() returns the signal's previous disposition. On failure, it returns SIG_ERR and sets errno to indicate the error.</p> <p>On success, sigset() returns SIG_HOLD if the signal had been blocked or the signal's previous disposition if it had not been blocked. On failure, it returns SIG_ERR and sets errno to indicate the error.</p> <p>All other functions return zero on success. On failure, they return -1 and set errno to indicate the error.</p>

ERRORS

These functions fail if any of the following are true:

- EINTR** A signal was caught during the function **sigpause()**.
EINVAL The value of the *sig* argument is not a valid signal or is equal to **SIGKILL** or **SIGSTOP**.

SEE ALSO

kill(2), **pause(2)**, **sigaction(2)**, **sigsend(2)**, **wait(2)**, **waitid(2)**, **signal(5)**

NOTES

sighold() 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 a value of -1 with **errno** set to **ECHILD** (see **wait(2)**, **waitid(2)**).

NAME	sigqueue – queue a signal to a process
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lposix4 [<i>library</i> ...] #include <signal.h> int sigqueue(pid_t <i>pid</i>, int <i>signo</i>, const union sigval <i>value</i>); union sigval { int sival_int; /* integer value */ void *sival_ptr; /* pointer value */ };</pre>
MT-LEVEL	Async-Safe
DESCRIPTION	<p>sigqueue() causes the signal, <i>signo</i> to be sent with the value, <i>value</i> to the process, <i>pid</i>. If <i>signo</i> is zero (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 kill(2).</p> <p>If resources were not available to queue the signal, sigqueue() exits and returns immediately. If SA_SIGINFO is set for <i>signo</i> in the receiving process, and if the resources were available, the signal is left queued and pending. If SA_SIGINFO is not set for <i>signo</i>, then <i>signo</i> is sent at least once to the receiving process.</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, either <i>signo</i> or at least the pending, unblocked signal with the lowest number will be delivered to the sending process before sigqueue() returns.</p>
RETURN VALUES	If successful, sigqueue() returns 0 , and queues the specified signal. Otherwise, sigqueue() returns -1 and sets errno to indicate the error condition.
ERRORS	<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 sigqueue() is not supported by this implementation.</p> <p>EPERM The process does not have the appropriate privilege to send the signal to the receiving process.</p> <p>ESRCH The process <i>pid</i> does not exist.</p>
SEE ALSO	kill(2) , sigwaitinfo(3R) , 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>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>These functions manipulate <i>sigset_t</i> data types, representing the set of signals supported by the implementation.</p> <p>sigemptyset() initializes the set pointed to by <i>set</i> to exclude all signals defined by the system.</p> <p>sigfillset() initializes the set pointed to by <i>set</i> to include all signals defined by the system.</p> <p>sigaddset() adds the individual signal specified by the value of <i>signo</i> to the set pointed to by <i>set</i>.</p> <p>sigdelset() deletes the individual signal specified by the value of <i>signo</i> from the set pointed to by <i>set</i>.</p> <p>sigismember() 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 a value of one if the specified signal is a member of the specified set, or a value of 0 if it is not. Upon successful completion, the other functions return a value of 0. Otherwise a value of -1 is returned and errno is set to indicate the error.</p>
ERRORS	<p>sigaddset(), sigdelset(), and sigismember() will fail if the following is true:</p> <p>EINVAL The value of the <i>signo</i> argument is not a valid signal number.</p> <p>sigfillset() will fail if the following is true:</p> <p>EFAULT The <i>set</i> argument specifies an invalid address.</p>
SEE ALSO	sigaction(2) , sigpending(2) , sigprocmask(2) , sigsuspend(2) , 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>sigstack() 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>ss_sp 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. ss_onstack field is zero or non-zero depending on whether the process is currently executing on the signal stack or not.</p> <p>If <i>nss</i> is not a NULL pointer, sigstack() sets the signal stack state to the value in the sigstack() structure pointed to by <i>nss</i>. Note: if ss_onstack is non-zero, the system will think that the process is executing on the signal stack. If <i>nss</i> is a NULL pointer, the signal stack state will be unchanged. If <i>oss</i> is not a NULL pointer, the current signal stack state is stored in the sigstack() structure pointed to by <i>oss</i>.</p>
RETURN VALUES	Upon successful completion, a value of 0 is returned. Otherwise, a value of –1 is returned and errno is set to indicate the error.
ERRORS	<p>sigstack() will fail and the signal stack context will remain unchanged if one of the following occurs.</p> <p>EFAULT Either <i>nss</i> or <i>oss</i> points to memory that is not a valid part of the process address space.</p>
SEE ALSO	sigaltstack(2) , 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.

WARNINGS

Signal stacks are not “grown” automatically, as is done for the normal stack. If the stack overflows unpredictable results may occur.

NAME	sigvec – software signal facilities
SYNOPSIS	<pre> /usr/ucb/cc [flag ...] file ... #include <signal.h> int sigvec(sig, nvec, ovec) int sig; 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. 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> <p>The action to be taken when the signal is delivered is specified by a sigvec() structure, which includes the following members:</p> <pre> 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 */ </pre>

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 fcntl(2))
SIGXCPU		cpu time limit exceeded (see getrlimit(2))
SIGXFSZ		file size limit exceeded (see getrlimit(2))
SIGVTALRM		virtual time alarm (see setitimer() on getitimer(2))
SIGPROF		profiling timer alarm (see setitimer() on getitimer(2))

SIGWINCH	•	window changed (see termio(7))
SIGLOST	*	resource lost (see lockd(1M))
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	<p>sigvec() will fail and no new signal handler will be installed if one of the following occurs:</p> <p>EFAULT Either <i>nvec</i> or <i>ovec</i> is not a NULL pointer and points to memory that is not a valid part of the process address space.</p> <p>EINVAL <i>sig</i> is not a valid signal number, or, SIGKILL, or SIGSTOP.</p>
SEE ALSO	<p>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(7), termio(7)</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-thread applications is unsupported.</p> <p>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(7)). Otherwise, the process will never receive SIGPOLLs0.</p> <p>The handler routine can be declared:</p> <pre>void handler(int sig, int code, struct sigcontext *scp, char *addr);</pre> <p>Here <i>sig</i> is the signal number; <i>code</i> is a parameter of certain signals that provides additional detail; <i>scp</i> is a pointer to the sigcontext structure (defined in signal.h), used to restore the context from before the signal; and <i>addr</i> is additional address information.</p> <p>The signals SIGKILL, SIGSTOP, and SIGCONT cannot be ignored.</p>

NAME	sigwaitinfo, sigtimedwait – wait for queued signals
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lposix4 [<i>library</i> ...] #include <signal.h> int sigwaitinfo(const sigset_t *set, siginfo_t *info); int sigtimedwait(const sigset_t *set, siginfo_t *info, const struct timespec *timeout); typedef struct siginfo { int si_signo; /* signal from signal.h */ int si_code; /* code from above */ ... int si_value; ... } siginfo_t; struct timespec { time_t tv_sec; /* seconds */ long tv_nsec; /* and nanoseconds */ };</pre>
MT-LEVEL	Async-Safe
DESCRIPTION	<p>sigwaitinfo() and sigtimedwait() select the pending signal from the set specified by <i>set</i>. When multiple signals are pending, the lowest numbered one will be selected. The selection order between realtime and non-realtime signals, or between multiple pending non-realtime signals, is unspecified.</p> <p>If no signal in <i>set</i> is pending at the time of the call, sigwaitinfo() suspends the calling process until one or more signals in <i>set</i> become pending or until it is interrupted by an unblocked, caught signal. sigtimedwait(), on the other hand, suspends itself for the time interval specified in the timespec structure referenced by <i>timeout</i>. If the timespec 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 the error EAGAIN.</p> <p>If, while sigwaitinfo() or sigtimedwait() is waiting, a signal occurs which is eligible for delivery (i.e., not blocked by the process signal mask), that signal is handled asynchronously and the wait is interrupted.</p> <p>If <i>info</i> is non-NULL, the selected signal number is stored in si_signo, and the cause of the signal is stored in the si_code. If any value is queued to the selected signal, the first such queued value is dequeued and, if <i>info</i> is non-NULL, the value is stored in the si_value member of <i>info</i>. The system resource used to queue the signal is released and made available to queue other signals.</p> <p>If the value of the si_code member is SI_NOINFO, only the si_signo member of siginfo_t is meaningful, and the value of all other members is unspecified.</p> <p>If no further signals are queued for the selected signal, the pending indication for that</p>

signal is reset.

RETURN VALUES

If one of the signals specified by *set* is either pending or generated, **sigwaitinfo()** or **sigtimedwait()** returns the selected signal number. Otherwise, the function returns **-1** and sets **errno** to indicate the error condition.

ERRORS

EINTR The wait was interrupted by an unblocked, caught signal.

ENOSYS **sigwaitinfo()** or **sigtimedwait()** is not supported by this implementation.

The following errors relate to only **sigtimedwait()**:

EAGAIN No signal specified by *set* was delivered within the specified timeout period.

EINVAL *timeout* specified a **tv_nsec** value less than 0 or greater than 1,000,000,000.

SEE ALSO

time(2), **sigqueue(3R)**, **siginfo(5)**, **signal(5)**

NAME	sleep – suspend execution for interval
SYNOPSIS	<i>/usr/ucb/cc [flag ...] file ...</i> int sleep(seconds) unsigned seconds;
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>
SEE ALSO	alarm(2), getitimer(2), longjmp(3C), siglongjmp(3C), sleep(3C), usleep(3B)
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	<p>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.</p>

NAME	sleep – suspend execution for interval
SYNOPSIS	#include <unistd.h> unsigned sleep(unsigned seconds);
MT-LEVEL	Safe
DESCRIPTION	<p>The current process is suspended from execution for the number of <i>seconds</i> 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.</p> <p>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().</p>
SEE ALSO	alarm(2), pause(2), signal(3C)
NOTES	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() .

NAME	socket – create an endpoint for communication
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lsocket -lnsl [<i>library</i> ...] #include <sys/types.h> #include <sys/socket.h> int socket(int <i>domain</i>, int <i>type</i>, int <i>protocol</i>);</pre>
MT-LEVEL	Safe
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 <sys/socket.h>. There must be an entry in the netconfig(4) 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 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 SOCK_STREAM type provides sequenced, reliable, two-way connection-based byte streams. An out-of-band data transmission mechanism may be supported. A SOCK_DGRAM socket supports datagrams (connectionless, unreliable messages of a fixed (typically small) maximum length). A SOCK_SEQPACKET 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. SOCK_RAW sockets provide access to internal network interfaces. The types SOCK_RAW, which is available only to the super-user, and SOCK_RDM, 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 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</p>

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.

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), recvfrom(3N), setsockopt(3N), send(3N), shutdown(3N), socketpair(3N)

NAME	socketpair – create a pair of connected sockets												
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lsocket -lnsl [<i>library</i> ...] #include <sys/types.h> #include <sys/socket.h> int socketpair(int <i>domain</i>, int <i>type</i>, int <i>protocol</i>, int <i>sv</i>[2]);</pre>												
MT-LEVEL	Safe												
DESCRIPTION	The socketpair() library call creates an unnamed pair of connected sockets in the specified address family <i>d</i> , of the specified <i>type</i> , and using the optionally specified <i>protocol</i> . The descriptors used in referencing the new sockets are returned in <i>sv</i> [0] and <i>sv</i> [1]. The two sockets are indistinguishable.												
RETURN VALUES	socketpair() returns -1 on failure, and 0 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.												
SEE ALSO	pipe(2) , read(2) , write(2)												
NOTES	This call is currently implemented only for the AF_UNIX address family.												

NAME	spray – scatter data in order to test the network
SYNOPSIS	<pre>cc [flag ...] file ... -lsocket -lnsl [library ...] #include <rpcsvc/spray.h> bool_t xdr_sprayarr(XDR *xdrs, sprayarr *objp); bool_t xdr_spraycumul(XDR *xdrs, spraycumul *objp);</pre>
MT-LEVEL	Unsafe
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 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> <p>SPRAYPROC_CLEAR This procedure clears the counter.</p> <p>SPRAYPROC_SPRAY This procedure sends the packet.</p> <p>SPRAYPROC_GET This procedure returns the count and the amount of time since the last SPRAYPROC_CLEAR.</p>
EXAMPLES	<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 = (u_int)100; spray_data.sprayarr_val = buf; clnt = clnt_create("somehost", SPRAYPROC, SPRAYVERS, "netpath"); if (clnt == (CLIENT *)NULL) {</pre>

```

        /* handle this error */
    }
    if (clnt_call(clnt, SPRAYPROC_CLEAR,
        xdr_void, NULL, xdr_void, NULL, timeout25)) {
        /* handle this error */
    }
    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);

```

SEE ALSO [spray\(1M\)](#), [rpc_clnt_calls\(3N\)](#)

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, (udp for example). 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 (before the packets get to the network medium).

NAME	sqrt, cbrt – square root, cube root
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lm [<i>library</i> ...] #include <math.h> double sqrt(double x); double cbrt(double x);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>sqrt(x) returns the square root of <i>x</i>, correctly rounded according to ANSI/IEEE 754-1985.</p> <p>cbrt(x) returns the cube root of <i>x</i>. cbrt() is accurate to within 0.7 <i>ulps</i>.</p>
RETURN VALUES	For exceptional cases, matherr(3M) tabulates the values to be returned as dictated by various Standards.
SEE ALSO	matherr(3M)

NAME	ssignal, gsignal – software signals
SYNOPSIS	<pre>#include <signal.h> void (*ssignal(int sig, int (*action)(int))(int); int gsignal(int sig);</pre>
MT-LEVEL	Unsafe
DESCRIPTION	<p>ssignal() and gsignal() implement a software facility similar to signal(3C). This facility is made available to users for their own purposes.</p> <p>Software signals made available to users are associated with integers in the inclusive range 1 through 17. A call to ssignal() associates a procedure, <i>action</i>, with the software signal <i>sig</i>; the software signal, <i>sig</i>, is raised by a call to gsignal(). Raising a software signal causes the action established for that signal to be <i>taken</i>.</p> <p>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) <i>action function</i> or one of the manifest constants SIG_DFL (default) or SIG_IGN (ignore). ssignal() 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.</p> <p>gsignal() raises the signal identified by its argument, <i>sig</i>:</p> <ul style="list-style-type: none">If an action function has been established for <i>sig</i>, then that action is reset to SIG_DFL and the action function is entered with argument <i>sig</i>. gsignal() returns the value returned to it by the action function.If the action for <i>sig</i> is SIG_IGN, gsignal() returns the value 1 and takes no other action.If the action for <i>sig</i> is SIG_DFL, gsignal() returns the value 0 and takes no other action.If <i>sig</i> has an illegal value or no action was ever specified for <i>sig</i>, gsignal() returns the value 0 and takes no other action.
SEE ALSO	raise(3C) , signal(3C)

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() and putchar(), and the higher-level routines fgetc(), fgets(), fprintf(), fputc(), fputs(), fread(), fscanf(), fwrite(), gets(), getw(), printf(), puts(), putw(), and scanf() 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 FILE. fopen() 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 <stdio.h> header and associated with the standard open files:</p> <table border="0" style="margin-left: 40px;"> <tr> <td style="padding-right: 10px;">stdin</td> <td>standard input file</td> </tr> <tr> <td>stdout</td> <td>standard output file</td> </tr> <tr> <td>stderr</td> <td>standard error file</td> </tr> </table> <p>The following symbolic values in <unistd.h> define the file descriptors that will be associated with the C-language <i>stdin</i>, <i>stdout</i> and <i>stderr</i> when the application is started:</p> <table border="0" style="margin-left: 40px;"> <tr> <td style="padding-right: 20px;">STDIN_FILENO</td> <td style="padding-right: 20px;">Standard input value</td> <td style="padding-right: 20px;">0</td> <td>stdin</td> </tr> <tr> <td>STDOUT_FILENO</td> <td>Standard output value</td> <td>1</td> <td>stdout</td> </tr> <tr> <td>STDERR_FILENO</td> <td>Standard error value</td> <td>2</td> <td>stderr</td> </tr> </table> <p>The constant NULL designates a null pointer.</p> <p>The integer-constant EOF 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 BUFSIZ specifies the size of the buffers used by the particular implementation.</p> <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.</p> <p>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.</p> <p>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(),</p>	stdin	standard input file	stdout	standard output file	stderr	standard error file	STDIN_FILENO	Standard input value	0	stdin	STDOUT_FILENO	Standard output value	1	stdout	STDERR_FILENO	Standard error value	2	stderr
stdin	standard input file																		
stdout	standard output file																		
stderr	standard error file																		
STDIN_FILENO	Standard input value	0	stdin																
STDOUT_FILENO	Standard output value	1	stdout																
STDERR_FILENO	Standard error value	2	stderr																

getchar(), **putc()**, **putchar()**, **ferror()**, **feof()**, **clearerr()**, and **fileno()**. 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). **setbuf()** or **setvbuf()** (both described in **setbuf(3S)**) 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()**, **dup()**, **fdopen()**, **fileno()**, and **fork()** (which duplicates existing ones into new processes). They can be destroyed by at least **fclose()**, **close()**, and the **exec** functions (which close some file descriptors and destroy streams).

A file descriptor that is never used in an operation, that could affect the file offset (for example **read()**, **write()**, or **lseek()**) 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()** 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()** or **abort()**. A file descriptor is closed by **close()**, **_exit()**, 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()** 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()** 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()** (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, "world0);
fputc(iop, 'a');
funlockfile(iop);
```

will print everything out together, blocking other threads that might want to write to the same file between `fprintf`'s.

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	stdipc, ftok – standard interprocess communication package
SYNOPSIS	<pre>#include <sys/types.h> #include <sys/ipc.h> key_t ftok(const char *path, int id);</pre>
MT-LEVEL	See the NOTES section of this page.
DESCRIPTION	<p>Certain interprocess communication facilities require the user to supply a key to be used by the msgget(2), semget(2), and shmget(2) functions to obtain interprocess communication identifiers. One suggested method for forming a key is to use the ftok() subroutine described below. 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.</p> <p>ftok() returns a key based on <i>path</i> and <i>id</i> that is usable in subsequent msgget(), semget(), and shmget() functions. <i>path</i> must be the path name of an existing file that is accessible to the process. <i>id</i> is a character that uniquely identifies a project. Note that ftok() returns the same key for linked files when called with the same <i>id</i> and that it returns different keys when called with the same file name but different <i>ids</i>.</p>
RETURN VALUES	ftok() returns (key_t) -1 if <i>path</i> does not exist or if it is not accessible to the process.
SEE ALSO	intro(2) , msgget(2) , semget(2) , shmget(2)
NOTES	<p>If the file whose <i>path</i> is passed to ftok() is removed when keys still refer to the file, future calls to ftok() with the same <i>path</i> and <i>id</i> returns an error. If the same file is recreated, then ftok() is likely to return a different key from the original call.</p> <p>ftok() is MT-Safe in mutli-thread applications.</p>

NAME	str2sig, sig2str – translation between signal name and signal number
SYNOPSIS	<pre>#include <signal.h> int str2sig(const char *str, int *signum); int sig2str(int signum, char *str);</pre>
DESCRIPTION	<p>The str2sig() function translates the signal name <i>str</i> to a signal number, and stores that result in the location referenced by <i>signum</i>. The name in <i>str</i> can be either the symbol for that signal, without the "SIG" prefix, or a decimal number. All the signal symbols defined in <sys/signal.h> 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 SIGRTMIN to SIGRTMAX, 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 sig2str() function translates the signal number <i>signum</i> to the symbol for that signal, without the "SIG" prefix, and stores that symbol at the location specified by <i>str</i>. The storage referenced by <i>str</i> should be large enough to hold the symbol and a terminating null byte. The symbol SIG2STR_MAX defined by <signal.h> gives the maximum size in bytes required.</p>
RETURN VALUES	<p>The str2sig() function returns 0 if it recognizes the signal name specified in <i>str</i>; otherwise, it returns -1.</p> <p>The sig2str() function returns 0 if the value <i>signum</i> corresponds to a valid signal number; otherwise, it returns -1.</p>
EXAMPLES	<pre>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	kill(1), strsignal(3C)

NAME	strccpy, streadd, strcadd, strccpy – copy strings, compressing or expanding escape codes
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lgen [<i>library</i> ...] #include <libgen.h> char *strccpy(char *output, const char *input); char *strcadd(char *output, const char *input); char *strccpy(char *output, const char *input, const char *exceptions); char *streadd(char *output, const char *input, const char *exceptions);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>strccpy() 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, <code>\n</code>, <code>\001</code>) 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. strccpy() returns the <i>output</i> argument.</p> <p>strcadd() is identical to strccpy(), except that it returns the pointer to the null byte that terminates the output.</p> <p>strccpy() 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, <code>\n</code>, <code>\001</code>). 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 <code>\</code> 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. strccpy() returns the <i>output</i> argument.</p> <p>streadd() is identical to strccpy(), except that it returns the pointer to the null byte that terminates the output.</p>
EXAMPLES	<pre>/* expand all but newline and tab */ strccpy(output, input, "\n\t"); /* concatenate and compress several strings */ cp = strcadd(output, input1); cp = strcadd(cp, input2); cp = strcadd(cp, input3);</pre>
SEE ALSO	<code>string(3C)</code> , <code>strfind(3G)</code>
NOTES	When compiling multi-thread applications, the <code>_REENTRANT</code> flag must be defined on the compile line. This flag should only be used in multi-thread applications.

NAME	strcoll – string collation
SYNOPSIS	#include <string.h> int strcoll(const char *s1, const char *s2);
MT-LEVEL	Safe with exceptions
DESCRIPTION	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)). Both strcoll() and strxfrm() 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() is a more appropriate function because the transformation process occurs only once.
FILES	/usr/lib/locale/LC_COLLATE LC_COLLATE database for <i>locale</i>
SEE ALSO	colltbl(1M) , setlocale(3C) , string(3C) , strxfrm(3C) , wsxfrm(3I) , environ(5)
NOTES	strcoll can be used safely in a multi-thread application, 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 <i>errnum</i>);
MT-LEVEL	Safe
DESCRIPTION	strerror() maps the error number in <i>errnum</i> to an error message string, and returns a pointer to that string. strerror() uses the same set of error messages as perror() . The returned string should not be overwritten.
ERRORS	strerror returns NULL if <i>errnum</i> is out-of-range.
SEE ALSO	gettext(3l) , perror(3C) , setlocale(3C)
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, strstrn, strtrns, str – string manipulations
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lgen [<i>library</i> ...] #include <libgen.h> int strfind(const char *as1, const char *as2); char *strstrn(const char *string, const char *tc); char * strtrns(const char *string, const char *old, const char *new, char *result);</pre>
MT-LEVEL	MT-Safe
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 -1.</p> <p>strstrn() 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	<pre>/* find offset to substring "hello" within as1 */ i = strfind(as1, "hello"); /* trim junk from end of string */ s2 = strstrn(s1, ".*#\$%"); *s2 = '\0'; /* transform lower case to upper case */ a1[] = "abcdefghijklmnopqrstuvwxy"; a2[] = "ABCDEFGHIJKLMNopqrstuvwxyz"; s2 = strtrns(s1, a1, a2, s2);</pre>
SEE ALSO	string(3C)
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>
MT-LEVEL	MT-Safe
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. 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>=f 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, negative amounts are enclosed within parentheses. If neither flag is specified, the '+' style is used.</p> <p>! Suppress the currency symbol from the output conversion.</p> <p>- Specify the alignment. If this flag is present all fields are left-justified (padded to the right) rather than right-justified.</p>

Field Width	<i>w</i>	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	<i>#n</i>	<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 sparators 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	<i>.p</i>	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		<p>The conversion characters and their meanings are:</p> <p><i>i</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).</p> <p><i>n</i> The double argument is formatted according to the locale’s national currency format (for example, in the U.S.A.: \$1,234.56).</p> <p><i>%</i> Convert to a %; no argument is converted. The entire conversion specification must be %%.</p>
Locale Information		The LC_MONETARY category of the program’s locale affects the behaviour 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 conformant 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

strfmon() will fail if:

ENOSYS The function is not supported.

E2BIG Conversion stopped due to lack of space in the buffer.

EXAMPLES

Given a locale for the U.S.A. and the values 123.45, -123.45, and 3456.781:

Conversion Specification	Output	Comments
%n	\$123.45 -\$123.45 \$3,456.78	default formatting
%11n	\$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

SEE ALSO | **localeconv(3C)**

NOTES | This interface is expected to be mandatory in a future issue of this document.
Lower-case conversion characters are reserved for future use and upper-case for implementation-dependent use.

NAME	strftime, cftime, asctime – convert date and time to string																																		
SYNOPSIS	<pre>#include <time.h> size_t strftime(const char *s, size_t maxsize, const char *format, const struct tm *timeptr); int cftime(char *s, char *format, const time_t *clock); int asctime(char *s, const char *format, const struct tm *timeptr);</pre>																																		
MT-LEVEL	MT-Safe																																		
DESCRIPTION	<p>strftime(), asctime(), and cftime() 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 strftime(), 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 strftime() the default format is the same as %c; for cftime() and asctime() the default format is the same as %C. cftime() and asctime() 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 strftime() and asctime(), and by the time represented by <i>clock</i> for cftime().</p> <table border="0"> <tr><td>%%</td><td>same as %</td></tr> <tr><td>%a</td><td>locale's abbreviated weekday name</td></tr> <tr><td>%A</td><td>locale's full weekday name</td></tr> <tr><td>%b</td><td>locale's abbreviated month name</td></tr> <tr><td>%B</td><td>locale's full month name</td></tr> <tr><td>%c</td><td>locale's appropriate date and time representation</td></tr> <tr><td>%C</td><td>locale's date and time representation as produced by date(1)</td></tr> <tr><td>%d</td><td>day of month [1,31]; single digits are preceded by 0</td></tr> <tr><td>%D</td><td>date as %m/%d/%y</td></tr> <tr><td>%e</td><td>day of month [1,31]; single digits are preceded by a space</td></tr> <tr><td>%h</td><td>locale's abbreviated month name</td></tr> <tr><td>%H</td><td>hour (24-hour clock) [0,23]; single digits are preceded by 0</td></tr> <tr><td>%I</td><td>hour (12-hour clock) [1,12]; single digits are preceded by 0</td></tr> <tr><td>%j</td><td>day number of year [1,366]; single digits are preceded by 0</td></tr> <tr><td>%k</td><td>hour (24-hour clock) [0,23]; single digits are preceded by a blank</td></tr> <tr><td>%l</td><td>hour (12-hour clock) [1,12]; single digits are preceded by a blank</td></tr> <tr><td>%m</td><td>month number [1,12]; single digits are preceded by a blank</td></tr> </table>	%%	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	%C	locale's date and time representation as produced by date(1)	%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	%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 a blank
%%	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																																		
%C	locale's date and time representation as produced by date(1)																																		
%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																																		
%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 a blank																																		

%M	minute [0,59]; leading zero 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 [0,61]
%t	insert a tab
%T	time as %H:%M:%S
%u	weekday as a decimal number [1,7], with 1 representing Monday
%U	week number of year as a decimal number [0,53], with Sunday as the first day of week 1
%V	week number of the year as a decimal number [01,53], with Monday as the first day of the week. If the week containing 1 January has four or more days in the new year, then it is considered week 1; otherwise, it is week 53 of the previous year, and the next week is week 1.
%w	weekday as a decimal number [0,6], with 0 representing Sunday
%W	week number of year as a decimal number [0,53], 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 [0,99]
%Y	year, including the century (for example 1993)
%Z	time zone name or abbreviation, or no bytes if no timezone 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 behaviour 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
%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

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()**.

Timezone

Local timezone information is used as though **tzset(3C)** were called.

RETURN VALUES

strftime(), **cftime()**, and **ascftime()** 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*, **strftime()** returns **0** and the contents of the array are indeterminate.

EXAMPLES

The following example illustrates the use of **strftime()**. 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.

```
strftime (str, strsize, "%A %b %d %j", tm_ptr)
```

This results in *str* containing "Thursday Aug 28 240".

FILES

/usr/lib/locale/locale/LC_TIME/time locale specific date and time information

SEE ALSO

date(1), **ctime(3C)**, **mktime(3C)**, **setlocale(3C)**, **strptime(3C)**, **tzset(3C)**, **TIMEZONE(4)**, **strftime(4)**, **environ(5)**

NOTES

The range of values for **%S** is [00,61] rather than [00,59] to allow for the occasional leap second and even more occasional double leap second.

In the XPG4 environment, the conversion specification **%C** is as follows:

%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**.

NAME	string, strcasecmp, strncasecmp, strcat, strncat, strchr, strrchr, strcmp, strncmp, strcpy, strncpy, strcspn, strspn, strdup, strlen, strpbrk, strstr, strtok, strtok_r – string operations
SYNOPSIS	<pre>#include <string.h> int strcasecmp(const char *s1, const char *s2); int strncasecmp(const char *s1, const char *s2, int n); char *strcat(char *dst, const char *src); char *strncat(char *dst, const char *src, 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 *dst, const char *src); char *strncpy(char *dst, const char *src, 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>
MT-LEVEL	See the NOTES section of this page.
DESCRIPTION	<p>string, strcasecmp, strncasecmp, strcat, strncat, strchr, strrchr, strcmp, strncmp, strcpy, strncpy, strcspn, strspn, strdup, strlen, strpbrk, strstr, strtok, strtok_r – string operations</p> <p>The arguments <i>s</i>, <i>s1</i>, <i>s2</i>, <i>src</i>, and <i>dst</i> point to strings (arrays of characters terminated by a null character). The functions strcat(), strncat(), strcpy(), strncpy(), strtok(), and strtok_r() all alter their first argument. These functions do not check for overflow of the array pointed to by the first argument.</p> <p>strcasecmp() and strncasecmp() are case-insensitive versions of strcmp() and strncmp() respectively, described below. strcasecmp() and strncasecmp() assume the ASCII character set and ignore differences in case when comparing lower and upper case characters.</p> <p>strcat() appends a copy of string <i>src</i>, including the terminating null character, to the end of string <i>dst</i>. strncat() appends at most <i>n</i> characters. Each returns a pointer to the null-terminated result. The initial character of <i>src</i> overrides the null character at the end of <i>dst</i>.</p>

strchr() returns a pointer to the first occurrence of *c* (converted to a **char**) in string *s*, or a null pointer if *c* does not occur in the string. **strrchr()** returns a pointer to the last occurrence of *c*. The null character terminating a string is considered to be part of the string.

strcmp() 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 *s1* is greater than, equal to, or less than the string pointed to by *s2* 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. **strncmp()** makes the same comparison but looks at a maximum of *n* bytes. Bytes following a null byte are not compared.

strcpy() copies string *src* to *dst* including the terminating null character, stopping after the null character has been copied. **strncpy()** copies exactly *n* bytes, truncating *src* or adding null characters to *dst* if necessary. The result will not be null-terminated if the length of *src* is *n* or more. Each function returns *dst*.

strcspn() returns the length of the initial segment of string *s1* that consists entirely of characters not from string *s2*. **strspn()** returns the length of the initial segment of string *s1* that consists entirely of characters from string *s2*.

strdup() returns a pointer to a new string that is a duplicate of the string pointed to by *s1*. The space for the new string is obtained using **malloc(3C)**. If the new string cannot be created, a null pointer is returned.

strlen() returns the number of bytes in *s*, not including the terminating null character.

strpbrk() returns a pointer to the first occurrence in string *s1* of any character from string *s2*, or a null pointer if no character from *s2* exists in *s1*.

strstr() locates the first occurrence of the string *s2* (excluding the terminating null character) in string *s1*. **strstr()** 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() 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*. **strtok()** 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() 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.

SEE ALSO `malloc(3C)`, `setlocale(3C)`, `strxfrm(3C)`

NOTES

The `strtok_r()` 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.

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.

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.

`strtok()` is unsafe in multi-thread applications. `strtok_r()` should be used instead.

`string()`, `strcasecmp()`, `strcat()`, `strchr()`, `strcmp()`, `strcpy()`, `strcspn()`, `strdup()`, `strlen()`, `strncasecmp()`, `strncat()`, `strncmp()`, `strncpy()`, `strpbrk()`, `strrchr()`, `strspn()`, and `strstr()`, are MT-Safe in multi-thread applications.

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 *pnread, 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 *pnread);</pre>								
MT-LEVEL	MT-Safe								
DESCRIPTION	<p>The char_to_decimal functions parse a numeric token from at most <i>nmax</i> characters in a string <i>**pc</i> or file <i>*pf</i> or function (<i>*pget</i>()) into a decimal record <i>*pd</i>, classifying the form of the string in <i>*pform</i> and <i>*pechar</i>. The accepted syntax is intended to be sufficiently flexible to accommodate many languages:</p> <p style="padding-left: 40px;"><i>whitespace value</i></p> <p>or</p> <p style="padding-left: 40px;"><i>whitespace sign value</i></p> <p>where <i>whitespace</i> is any number of characters defined by <i>isspace</i> in <ctype.h>, <i>sign</i> is either of [+–], and <i>value</i> can be <i>number</i>, <i>nan</i>, or <i>inf</i>. <i>inf</i> can be INF (<i>inf_form</i>) or INFINITY (<i>infinity_form</i>) without regard to case. <i>nan</i> can be NAN (<i>nan_form</i>) or NAN(<i>nstring</i>) (<i>nanstring_form</i>) without regard to case; <i>nstring</i> is any string of characters not containing ']' or NULL; <i>nstring</i> is copied to <i>pd</i>→ds and, currently, not used subsequently. <i>number</i> consists of</p> <p style="padding-left: 40px;"><i>significand</i></p> <p>or</p> <p style="padding-left: 40px;"><i>significand efield</i></p> <p>where <i>significand</i> must contain one or more digits and may contain one point; possible forms are</p> <table border="0" style="margin-left: 40px;"> <tr> <td><i>digits</i></td> <td>(<i>int_form</i>)</td> </tr> <tr> <td><i>digits.</i></td> <td>(<i>intdot_form</i>)</td> </tr> <tr> <td><i>.digits</i></td> <td>(<i>dotfrac_form</i>)</td> </tr> <tr> <td><i>digits.digits</i></td> <td>(<i>intdotfrac_form</i>)</td> </tr> </table> <p><i>efield</i> consists of</p> <p style="padding-left: 40px;"><i>echar digits</i></p>	<i>digits</i>	(<i>int_form</i>)	<i>digits.</i>	(<i>intdot_form</i>)	<i>.digits</i>	(<i>dotfrac_form</i>)	<i>digits.digits</i>	(<i>intdotfrac_form</i>)
<i>digits</i>	(<i>int_form</i>)								
<i>digits.</i>	(<i>intdot_form</i>)								
<i>.digits</i>	(<i>dotfrac_form</i>)								
<i>digits.digits</i>	(<i>intdotfrac_form</i>)								

or

echar sign digits

where *echar* is one of [Ee], and *digits* contains one or more digits.

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 **ungetc** 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)
{ ... }
int (*pget)(void) = xget;
int xunget(int c)
{ ... }
int (*punget)(int) = xunget;
```

If no valid number was detected, *pd->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()**.

SEE ALSO

ctype(3C), **localeconv(3C)**, **scanf(3S)**, **setlocale(3C)**, **strtod(3C)**, **ungetc(3S)**

NAME	strptime – date and time conversion																																																		
SYNOPSIS	#include <time.h> char *strptime(const char *buf, const char *format, struct tm *tm);																																																		
MT-LEVEL	MT-Safe																																																		
DESCRIPTION	<p>The strptime() function converts the character string pointed to by <i>buf</i> to values which are stored in the tm structure pointed to by <i>tm</i>, using the format specified by <i>format</i>.</p> <p><i>format</i> 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 isspace(3C)) may precede or follow a conversion specification. There must be white-space or other non-alphanumeric characters between any two conversion specifications.</p> <p>The following conversion specifications are supported:</p> <table border="0"> <tr><td>%%</td><td>same as %</td></tr> <tr><td>%a</td><td>day of week, using the locale's weekday names; either the abbreviated or full name may be specified</td></tr> <tr><td>%A</td><td>same as %a</td></tr> <tr><td>%b</td><td>month, using the locale's month names; either the abbreviated or full name may be specified</td></tr> <tr><td>%B</td><td>same as %b</td></tr> <tr><td>%c</td><td>locale's appropriate date and time representation</td></tr> <tr><td>%C</td><td>century number [0,99]; leading zero is permitted but not required</td></tr> <tr><td>%d</td><td>day of month [1,31]; leading zero is permitted but not required</td></tr> <tr><td>%D</td><td>date as %m/%d/%y</td></tr> <tr><td>%e</td><td>same as %d</td></tr> <tr><td>%h</td><td>same as %b</td></tr> <tr><td>%H</td><td>hour (24-hour clock) [0,23]; leading zero is permitted but not required</td></tr> <tr><td>%I</td><td>hour (12-hour clock) [1,12]; leading zero is permitted but not required</td></tr> <tr><td>%j</td><td>day number of the year [1,366]; leading zeros are permitted but not required</td></tr> <tr><td>%m</td><td>month number [1,12]; leading zero is permitted but not required</td></tr> <tr><td>%M</td><td>minute [0-59]; leading zero is permitted but not required</td></tr> <tr><td>%n</td><td>any white space</td></tr> <tr><td>%p</td><td>locale's equivalent of either a.m. or p.m.</td></tr> <tr><td>%r</td><td>appropriate time representation in the 12-hour clock format with %p</td></tr> <tr><td>%R</td><td>time as %H:%M</td></tr> <tr><td>%S</td><td>seconds [0,61]; leading zero is permitted but not required</td></tr> <tr><td>%t</td><td>any white space</td></tr> <tr><td>%T</td><td>time as %H:%M:%S</td></tr> <tr><td>%U</td><td>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</td></tr> <tr><td>%w</td><td>weekday as a decimal number [0,6], with 0 representing Sunday;</td></tr> </table>	%%	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	%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	%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 zeros are permitted but not required	%w	weekday as a decimal number [0,6], with 0 representing Sunday;
%%	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																																																		
%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																																																		
%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 zeros are 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 the century [0,99]; leading zero is permitted but not required
%Y	year, including the century (for example, 1993)
%Z	timezone name or no characters if no time zone information exists

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 behaviour 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.

FILES

/usr/lib/locale/locale/LC_TIME/time
locale specific date and time information
/usr/lib/locale/locale/LC_CTYPE/ctype
character characterization information

SEE ALSO

isspace(3C), **getdate(3C)**, **setlocale(3C)**, **strftime(3C)**

NOTES

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 **strftime()** and **strptime()**. The range of values for **%S** is [00,61] rather than [00,59] to allow for the occasional leap second and even more occasional double leap second.

NAME	strsignal – get error message string
SYNOPSIS	#include <string.h> char *strsignal(int sig);
MT-LEVEL	Safe
DESCRIPTION	strsignal() maps the signal number in <i>sig</i> to a string describing the signal, and returns a pointer to that string. strsignal() uses the same set of the messages as psignal(3C) . The returned string should not be overwritten.
RETURN VALUES	strsignal() returns NULL if <i>sig</i> is not a valid signal number.
SEE ALSO	gettext(3l) , psignal(3C) , setlocale(3C) , str2sig(3C)
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	strtod, atof – convert string to double-precision number
SYNOPSIS	<pre>#include <stdlib.h> double strtod(const char *nptr, char **endptr); double atof(const char *nptr);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>strtod() returns as a double-precision floating-point number the value represented by the character string pointed to by <i>nptr</i>. The string is scanned up to the first unrecognized character.</p> <p>strtod() recognizes an optional string of “white-space” characters (as defined by isspace() in ctype(3C)), then an optional sign, then a string of digits optionally containing a decimal point character, then an optional exponent part including e or E followed by an optional sign, followed by an integer. The decimal point character is defined by the program’s locale (category LC_NUMERIC). In the “C” locale, or in a locale where the decimal point character is not defined, the decimal point character defaults to a period (.).</p> <p>If the value of <i>endptr</i> is not (char **)NULL, a pointer to the character terminating the scan is returned in the location pointed to by <i>endptr</i>. If no number can be formed, <i>endptr</i> is set to <i>nptr</i>, and zero is returned.</p> <p>atof(nptr) is equivalent to: strtod(nptr, (char **)NULL).</p> <p>LC_NUMERIC Determines how strtod and atof handle numeric formats. In the “C” locale, numeric handling follows the U.S. rules.</p>
RETURN VALUES	<p>If the correct value would cause overflow, ±HUGE is returned (according to the sign of the value), and errno is set to ERANGE.</p> <p>If the correct value would cause underflow, 0 is returned and errno is set to ERANGE.</p> <p>When the -Xc or -Xa compilation options are used, HUGE_VAL is returned instead of HUGE.</p> <p>If <i>nptr</i> is NaN, then atof() returns NaN.</p>
FILES	<pre>/usr/lib/locale/locale/LC_NUMERIC/numeric LC_NUMERIC database for locale</pre>
SEE ALSO	ctype(3C) , matherr(3M) , scanf(3S) , strtol(3C) , math(5)

NAME	strtol, strtoll, strtoul, strtoull, atol, atoll, atoi, lltostr, ulltostr – conversion routines
SYNOPSIS	<pre>#include <stdlib.h> long strtol(const char *str, char **ptr, int base); long long strtoll(const char *str, char **ptr, int base); unsigned long strtoul(const char *str, char **ptr, int base); unsigned long long strtoull(const char *str, char **ptr, int base); long atol(const char *str); long long atoll(const char *str); int atoi(const char *str); char *lltostr(long long value, char *ptr); char *ulltostr(unsigned long long value, char *ptr);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>strtol() returns as a long integer the value represented by the character string pointed to by str. The string is scanned up to the first character inconsistent with <i>base</i>. Leading “white-space” characters (as defined by isspace() in ctype(3C)) are ignored.</p> <p>strtoll() is similar to strtol() except that the value is returned as a long long.</p> <p>If the value of <i>ptr</i> is not (char **)NULL, a pointer to the character terminating the scan is returned in the location pointed to by <i>ptr</i>. If no integer can be formed, that location is set to str, and zero is returned.</p> <p>If <i>base</i> is positive (and not greater than 36), it is used as the base for conversion. After an optional leading sign, leading zeros are ignored, and “0x” or “0X” is ignored if <i>base</i> is 16.</p> <p>If <i>base</i> is zero, the string itself determines the base as follows: After an optional leading sign a leading zero indicates octal conversion, and a leading “0x” or “0X” hexadecimal conversion. Otherwise, decimal conversion is used.</p> <p>Truncation from long to int can, of course, take place upon assignment or by an explicit cast.</p> <p>If the value represented by <i>str</i> would cause overflow, LONG_MAX or LONG_MIN is returned (according to the sign of the value), and errno is set to the value, ERANGE.</p> <p>strtoul() and strtoull() are similar to strtol() except that strtoul() returns the value represented by <i>str</i> as an unsigned long integer and strtoull() returns this value as an unsigned long long. If the value represented by <i>str</i> would cause overflow, ULONG_MAX is returned, and errno is set to the value, ERANGE.</p> <p>Except for behavior on error, atol() is equivalent to: strtol(str, (char **)NULL, 10).</p> <p>Except for behavior on error, atoll() is equivalent to: strtoll(str, (char **)NULL, 10).</p>

Except for behavior on error, **atoi()** is equivalent to: **(int) strtol(str, (char **)NULL, 10)**.
lltostr() returns a pointer to the string represented by the **long long value**.
ulltostr() is similar to **lltostr()** except that *value* is an **unsigned long long**.

RETURN VALUES

If **strtol()** is given a *base* greater than 36, it returns 0 and sets **errno** to **EINVAL**.

SEE ALSO

ctype(3C), **scanf(3S)**, **strtod(3C)**

NOTES

strtol() no longer accepts values greater than **LONG_MAX** as valid input. Use **strtoul()** instead.

NAME	strxfrm – string transformation
SYNOPSIS	#include <string.h> size_t strxfrm(char *dst, const char *src, size_t n);
MT-LEVEL	Safe with exceptions
DESCRIPTION	strxfrm() transforms the string <i>src</i> and places the resulting string into the array <i>dst</i> . If strcmp() is applied to two transformed strings, it will return the same result as strcoll() applied to the same two original strings. The transformation is based on the program's locale for category LC_COLLATE (see setlocale(3C)). No more than <i>n</i> bytes will be placed into the resulting array pointed to by <i>dst</i> , including the terminating null character. If <i>n</i> is 0 and <i>dst</i> is a NULL parameter, strxfrm() returns the number of bytes required for the transformed string. If copying takes place between objects that overlap, the behavior is undefined. strxfrm() returns the length of the transformed string (not including the terminating null character). If the value returned is <i>n</i> or more, the contents of the array <i>dst</i> are indeterminate.
RETURN VALUES	On failure, strxfrm() returns (size_t)-1 .
EXAMPLES	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> . 1 + strxfrm(NULL, s, 0);
FILES	/usr/lib/locale/locale/LC_COLLATE LC_COLLATE database for <i>locale</i>
SEE ALSO	colltbl(1M) , setlocale(3C) , strcoll(3C) , string(3C) , wscoll(3I) , environ(5)
NOTES	strxfrm can be used safely in a multi-thread application, as long as setlocale(3C) is not being called to change the locale.

NAME	swab – swap bytes
SYNOPSIS	#include <stdlib.h> void swab(const char *from, char *to, int nbytes);
MT-LEVEL	MT-Safe
DESCRIPTION	swab() copies <i>nbytes</i> bytes pointed to by <i>from</i> to the array pointed to by <i>to</i> , exchanging adjacent even and odd bytes. <i>nbytes</i> should be even and non-negative. If <i>nbytes</i> is odd and positive, swab() uses <i>nbytes</i> –1 instead. If <i>nbytes</i> is negative, swab() does nothing.

NAME	syscall – indirect system call
SYNOPSIS	<pre>/usr/ucb/cc [<i>flag</i> ...] <i>file</i> ... #include <sys/syscall.h> int syscall(<i>number</i>, <i>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 errno (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);																																												
MT-LEVEL	Safe																																												
DESCRIPTION	<p>The sysconf() function provides a method for an application to determine the current value of a configurable system limit or option (variable).</p> <p>The <i>name</i> 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 <i>name</i>.</p> <table border="0" style="width: 100%;"> <tr> <td style="width: 30%;">_SC_ARG_MAX</td> <td style="width: 30%;">ARG_MAX</td> <td style="width: 40%;">Max combined size of argv[] and envp[]</td> </tr> <tr> <td>_SC_CHILD_MAX</td> <td>CHILD_MAX</td> <td>Max processes allowed to any UID</td> </tr> <tr> <td>_SC_CLK_TCK</td> <td>CLK_TCK</td> <td>Ticks per second (clock_t)</td> </tr> <tr> <td>_SC_NGROUPS_MAX</td> <td>NGROUPS_MAX</td> <td>Max simultaneous groups one process may belong to</td> </tr> <tr> <td>_SC_OPEN_MAX</td> <td>OPEN_MAX</td> <td>Max open files per process</td> </tr> <tr> <td>_SC_PASS_MAX</td> <td>PASS_MAX</td> <td></td> </tr> <tr> <td>_SC_PAGESIZE</td> <td>PAGESIZE</td> <td>System memory page size</td> </tr> <tr> <td>_SC_JOB_CONTROL</td> <td>_POSIX_JOB_CONTROL</td> <td>Job control supported (boolean)</td> </tr> <tr> <td>_SC_SAVED_IDS</td> <td>_POSIX_SAVED_IDS</td> <td>Saved ids (seteuid(2)) supported (boolean)</td> </tr> <tr> <td>_SC_VERSION</td> <td>_POSIX_VERSION</td> <td>Version of the POSIX.1 standard supported</td> </tr> <tr> <td>_SC_XOPEN_VERSION</td> <td>_XOPEN_VERSION</td> <td></td> </tr> <tr> <td>_SC_LOGNAME_MAX</td> <td>LOGNAME_MAX</td> <td></td> </tr> <tr> <td>_SC_NPROCESSORS_CONF</td> <td></td> <td>Number of processors (CPUs) configured</td> </tr> <tr> <td>_SC_NPROCESSORS_ONLN</td> <td></td> <td>Number of processors online</td> </tr> </table>			_SC_ARG_MAX	ARG_MAX	Max combined size of argv[] and envp[]	_SC_CHILD_MAX	CHILD_MAX	Max processes allowed to any UID	_SC_CLK_TCK	CLK_TCK	Ticks per second (clock_t)	_SC_NGROUPS_MAX	NGROUPS_MAX	Max simultaneous groups one process may belong to	_SC_OPEN_MAX	OPEN_MAX	Max open files per process	_SC_PASS_MAX	PASS_MAX		_SC_PAGESIZE	PAGESIZE	System memory page size	_SC_JOB_CONTROL	_POSIX_JOB_CONTROL	Job control supported (boolean)	_SC_SAVED_IDS	_POSIX_SAVED_IDS	Saved ids (seteuid(2)) supported (boolean)	_SC_VERSION	_POSIX_VERSION	Version of the POSIX.1 standard supported	_SC_XOPEN_VERSION	_XOPEN_VERSION		_SC_LOGNAME_MAX	LOGNAME_MAX		_SC_NPROCESSORS_CONF		Number of processors (CPUs) configured	_SC_NPROCESSORS_ONLN		Number of processors online
_SC_ARG_MAX	ARG_MAX	Max combined size of argv[] and envp[]																																											
_SC_CHILD_MAX	CHILD_MAX	Max processes allowed to any UID																																											
_SC_CLK_TCK	CLK_TCK	Ticks per second (clock_t)																																											
_SC_NGROUPS_MAX	NGROUPS_MAX	Max simultaneous groups one process may belong to																																											
_SC_OPEN_MAX	OPEN_MAX	Max open files per process																																											
_SC_PASS_MAX	PASS_MAX																																												
_SC_PAGESIZE	PAGESIZE	System memory page size																																											
_SC_JOB_CONTROL	_POSIX_JOB_CONTROL	Job control supported (boolean)																																											
_SC_SAVED_IDS	_POSIX_SAVED_IDS	Saved ids (seteuid(2)) supported (boolean)																																											
_SC_VERSION	_POSIX_VERSION	Version of the POSIX.1 standard supported																																											
_SC_XOPEN_VERSION	_XOPEN_VERSION																																												
_SC_LOGNAME_MAX	LOGNAME_MAX																																												
_SC_NPROCESSORS_CONF		Number of processors (CPUs) configured																																											
_SC_NPROCESSORS_ONLN		Number of processors online																																											

<u>_SC_PHYS_PAGES</u>		The total number of pages of physical memory in the system.
<u>_SC_AVPHYS_PAGES</u>		The number of pages of physical memory not currently in use by the system.
<u>_SC_AIO_LISTIO_MAX</u>	AIO_LISTIO_MAX	Maximum number of I/O operations in a single list I/O call supported by the implementation.
<u>_SC_AIO_MAX</u>	AIO_MAX	Maximum number of outstanding asynchronous I/O operations supported by the implementation.
<u>_SC_AIO_PRIO_DELTA_MAX</u>	AIO_PRIO_DELTA_MAX	The maximum amount by which a process can decrease its asynchronous I/O priority level from its own scheduling priority.
<u>_SC_DELAYTIMER_MAX</u>	DELAYTIMER_MAX	Maximum number of timer expiration overruns.
<u>_SC_MQ_OPEN_MAX</u>	MQ_OPEN_MAX	The maximum number of open message queue descriptors a process may hold.
<u>_SC_MQ_PRIO_MAX</u>	MQ_PRIO_MAX	The maximum number of message priorities supported by the implementation.

_SC_RTSIG_MAX	RTSIG_MAX	Maximum number of realtime signals reserved for application use in this implementation.
_SC_SEM_NSEMS_MAX	SEM_NSEMS_MAX	Maximum number of semaphores that a process may have.
_SC_SEM_VALUE_MAX	SEM_VALUE_MAX	The maximum value a semaphore may have.
_SC_SIGQUEUE_MAX	SIGQUEUE_MAX	Maximum number of queued signals that a process may send and have pending at the receiver(s) at any time.
_SC_TIMER_MAX	TIMER_MAX	Maximum number of timers per process supported by the implementation.
_SC_ASYNCHRONOUS_IO	_POSIX_ASYNCHRONOUS_IO	Supports Asynchronous Input and Output.
_SC_FSYNC	_POSIX_FSYNC	Supports File Synchronization.
_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.
_SC_PRIORITIZED_IO	_POSIX_PRIORITIZED_IO	Supports Prioritized Input and Output.
_SC_PRIORITY_SCHEDULING	_POSIX_PRIORITY_SCHEDULING	Supports Process Scheduling.

<code>_SC_REALTIME_SIGNALS</code>	<code>_POSIX_REALTIME_SIGNALS</code>	Supports Realtime Signals Extension.
<code>_SC_SEMAPHORES</code>	<code>_POSIX_SEMAPHORES</code>	Supports Semaphores.
<code>_SC_SHARED_MEMORY_OBJECTS</code>	<code>_POSIX_SHARED_MEMORY_OBJECTS</code>	Supports Shared Memory Objects.
<code>_SC_SYNCHRONIZED_IO</code>	<code>_POSIX_SYNCHRONIZED_IO</code>	Supports Synchronized Input and Output.
<code>_SC_TIMERS</code>	<code>_POSIX_TIMERS</code>	Supports Timers.

RETURN VALUES

If *name* is an invalid value, `sysconf()` will return `-1` and set `errno` to indicate the error. If `sysconf()` fails due to a value of *name* that is not defined on the system, the function will return a value of `-1` without changing the value of `errno`.

SEE ALSO

`fpathconf(2)`, `seteuid(2)`, `setrlimit(2)`

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.

`_SC_PHYS_PAGES` and `_SC_AVPHYS_PAGES` are specific to Solaris 2.3 and later releases.

The value of `CLK_TCK` may be variable and it should not be assumed that `CLK_TCK` is a compile-time constant.

NAME	syslog, openlog, closelog, setlogmask – control system log																		
SYNOPSIS	<pre>#include <syslog.h> void openlog(char *ident, int logopt, int facility); void syslog(int priority, char *logstring, /* parameters */ ...); void closelog(void); int setlogmask(int maskpri);</pre>																		
MT-LEVEL	Safe																		
DESCRIPTION	<p>syslog() passes a message to syslogd(1M), which may append it to a log file, write it to the system console, or forward it (to either a list of users or syslogd on another host on the network), depending on the configuration of /etc/syslog.conf. <i>logstring</i> is tagged with a priority of <i>priority</i>, and looks like a printf(3B) string with one additional allowable format specification, %m, which is replaced with the error message string corresponding to the error number in errno. A trailing NEWLINE is added if needed. Options passed to openlog() may cause the size of the message to expand. The maximum size of the message passed to syslogd is 1024 bytes.</p> <p>Priorities are encoded as a <i>facility</i> and a <i>level</i>. The facility describes the part of the system generating the message. The level is selected from the bitwise inclusive OR of zero or more of the following flags, defined in the header <syslog.h>.</p> <table border="0" style="margin-left: 2em;"> <tr> <td style="padding-right: 1em;">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> <tr> <td>LOG_DEBUG</td> <td>Messages that contain information normally of use only when debugging a program.</td> </tr> </table> <p>If special processing is needed, openlog() can be called to initialize the log file. The parameter <i>ident</i> is a string that is prepended to every message. <i>logopt</i> is a bit field indicating logging options. Values for <i>logopt</i> are:</p> <table border="0" style="margin-left: 2em;"> <tr> <td style="padding-right: 1em;">LOG_PID</td> <td>Log the process ID with each message. This is useful for identifying specific daemon processes (for daemons that fork).</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_DEBUG	Messages that contain information normally of use only when debugging a program.	LOG_PID	Log the process ID with each message. This is useful for identifying specific daemon processes (for daemons that fork).
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.																		
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 syslogd(1M) . This option is safe to use in daemon processes that have no controlling terminal, since syslog() forks before opening the console.
LOG_NDELAY	Open the connection to syslogd(1M) 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_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 SIGCHLD , since syslog() may otherwise block waiting for a child whose exit status has already been collected.

The *facility* parameter encodes a default facility to be assigned to all messages that do not have an explicit facility already encoded:

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 in.ftpd(1M) .
LOG_AUTH	The authorization system: login(1) , su(1M) , getty(1M) , etc.
LOG_LPR	The line printer spooling system: lpr(1B) , lpc(1B) , etc.
LOG_NEWS	Reserved for the USENET network news system.
LOG_UUCP	Reserved for the UUCP system; it does not currently use syslog .
LOG_CRON	The cron/at facility; crontab(1) , at(1) , cron(1M) , etc.
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.

closelog() can be used to close the log file.

setlogmask() sets the log priority mask to *maskpri* and returns the previous mask. Calls 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_UPTO(toppri)**. The default allows all priorities to be logged.

EXAMPLES

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");
```

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)**

NAME	system, asystem – return physical memory information
SYNOPSIS	long system(void); long asystem(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: system(AS_LIB), asystem(AS_LIB), ... They are therefore removed."</p> <p>The routine system() 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 system() 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. sysconf(SC_PHYS_PAGES) should be used to avoid this limitation. (See sysconf(3C).)</p> <p>The routine asystem() 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 asystem() returns a long integer it is limited similar to system(). sysconf(SC_AVPHYS_PAGES) 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	system() and asystem() are obsolete and should be replaced with sysconf(3C) .

NAME	system – issue a shell command
SYNOPSIS	#include <stdlib.h> int system(const char *string);
MT-LEVEL	MT-Safe
DESCRIPTION	system() causes the <i>string</i> to be given to the shell (see sh(1)) as input, as if the string 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 /bin/sh exists and is executable. If /bin/sh is available, system() returns non-zero; otherwise it returns zero.
RETURN VALUES	system() forks to create a child process that in turn execs /bin/sh in order to execute <i>string</i> . If the fork or exec fails, system() returns a value of -1 and sets errno .
ERRORS	system() fails if one or more of the following are true: EAGAIN The system-imposed limit on the total number of processes under execution by a single user would be exceeded. EINTR system() was interrupted by a signal. ENOMEM The new process requires more memory than is available.
SEE ALSO	sh(1) , exec(2) , fork(2) , waitpid(2)

NAME	t_accept – accept a connect request
SYNOPSIS	<pre>cc [flag ...] file ... -lnsl [library ...] #include <tiuser.h> int t_accept(int <i>fildev</i>, int <i>resfd</i>, struct t_call *<i>call</i>);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>This function is issued by a transport user to accept a connect request. <i>fildev</i> identifies the local transport endpoint where the connect 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. <i>call</i> points to a t_call structure that contains the following members:</p> <pre> struct netbuf addr; struct netbuf opt; struct netbuf udata; int sequence; /* struct netbuf { unsigned int maxlen; unsigned int len; char *buf; }; */</pre> <p>In <i>call</i>, addr is the address of the caller, opt indicates any protocol-specific parameters associated with the connection, udata points to any user data to be returned to the caller, and sequence is the value returned by t_listen that uniquely associates the response with a previously received connect indication.</p> <p>A transport user may accept a connection on either the same, or on a different, local transport endpoint from the one on which the connect indication arrived. If the same endpoint is specified (that is, <i>resfd=fildev</i>), the connection can be accepted unless the following condition is true: The user has received other indications on that endpoint but has not responded to them (with t_accept() or t_snddis(3N)). For this condition, t_accept(3N) will fail and set t_errno to TBADF.</p> <p>If a different transport endpoint is specified (<i>resfd!=fildev</i>), the endpoint must be bound to a protocol address and must be in the T_IDLE state (see t_getstate(3N)) before the t_accept(3N) is issued.</p> <p>For both types of endpoints, t_accept() will fail and set t_errno to TLOOK if there are indications (for example, a connect or disconnect) waiting to be received on that endpoint.</p>

The values of parameters specified by **opt** and the syntax of those values are protocol specific. 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.

RETURN VALUES

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned, **t_errno** is set to indicate the error, and possibly **errno** is set.

ERRORS

On failure, **t_errno** will be set to one of the following:

TACCES	The user does not have permission to accept a connection on the responding transport endpoint or to use the specified options.
TBADDF	The specified file descriptor does not refer to a transport endpoint, or the user is illegally accepting a connection on the same transport endpoint on which the connect indication arrived.
TBADDATA	The amount of user data specified was not within the bounds allowed by the transport provider.
TBADOPT	The specified options were in an incorrect format or contained illegal information.
TBADSEQ	An invalid sequence number was specified.
TLOOK	An asynchronous event has occurred on the transport endpoint referenced by <i>fildev</i> and requires immediate attention.
TNOTSUPPORT	This function is not supported by the underlying transport provider.
TOUTSTATE	The function was issued in the wrong sequence on the transport endpoint referenced by <i>fildev</i> , or the transport endpoint referred to by <i>resfd</i> is not in the T_IDLE state.
TSYSERR	A system error has occurred during execution of this function, errno will be set to the specific error.

SEE ALSO

t_accept(3N), **t_connect(3N)**, **t_getinfo(3N)**, **t_getstate(3N)**, **t_listen(3N)**, **t_open(3N)**, **t_snddis(3N)**, **t_rcvconnect(3N)**

Network Interfaces Programmer's Guide

NOTES

This interface is safe in multithreaded applications.

NAME	t_alloc – allocate a library structure														
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lnsl [<i>library</i> ...] #include <tiuser.h> char *t_alloc(int <i>fildev</i>, int <i>struct_type</i>, int <i>fields</i>);</pre>														
MT-LEVEL	MT-Safe														
DESCRIPTION	<p>The t_alloc() 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. The structure to allocate is specified by <i>struct_type</i>, and can be one of the following:</p> <table border="0"> <tr> <td>T_BIND</td> <td>struct t_bind</td> </tr> <tr> <td>T_CALL</td> <td>struct t_call</td> </tr> <tr> <td>T_OPTMGMT</td> <td>struct t_optmgmt</td> </tr> <tr> <td>T_DIS</td> <td>struct t_discon</td> </tr> <tr> <td>T_UNITDATA</td> <td>struct t_unitdata</td> </tr> <tr> <td>T_UDERROR</td> <td>struct t_uderr</td> </tr> <tr> <td>T_INFO</td> <td>struct t_info</td> </tr> </table> <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 T_INFO, contains at least one field of type struct netbuf. netbuf is described in t_connect(3N). For each field of this type, the user may specify that the buffer for that field should be allocated as well. The <i>fields</i> argument specifies this option, where the argument is the bitwise-OR of any of the following:</p> <p>T_ADDR The addr field of the t_bind, t_call, t_unitdata, or t_uderr structures.</p> <p>T_OPT The opt field of the t_optmgmt, t_call, t_unitdata, or t_uderr structures.</p> <p>T_UDATA The udata field of the t_call, t_discon, or t_unitdata structures.</p> <p>T_ALL All relevant fields of the given structure.</p> <p>For each field specified in <i>fields</i>, t_alloc() will allocate memory for the buffer associated with the field, and initialize the buf pointer and maxlen (see netbuf in t_connect(3N) for description of buf and maxlen) field accordingly. The length of the buffer allocated will be based on the same size information that is returned to the user on t_open(3N) and t_getinfo(3N). Thus, <i>fildev</i> must refer to the transport endpoint through which the newly allocated structure will be passed, so that the appropriate size information can be accessed. If the size value associated with any specified field is -1, the underlying service provider can support a buffer of unlimited size. If this is the case, t_alloc() will allocate a buffer with the default size 1024 bytes. See the NOTES section for information regarding memory allocation for buffers other than 1024 bytes. If the size value is -2, t_alloc() will set the buffer pointer to NULL and the buffer maximum size to 0, and then will return</p>	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
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														

success (see **t_open(3N)** or **t_getinfo(3N)**).

For any field not specified in *fields*, **buf** will be set to NULL and **maxlen** will be set to zero.

Use of **t_alloc()** to allocate structures will help ensure the compatibility of user programs with future releases of the transport interface.

RETURN VALUES

On successful completion, **t_alloc()** returns a pointer to the newly allocated structure.

On failure, NULL is returned, **t_errno** is set to indicate the error, and possibly **errno** is set.

ERRORS

On failure, **t_errno** will be set to one of the following:

TBADF The specified file descriptor does not refer to a transport endpoint.

TSYSERR A system error has occurred during execution of this function, **errno** will be set to the specific error.

SEE ALSO

t_connect(3N), **t_free(3N)**, **t_getinfo(3N)**, **t_open(3N)**

Network Interfaces Programmer's Guide

NOTES

If the underlying service provider supports a buffer of unlimited size in the **netbuf** structure (see **t_connect(3N)**), **t_alloc()** 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 **malloc(3C)**. The **buf** and **maxlen** fields of the **netbuf** data structure can then be updated with the address of the new buffer and the 1024 byte buffer originally allocated by **t_alloc()** can be freed using **free(3C)**.

This interface is safe in multithreaded applications.

NAME	t_bind – bind an address to a transport endpoint
SYNOPSIS	<pre>cc [flag ...] file ... -lnsl [library ...] #include <tiuser.h> int t_bind(int fildes, const struct t_bind *req, struct t_bind *ret);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>This function associates a protocol address with the transport endpoint specified by <i>fildes</i> and activates that transport endpoint. In connection mode, the transport provider may begin accepting or requesting connections on the transport endpoint. In connectionless mode, the transport user may send or receive data units through the transport endpoint. The <i>req</i> and <i>ret</i> arguments point to a t_bind structure containing the following members:</p> <pre> struct netbuf addr; unsigned qlen;</pre> <p>netbuf is described in t_connect(3N). The addr field of the t_bind structure specifies a protocol address and the qlen field is used to indicate the maximum number of outstanding connect indications.</p> <p><i>req</i> is used to request that an address, represented by the netbuf structure, be bound to the given transport endpoint. len (see netbuf in t_connect(3N); also for buf and maxlen) specifies the number of bytes in the address and buf points to the address buffer. maxlen has no meaning for the <i>req</i> argument. On return, <i>ret</i> contains the address that the transport provider actually bound to the transport endpoint; this may be different from the address specified by the user in <i>req</i>. In <i>ret</i>, the user specifies maxlen, which is the maximum size of the address buffer, and buf, which points to the buffer where the address is to be placed. On return, len specifies the number of bytes in the bound address and buf points to the bound address. If maxlen is not large enough to hold the returned address, an error will result.</p> <p>If the requested address is not available, or if no address is specified in <i>req</i> (the len field of addr in <i>req</i> is zero) the transport provider may assign an appropriate address to be bound, and will return that address in the addr field of <i>ret</i>. The user can compare the addresses in <i>req</i> and <i>ret</i> to determine whether the transport provider bound the transport endpoint to a different address than that requested.</p> <p><i>req</i> may be NULL 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 must assign an address to the transport endpoint. Similarly, <i>ret</i> may be NULL 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 <i>req</i> and <i>ret</i> to NULL 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.</p>

The **qlen** field has meaning only when initializing a connection-mode service. It specifies the number of outstanding connect indications the transport provider should support for the given transport endpoint. An outstanding connect indication is one that has been passed to the transport user by the transport provider. 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 connect indications. On return, the **qlen** field in *ret* will contain the negotiated value.

This function allows more than one transport endpoint to be bound to the same protocol address (however, the transport provider must support this capability also), but it is not allowable to bind more than one protocol address to the same transport endpoint. If a user binds more than one transport endpoint to the same protocol address, only one endpoint can be used to listen for connect 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 connect indication. If a user attempts to bind a protocol address to a second transport endpoint with a value of **qlen** greater than zero, the transport provider will assign another address to be bound to that endpoint. If 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 that connection. No other transport endpoints may be bound for listening while that initial listening endpoint is in the data transfer phase. This will prevent more than one transport endpoint bound to the same protocol address from accepting connect indications.

RETURN VALUES

t_bind() returns **0** on success. On failure, **t_bind()** returns **-1**, **t_errno** is set to indicate the error, and possibly **errno** is set.

ERRORS

On failure, **t_errno** will be set to one of the following:

TACCES	The user does not have permission to use the specified address.
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 is not sufficient to store the value of that argument. The provider's state will change.
TNOADDR	The transport provider could not allocate an address. T_IDLE and the information to be returned in <i>ret</i> will be discarded.

TOUTSTATE The function was issued in the wrong sequence.
TSYSERR A system error has occurred during execution of this function, **errno** will be set to the specific error.

SEE ALSO **t_connect(3N), t_open(3N), t_optmgmt(3N), t_unbind(3N)**

Network Interfaces Programmer's Guide

NOTES This interface is safe in multithreaded applications.

NAME	t_close – close a transport endpoint
SYNOPSIS	<pre>cc [flag ...] file ... -lnsl [library ...] #include <tiuser.h> int t_close(int fildes);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>The t_close() function informs the transport provider that the user is finished with the transport endpoint specified by <i>fildes</i>, and frees any local library resources associated with the endpoint. In addition, t_close() closes the file associated with the transport endpoint.</p> <p>t_close() should be called from the T_UNBND state (see t_getstate(3N)). 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, close(2) will be issued for that file descriptor; the close will be abortive if no other process has that file open, and will break any transport connection that may be associated with that endpoint.</p>
RETURN VALUES	t_close returns 0 on success. On failure t_close returns -1, t_errno is set to indicate the error, and possibly errno is set.
ERRORS	<p>On failure, t_errno will be set to the following:</p> <p>TBADF The specified file descriptor does not refer to a transport endpoint.</p> <p>TSYSERR A system error occurred during execution of this function, errno will be set to the specific error.</p>
SEE ALSO	close(2) , t_getstate(3N) , t_open(3N) , t_unbind(3N) <i>Network Interfaces Programmer's Guide</i>
NOTES	This interface is safe in multithreaded applications.

NAME	t_connect – establish a connection with another transport user
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lnsl [<i>library</i> ...] #include <tiuser.h> int t_connect(int <i>fildev</i>, const struct t_call *<i>sndcall</i>, struct t_call *<i>rcvcall</i>);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>This function enables a transport user to request a connection to the specified destination transport user. <i>fildev</i> identifies the local transport endpoint where communication will be established, while <i>sndcall</i> and <i>rcvcall</i> point to a t_call structure that contains the following members:</p> <pre> struct netbuf addr; struct netbuf opt; struct netbuf udata; int sequence;</pre> <p><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. The address is specified in the netbuf structure which has the following format:</p> <pre>struct netbuf { unsigned int maxlen; unsigned int len; char *buf; }</pre> <p>where maxlen specifies the maximum length of the buffer in bytes, len specifies the bytes of data in the buffer, and buf points to the buffer that contains the data.</p> <p>In <i>sndcall</i>, addr specifies the protocol address of the destination transport user, opt presents any protocol-specific information that might be needed by the transport provider, udata points to optional user data that may be passed to the destination transport user during connection establishment, and sequence has no meaning for this function.</p> <p>On return in <i>rcvcall</i>, addr returns the protocol address associated with the responding transport endpoint, opt presents any protocol-specific information associated with the connection, udata points to optional user data that may be returned by the destination transport user during connection establishment, and sequence has no meaning for this function.</p> <p>The opt argument implies no structure on the options that may be passed to the transport provider. The transport provider is free to specify the structure of any options passed to it. These options are specific to the underlying protocol of the transport provider. The user may choose not to negotiate protocol options by setting the len field of opt to zero. In this case, the provider may use default options.</p>

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** (see **netbuf** in **t_connect(3N)**) field 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** (see **netbuf** in **t_connect(3N)**) field of each argument must be set before issuing this function to indicate the maximum size of the buffer for each. However, *rcvcall* may be NULL, in which case no information is given to the user on return from **t_connect()**.

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_NDELAY** or **O_NONBLOCK** is set (using **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 connect request to the destination transport user.

RETURN VALUES

t_connect() returns 0 on success. On failure **t_connect()** returns -1, **t_errno** is set to indicate the error, and possibly **errno** is set.

ERRORS

On failure, **t_errno** will be set to one of the following:

TACCES	The user does not have permission to use the specified address or options.
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 is 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 T_DATAXFER , and the connect indication 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	O_NDELAY or O_NONBLOCK 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 function was issued in the wrong sequence.

TSYSERR A system error has occurred during execution of this function, **errno** will be set to the specific error.

SEE ALSO **fcntl(2)**, **t_accept(3N)**, **t_getinfo(3N)**, **t_listen(3N)**, **t_open(3N)**, **t_optmgmt(3N)**, **t_rcvconnect(3N)**

Network Interfaces Programmer's Guide

NOTES This interface is safe in multithreaded applications.

NAME	t_error – produce error message
SYNOPSIS	<pre>cc [flag ...] file ... -lnsl [library ...] #include <tiuser.h> void t_error(const char *errmsg); extern int t_errno; extern char *t_errlist[]; extern int t_nerr;</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>t_error() produces a message on the standard error output which describes the last error encountered during a call to a transport function. The argument string <i>errmsg</i> is a user-supplied error message that gives context to the error.</p> <p>t_error() prints the user-supplied error message followed by a colon and the standard transport function error message for the current value contained in t_errno. If t_errno is TSYSERR, t_error will also print the standard error message for the current value contained in errno (see intro(2)).</p> <p>t_errlist is the array of message strings, to allow user message formatting. t_errno can be used as an index into this array to retrieve the error message string (without a terminating newline). t_nerr is the maximum index value for the t_errlist array.</p> <p>t_errno is set when an error occurs and is not cleared on subsequent successful calls.</p>
EXAMPLES	<p>If a t_connect() function fails on transport endpoint <i>fd2</i> 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 would print as:</p> <pre>t_connect failed on fd2: Incorrect transport address format</pre> <p>where "t_connect failed on fd2" tells the user which function failed on which transport endpoint, and "Incorrect transport address format" identifies the specific error that occurred.</p>
SEE ALSO	<p>intro(2) <i>Network Interfaces Programmer's Guide</i></p>
WARNINGS	<p>This interface is deprecated. Use t_strerror(3N) instead. For details on the ramifications of referencing t_errlist[] directly, please refer to the Copy Relocations section of Chapter 4 in the <i>Linker and Libraries Guide</i>.</p>
NOTES	<p>This interface is safe in multithreaded applications.</p>

NAME	t_free – free a library structure														
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lnsl [<i>library</i> ...] #include <tiuser.h> int t_free(char *ptr, int struct_type);</pre>														
MT-LEVEL	MT-Safe														
DESCRIPTION	<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><i>ptr</i> points to one of the six structure types described for t_alloc(3N), and <i>struct_type</i> identifies the type of that structure, which can be one of the following:</p> <table border="0"> <tr> <td>T_BIND</td> <td>struct t_bind</td> </tr> <tr> <td>T_CALL</td> <td>struct t_call</td> </tr> <tr> <td>T_OPTMGMT</td> <td>struct t_optmgmt</td> </tr> <tr> <td>T_DIS</td> <td>struct t_discon</td> </tr> <tr> <td>T_UNITDATA</td> <td>struct t_unitdata</td> </tr> <tr> <td>T_UDERROR</td> <td>struct t_uderr</td> </tr> <tr> <td>T_INFO</td> <td>struct t_info</td> </tr> </table> <p>where each of these structures is used as an argument to one or more transport functions. t_free() will check the addr, opt, and udata fields of the given structure (as appropriate), and free the buffers pointed to by the buf field of the netbuf (see t_connect(3N)) structure. If buf is NULL, 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 buf pointers points to a block of memory that was not previously allocated by t_alloc(3N).</p>	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
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														
RETURN VALUES	t_free returns 0 on success. On failure t_free returns -1, t_errno is set to indicate the error, and possibly errno is set.														
ERRORS	<p>On failure, t_errno will be set to the following:</p> <table border="0"> <tr> <td>TSYSERR</td> <td>A system error has occurred during execution of this function, errno will be set to the specific error.</td> </tr> </table>	TSYSERR	A system error has occurred during execution of this function, errno will be set to the specific error.												
TSYSERR	A system error has occurred during execution of this function, errno will be set to the specific error.														
SEE ALSO	t_connect(3N) , t_alloc(3N) <i>Network Interfaces Programmer's Guide</i>														
NOTES	This interface is safe in multithreaded applications.														

NAME	t_getinfo – get protocol-specific service information
SYNOPSIS	<pre>cc [flag ...] file ... -lnsl [library ...] #include <tiuser.h> int t_getinfo(int fildes, struct t_info *info);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>This function returns the current characteristics of the underlying transport protocol associated with file descriptor <i>fildes</i>. The <i>info</i> structure is used to return the same information returned by t_open(3N). This function enables a transport user to access this information during any phase of communication.</p> <p>The <i>info</i> argument points to a t_info structure, which contains the following members:</p> <pre> long addr; /* max size of transport protocol address */ long options; /* max number of bytes of protocol-specific options */ long tsdu; /* max size of transport service data unit (TSDU) */ long etsdu; /* max size of expedited transport service data unit (ETSDU) */ long connect; /* max amount of data allowed on connection establishment functions */ long discon; /* max amount of data allowed on t_snddis(3N) and t_rcvdis(3N) functions */ long servtype; /* service type supported by the transport provider */</pre> <p>The values of the fields have the following meanings:</p> <p>addr A value greater than or equal to zero indicates the maximum size of a transport protocol address; a value of -1 specifies that there is no limit on the address size; and a value of -2 specifies that the transport provider does not provide user access to transport protocol addresses.</p> <p>options A value greater than or equal to zero indicates the maximum number of bytes of protocol-specific options supported by the provider; a value of -1 specifies that there is no limit on the option size; and a value of -2 specifies that the transport provider does not support user-settable options.</p> <p>tsdu A value greater than zero specifies the maximum size of a transport service data unit (TSDU); a value of zero specifies that the transport provider does not support the concept of TSDU, although it does support the sending across a connection of a data stream with no logical boundaries preserved; a value of -1 specifies that there is no limit on the size of a TSDU; and a value of -2 specifies that the transfer of normal data is not supported by the transport provider.</p>

etsdu	A value greater than zero specifies the maximum size of an expedited transport service data unit (ETSDU); a value of zero specifies that the transport provider does not support the concept of ETSDU , although it does support the sending across a connection of an expedited data stream with no logical boundaries preserved. A value of -1 specifies that there is no limit on the size of an ETSDU ; and a value of -2 specifies that the transfer of expedited data is not supported by the transport provider.
connect	A value greater than or equal to zero specifies the maximum amount of data that may be associated with connection establishment functions; a value of -1 specifies that there is no limit on the amount of data sent during connection establishment; and a value of -2 specifies that the transport provider does not allow data to be sent with connection establishment functions.
discon	A value greater than or equal to zero specifies the maximum amount of data that may be associated with the t_snddis(3N) and t_rcvdis(3N) functions; a value of -1 specifies that there is no limit on the amount of data sent with these abortive release functions; and a value of -2 specifies that the transport provider does not allow data to be sent with the abortive release functions.
servtype	This field specifies the service type supported by the transport provider, as described below.

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()** 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 option negotiation, and **t_getinfo()** enables a user to retrieve the current characteristics.

The **servtype** field of *info* may specify 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(3N) will return -2 for etsdu , connect , and discon .

RETURN VALUES

t_getinfo() returns 0 on success. On failure **t_getinfo()** returns -1 , **t_errno** is set to indicate the error, and possibly **errno** is set.

- ERRORS** On failure, **t_errno** will be set to one of the following:
- TBADF** The specified file descriptor does not refer to a transport endpoint.
- TSYSERR** A system error has occurred during execution of this function, **errno** will be set to the specific error.
- SEE ALSO** **t_open(3N)**, **t_rcvdis(3N)**, **t_snddis(3N)**
Network Interfaces Programmer's Guide
- NOTES** This interface is safe in multithreaded applications.

NAME	t_getstate – get the current state														
SYNOPSIS	<pre>cc [flag ...] file ... -lnsl [library ...] #include <tiuser.h> int t_getstate(int fildes);</pre>														
MT-LEVEL	MT-Safe														
DESCRIPTION	The t_getstate() function returns the current state of the provider associated with the transport endpoint specified by <i>fildes</i> .														
RETURN VALUES	<p>t_getstate() returns the current state on successful completion. On failure t_getstate() returns -1, t_errno is set to indicate the error, and possibly errno is set. The current state may be 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 orderly release (waiting for an orderly release indication)</td> </tr> <tr> <td>T_INREL</td> <td>incoming orderly release (waiting for an orderly release request)</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 orderly release (waiting for an orderly release indication)	T_INREL	incoming orderly release (waiting for an orderly release request)
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	<p>On failure, t_errno will be 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>TSTATECHNG</td> <td>The transport provider is undergoing a state change.</td> </tr> <tr> <td>TSYSERR</td> <td>A system error has occurred during execution of this function, errno will be set to the specific error.</td> </tr> </table>	TBADF	The specified file descriptor does not refer to a transport endpoint.	TSTATECHNG	The transport provider is undergoing a state change.	TSYSERR	A system error has occurred during execution of this function, errno will be set to the specific error.								
TBADF	The specified file descriptor does not refer to a transport endpoint.														
TSTATECHNG	The transport provider is undergoing a state change.														
TSYSERR	A system error has occurred during execution of this function, errno will be set to the specific error.														
SEE ALSO	<p>t_open(3N) <i>Network Interfaces Programmer's Guide</i></p>														
NOTES	This interface is safe in multithreaded applications.														

NAME	t_listen – listen for a connect request				
SYNOPSIS	<pre>cc [flag ...] file ... -lnsl [library ...] #include <tiuser.h> int t_listen(int fildes, struct t_call *call);</pre>				
MT-LEVEL	MT-Safe				
DESCRIPTION	<p>This function listens for a connect request from a calling transport user. <i>fildes</i> identifies the local transport endpoint where connect indications arrive, and on return, <i>call</i> contains information describing the connect indication. <i>call</i> points to a t_call structure, which contains the following members:</p> <pre> struct netbuf addr; struct netbuf opt; struct netbuf udata; int sequence;</pre> <p>netbuf is described in t_connect(3N). In <i>call</i>, addr returns the protocol address of the calling transport user, opt returns protocol-specific parameters associated with the connect request, udata returns any user data sent by the caller on the connect request, and sequence is a number that uniquely identifies the returned connect indication. The value of sequence enables the user to listen for multiple connect indications before responding to any of them.</p> <p>Since this function returns values for the addr, opt, and udata fields of <i>call</i>, the maxlen (see netbuf in t_connect(3N)) field of each must be set before issuing t_listen() to indicate the maximum size of the buffer for each.</p> <p>By default, t_listen() executes in synchronous mode and waits for a connect indication to arrive before returning to the user. However, if O_NDELAY or O_NONBLOCK is set (using t_open() or fcntl()), t_listen() executes asynchronously, reducing to a poll for existing connect indications. If none are available, it returns -1 and sets t_errno to TNO-DATA.</p>				
RETURN VALUES	t_listen() returns 0 on success. On failure t_listen() returns -1 , t_errno is set to indicate the error, and possibly errno is set.				
ERRORS	<p>On failure, t_errno will be 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>TBUFOVFLW</td> <td>The number of bytes allocated for an incoming argument is not sufficient to store the value of that argument. The provider's state, as seen by the user, changes to T_INCON, and the connect indication information to be returned in <i>call</i> is discarded.</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 is not sufficient to store the value of that argument. The provider's state, as seen by the user, changes to T_INCON , and the connect indication information to be returned in <i>call</i> is discarded.
TBADF	The specified file descriptor does not refer to a transport endpoint.				
TBUFOVFLW	The number of bytes allocated for an incoming argument is not sufficient to store the value of that argument. The provider's state, as seen by the user, changes to T_INCON , and the connect indication information to be returned in <i>call</i> is discarded.				

TLOOK	An asynchronous event has occurred on this transport endpoint and requires immediate attention.
TNODATA	O_NDELAY or O_NONBLOCK was set, but no connect indications had been queued.
TNOTSUPPORT	This function is not supported by the underlying transport provider.
TSYSERR	A system error has occurred during execution of this function, errno will be set to the specific error.

SEE ALSO **t_accept(3N)**, **t_bind(3N)**, **t_connect(3N)**, **t_open(3N)**, **t_rcvconnect(3N)**

Network Interfaces Programmer's Guide

WARNINGS If a user issues **t_listen()** in synchronous mode on a transport endpoint that was not bound for listening (that is, **qlen** was zero on **t_bind()**), the call will wait forever because no connect indications will arrive on that endpoint.

NOTES This interface is safe in multithreaded applications.

NAME	t_look – look at the current event on a transport endpoint														
SYNOPSIS	<pre>cc [flag ...] file ... -lnsl [library ...] #include <tiuser.h> int t_look(int fildes);</pre>														
MT-LEVEL	MT-Safe														
DESCRIPTION	<p>This function returns the current event on the transport endpoint specified by <i>fildes</i>. This function enables a transport provider to notify a transport user of an asynchronous event when the user is issuing 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. Otherwise, t_look() 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>disconnect 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> </table> <p>On failure, -1 is returned, t_errno is set to indicate the error, and possibly errno is set.</p>	T_LISTEN	connection indication received	T_CONNECT	connect confirmation received	T_DATA	normal data received	T_EXDATA	expedited data received	T_DISCONNECT	disconnect received	T_UDERR	datagram error indication	T_ORDREL	orderly release indication
T_LISTEN	connection indication received														
T_CONNECT	connect confirmation received														
T_DATA	normal data received														
T_EXDATA	expedited data received														
T_DISCONNECT	disconnect received														
T_UDERR	datagram error indication														
T_ORDREL	orderly release indication														
ERRORS	<p>On failure, t_errno will be 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>TSYSERR</td> <td>A system error has occurred during execution of this function, errno will be set to the specific error.</td> </tr> </table>	TBADF	The specified file descriptor does not refer to a transport endpoint.	TSYSERR	A system error has occurred during execution of this function, errno will be set to the specific error.										
TBADF	The specified file descriptor does not refer to a transport endpoint.														
TSYSERR	A system error has occurred during execution of this function, errno will be set to the specific error.														
SEE ALSO	<p>t_open(3N) <i>Network Interfaces Programmer's Guide</i></p>														
NOTES	This interface is safe in multithreaded applications.														

NAME	t_open – establish a transport endpoint				
SYNOPSIS	<pre>cc [flag ...] file ... -lnsl [library ...] #include <tiuser.h> #include <fcntl.h> int t_open(const char *path, int oflag, struct t_info *info);</pre>				
MT-LEVEL	MT-Safe				
DESCRIPTION	<p>t_open() must be called as the first step in the initialization of a transport endpoint. This function establishes a transport endpoint by opening a file that identifies a particular transport provider (that is, transport protocol) and returning a file descriptor that identifies that endpoint. For example, opening the file /dev/iso_cots identifies an OSI connection-oriented transport layer protocol as the transport provider.</p> <p>path points to the path name of the file to open, and oflag identifies any open flags (as in open(2)). oflag may be constructed from O_NDELAY or O_NONBLOCK OR-ed with O_RDWR. These flags are defined in the header <fcntl.h>. t_open() returns a file descriptor that 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 t_info structure. See t_getinfo(3N) for a description of the t_info structure.</p> <p>If info is set to NULL by the transport user, no protocol information is returned by t_open().</p>				
RETURN VALUES	t_open() returns a valid file descriptor on success. On failure t_open() returns -1, t_errno is set to indicate the error, and possibly errno is set.				
ERRORS	<p>On failure, t_errno will be set to the following:</p> <table border="0"> <tr> <td style="padding-right: 20px;">TBADFLAG</td> <td>An invalid flag is specified.</td> </tr> <tr> <td>TSYSERR</td> <td>A system error has occurred during execution of this function, errno will be set to the specific error.</td> </tr> </table>	TBADFLAG	An invalid flag is specified.	TSYSERR	A system error has occurred during execution of this function, errno will be set to the specific error.
TBADFLAG	An invalid flag is specified.				
TSYSERR	A system error has occurred during execution of this function, errno will be set to the specific error.				
SEE ALSO	<p>open(2), t_getinfo(3N)</p> <p><i>Network Interfaces Programmer's Guide</i></p>				
NOTES	This interface is safe in multithreaded applications.				

NAME	t_optmgmt – manage options for a transport endpoint
SYNOPSIS	<pre>cc [flag ...] file ... -lnsl [library ...] #include <tiuser.h> int t_optmgmt(int <i>fildev</i>, const struct t_optmgmt *<i>req</i>, struct t_optmgmt *<i>ret</i>);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>The t_optmgmt() function enables a transport user to retrieve, verify, or negotiate protocol options with the transport provider. <i>fildev</i> identifies a bound transport endpoint. The <i>req</i> and <i>ret</i> arguments point to a t_optmgmt structure containing the following members:</p> <pre style="margin-left: 40px;">struct netbuf <i>opt</i>; long <i>flags</i>;</pre> <p>The opt field identifies protocol options and the flags field is used to specify the action to take with those options.</p> <p>The options are represented by a netbuf (see t_connect(3N)); also for len, buf, and maxlen structure in a manner similar to the address in t_bind(3N). <i>req</i> is used to request a specific action of the provider and to send options to the provider. len specifies the number of bytes in the options, buf points to the options buffer, and maxlen has no meaning for the <i>req</i> argument. The transport provider may return options and flag values to the user through <i>ret</i>. For <i>ret</i>, maxlen specifies the maximum size of the options buffer and buf points to the buffer where the options are to be placed. On return, len specifies the number of bytes of options returned. maxlen has no meaning for the <i>req</i> argument, but must be set in the <i>ret</i> argument to specify the maximum number of bytes the options buffer can hold. The actual structure and content of the options is imposed by the transport provider.</p> <p>The flags field of <i>req</i> can specify one of the following actions:</p> <p>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 <i>ret</i>.</p> <p>T_CHECK This action enables the user to verify whether the options specified in <i>req</i> are supported by the transport provider. On return, the flags field of <i>ret</i> 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.</p> <p>T_DEFAULT This action enables a user to retrieve the default options supported by the transport provider into the opt field of <i>ret</i>. In <i>req</i>, the len field of opt must be zero and the buf field may be NULL.</p>

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.

RETURN VALUES

t_optmgmt() returns 0 on success. On failure **t_optmgmt()** returns -1, **t_errno** is set to indicate the error, and possibly **errno** is set.

ERRORS

On failure, **t_errno** will be set to one of the following:

TBADF	The specified file descriptor does not refer to a transport endpoint.
TACCES	The user does not have permission to negotiate the specified options.
TBADFLAG	An invalid flag was specified.
TBADOPT	The specified protocol options were in an incorrect format or contained illegal information.
TBUFOVFLW	The number of bytes allowed for an incoming argument is not sufficient to store the value of that argument. The information to be returned in <i>ret</i> will be discarded.
TOUTSTATE	The function was issued in the wrong sequence.
TSYSERR	A system error has occurred during execution of this function, errno will be set to the specific error.

SEE ALSO

t_bind(3N), **t_connect(3N)**, **t_getinfo(3N)**, **t_open(3N)**

Network Interfaces Programmer's Guide

NOTES

This interface is safe in multithreaded applications.

NAME	t_rcv – receive data or expedited data sent over a connection
SYNOPSIS	cc [<i>flag ...</i>] <i>file ...</i> -lnsl [<i>library ...</i>] int t_rcv(int <i>fildev</i> , char * <i>buf</i> , unsigned <i>nbytes</i> , int * <i>flags</i>);
MT-LEVEL	MT-Safe
DESCRIPTION	<p>This function receives either normal or expedited data. <i>fildev</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. <i>flags</i> may be set on return from t_rcv() and specifies optional flags as described below.</p> <p>By default, t_rcv() operates in synchronous mode and will wait for data to arrive if none is currently available. However, if O_NDELAY or O_NONBLOCK is set (using t_open(3N) orfcntl(2)), t_rcv() will execute in asynchronous mode and will fail if no data is available. (See TNODATA below.)</p> <p>On return from the call, if T_MORE 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_rcv() calls. Each t_rcv() with the T_MORE flag set indicates that another t_rcv() must follow to get more data for the current TSDU. The end of the TSDU is identified by the return of a t_rcv() call with the T_MORE 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 t_open(3N) or t_getinfo(3N), the T_MORE flag is not meaningful and should be ignored.</p> <p>On return, the data returned is expedited data if T_EXPEDITED is set in <i>flags</i>. If the number of bytes of expedited data exceeds <i>nbytes</i>, t_rcv() will set T_EXPEDITED and T_MORE on return from the initial call. Subsequent calls to retrieve the remaining ETSDU will have T_EXPEDITED set on return. The end of the ETSDU is identified by the return of a t_rcv() call with the T_MORE flag not set.</p> <p>If expedited data arrives after part of a ETSDU has been retrieved, receipt of the remainder of the TSDU will be suspended until the ETSDU has been processed. Only after the full ETSDU has been retrieved (T_MORE not set) will the remainder of the TSDU be available to the user.</p>
RETURN VALUES	On successful completion, t_rcv() returns the number of bytes received. On failure t_rcv() returns -1, t_errno is set to indicate the error, and possibly errno is set.
ERRORS	<p>On failure, t_errno will be set to one of the following:</p> <p>TBADF The specified file descriptor does not refer to a transport endpoint.</p> <p>TLOOK An asynchronous event has occurred on this transport endpoint and requires immediate attention.</p> <p>TNODATA O_NDELAY or O_NONBLOCK was set, but no data is currently available from the transport provider.</p>

TNOTSUPPORT This function is not supported by the underlying transport provider.

TSYSERR A system error has occurred during execution of this function, **errno** will be set to the specific error.

SEE ALSO **fcntl(2), t_getinfo(3N), t_open(3N), t_snd(3N)**

Network Interfaces Programmer's Guide

NOTES This interface is safe in multithreaded applications.

NAME	t_rcvconnect – receive the confirmation from a connect request
SYNOPSIS	<pre>cc [flag ...] file ... -lnsl [library ...] #include <tiuser.h> int t_rcvconnect(int <i>fildev</i>, struct <i>t_call</i> *<i>call</i>);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>This function enables a calling transport user to determine the status of a previously sent connect request and is used in conjunction with t_connect(3N) to establish a connection in asynchronous mode. The connection will be established on successful completion of this function.</p> <p><i>fildev</i> identifies the local transport endpoint where communication will be established, and <i>call</i> contains information associated with the newly established connection. <i>call</i> points to a t_call structure which contains the following members:</p> <pre> struct netbuf addr; struct netbuf opt; struct netbuf udata; int sequence;</pre> <p>netbuf is described in t_connect(3N). In <i>call</i>, addr returns the protocol address associated with the responding transport endpoint, opt presents any protocol-specific information associated with the connection, udata points to optional user data that may be returned by the destination transport user during connection establishment, and sequence has no meaning for this function.</p> <p>The maxlen (see netbuf in t_connect(3N)) field of each argument must be set before issuing this function to indicate the maximum size of the buffer for each. However, <i>call</i> may be NULL, in which case no information is given to the user on return from t_rcvconnect(). By default, t_rcvconnect() executes in synchronous mode and waits for the connection to be established before returning. On return, the addr, opt, and udata fields reflect values associated with the connection.</p> <p>If O_NDELAY or O_NONBLOCK is set (using t_open(3N) or fcntl(2)), t_rcvconnect() executes in asynchronous mode, and reduces to a poll for existing connect confirmations. If none are available, t_rcvconnect() fails and returns immediately without waiting for the connection to be established. (See TNODATA below.) t_rcvconnect() must be re-issued at a later time to complete the connection establishment phase and retrieve the information returned in <i>call</i>.</p>
RETURN VALUES	t_rcvconnect() returns 0 on success. On failure t_rcvconnect() returns -1, t_errno is set to indicate the error, and possibly errno is set.

ERRORS	On failure, t_errno will be 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 an incoming argument is not sufficient to store the value of that argument and the connect information to be returned in <i>call</i> will be discarded. The provider's state, as seen by the user, will be changed to DATAXFER .
TLOOK	An asynchronous event has occurred on this transport connection and requires immediate attention.
TNODATA	O_NDELAY or O_NONBLOCK was set, but a connect confirmation has not yet arrived.
TNOTSUPPORT	This function is not supported by the underlying transport provider.
TSYSERR	A system error has occurred during execution of this function, errno will be set to the specific error.

SEE ALSO **fcntl(2)**, **t_accept(3N)**, **t_bind(3N)**, **t_connect(3N)**, **t_listen(3N)**, **t_open(3N)**

Network Interfaces Programmer's Guide

NOTES This interface is safe in multithreaded applications.

NAME	t_rcvdis – retrieve information from disconnect
SYNOPSIS	<pre>cc [flag ...] file ... -lnsl [library ...] #include <tiuser.h> int t_rcvdis(int <i>fildev</i>, struct t_discon *<i>discon</i>);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>This function is used to identify the cause of a disconnect, and to retrieve any user data sent with the disconnect. <i>fildev</i> identifies the local transport endpoint where the connection existed, and <i>discon</i> points to a t_discon structure containing the following members:</p> <pre> struct netbuf udata; int reason; int sequence; /* struct netbuf { unsigned int maxlen; unsigned int len; char *buf; }; */</pre> <p>reason specifies the reason for the disconnect through a protocol-dependent reason code, udata identifies any user data that was sent with the disconnect, and sequence may identify an outstanding connect indication with which the disconnect is associated. sequence is only meaningful when t_rcvdis() is issued by a passive transport user who has executed one or more t_listen(3N) functions and is processing the resulting connect indications. If a disconnect indication occurs, sequence can be used to identify which of the outstanding connect indications is associated with the disconnect.</p> <p>If a user does not care if there is incoming data and does not need to know the value of reason or sequence, <i>discon</i> may be NULL and any user data associated with the disconnect will be discarded. However, if a user has retrieved more than one outstanding connect indication (using t_listen(3N)) and <i>discon</i> is NULL, the user will be unable to identify which connect indication the disconnect is associated with.</p>
RETURN VALUES	t_rcvdis returns 0 on success. On failure t_rcvdis returns -1, t_errno is set to indicate the error, and possibly errno is set.
ERRORS	<p>On failure, t_errno will be set to one of the following:</p> <p>TBADF The specified file descriptor does not refer to a transport endpoint.</p>

TBUFOVFLW	The number of bytes allocated for incoming data is not sufficient to store the data. The provider's state, as seen by the user, will change to T_IDLE , and the disconnect indication information to be returned in <i>discon</i> will be discarded.
TNODIS	No disconnect indication currently exists on the specified transport endpoint.
TNOTSUPPORT	This function is not supported by the underlying transport provider.
TSYSERR	A system error has occurred during execution of this function, errno will be set to the specific error.

SEE ALSO **t_connect(3N)**, **t_listen(3N)**, **t_open(3N)**, **t_snddis(3N)**

Network Interfaces Programmer's Guide

NOTES This interface is safe in multithreaded applications.

NAME	t_rcvrel – acknowledge receipt of an orderly release indication
SYNOPSIS	cc [<i>flag</i> ...] <i>file</i> ... -lnsl [<i>library</i> ...] #include <tiuser.h> int t_rcvrel(int <i>fildev</i>);
MT-LEVEL	MT-Safe
DESCRIPTION	This function is used to acknowledge receipt of an orderly release indication. <i>fildev</i> identifies the local transport endpoint where the connection exists. After receipt of this indication, the user should not attempt to receive more data because such an attempt will block forever. However, the user may continue to send data over the connection if t_sndrel() has not been issued by the user. This function is an optional service of the transport provider, and is only supported if the transport provider returned service type T_COTS_ORD on t_open() or t_getinfo() .
RETURN VALUES	t_rcvrel() returns 0 on success. On failure t_rcvrel() returns -1, t_errno is set to indicate the error, and possibly errno is set.
ERRORS	On failure, t_errno will be 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. TNOREL No orderly release indication currently exists on the specified transport endpoint. TNOTSUPPORT This function is not supported by the underlying transport provider. TSYSERR A system error has occurred during execution of this function, errno will be set to the specific error.
SEE ALSO	t_open(3N) , t_sndrel(3N) <i>Network Interfaces Programmer's Guide</i>
NOTES	This interface is safe in multithreaded applications.

NAME	t_rcvudata – receive a data unit						
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lnsl [<i>library</i> ...] #include <tiuser.h> int t_rcvudata(int <i>fildev</i>, struct t_unitdata *<i>unitdata</i>, int *<i>flags</i>);</pre>						
MT-LEVEL	MT-Safe						
DESCRIPTION	<p>This function is used in connectionless mode to receive a data unit from another transport user. <i>fildev</i> identifies the local transport endpoint through which data will be received, <i>unitdata</i> holds information associated with the received data unit, and <i>flags</i> is set on return to indicate that the complete data unit was not received. <i>unitdata</i> points to a t_unitdata structure containing the following members:</p> <pre style="margin-left: 40px;">struct netbuf addr; struct netbuf opt; struct netbuf udata;</pre> <p>The maxlen (see netbuf in t_connect(3N)) field of addr, opt, and udata must be set before issuing this function to indicate the maximum size of the buffer for each.</p> <p>On return from this call, addr specifies the protocol address of the sending user, opt identifies protocol-specific options that were associated with this data unit, and udata specifies the user data that was received.</p> <p>By default, t_rcvudata() operates in synchronous mode and will wait for a data unit to arrive if none is currently available. However, if O_NDELAY or O_NONBLOCK is set (using t_open(3N) or fcntl(2)), t_rcvudata() will execute in asynchronous mode and will fail if no data units are available.</p> <p>If the buffer defined in the udata field of <i>unitdata</i> is not large enough to hold the current data unit, the buffer will be filled and T_MORE will be set in <i>flags</i> on return to indicate that another t_rcvudata() should be issued to retrieve the rest of the data unit. Subsequent t_rcvudata() call(s) will return zero for the length of the address and options until the full data unit has been received.</p>						
RETURN VALUES	t_rcvudata() returns 0 on successful completion. On failure t_rcvudata() returns -1, t_errno is set to indicate the error, and possibly errno is set.						
ERRORS	<p>On failure, t_errno will be set to one of the following:</p> <table border="0" style="margin-left: 20px;"> <tr> <td style="padding-right: 20px;">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 is greater than zero 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> </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 is greater than zero 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.
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 is greater than zero 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_NDELAY or 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.

TSYSERR A system error has occurred during execution of this function, **errno** will be set to the specific error.

SEE ALSO **fcntl(2)**, **t_connect(3N)**, **t_rcvuderr(3N)**, **t_sndudata(3N)**

Network Interfaces Programmer's Guide

NOTES This interface is safe in multithreaded applications.

NAME	t_rcvuderr – receive a unit data error indication										
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lnsl [<i>library</i> ...] #include <tiuser.h> int t_rcvuderr(int <i>fildev</i>, struct t_uderr *<i>uderr</i>);</pre>										
MT-LEVEL	MT-Safe										
DESCRIPTION	<p>This function is used in connectionless mode to receive information concerning an error on a previously sent data unit, and should be issued only after 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. <i>fildev</i> identifies the local transport endpoint through which the error report will be received, and <i>uderr</i> points to a t_uderr structure containing the following members:</p> <pre> struct netbuf addr; struct netbuf opt; long error;</pre> <p>netbuf is described in t_connect(3N). The maxlen (see netbuf in t_connect(3N)) field of addr and opt must be set before issuing this function to indicate the maximum size of the buffer for each.</p> <p>On return from this call, the addr structure specifies the destination protocol address of the erroneous data unit, the opt structure identifies protocol-specific options that were associated with the data unit, and error specifies a protocol-dependent error code.</p> <p>If the user does not care to identify the data unit that produced an error, <i>uderr</i> may be set to NULL and t_rcvuderr() will simply clear the error indication without reporting any information to the user.</p>										
RETURN VALUES	t_rcvuderr() returns 0 on successful completion. On failure t_rcvuderr() returns -1, t_errno is set to indicate the error, and possibly errno is set.										
ERRORS	<p>On failure, t_errno will be 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>TBUFOVFLW</td> <td>The number of bytes allocated for the incoming protocol address or options is not sufficient to store the information. The unit data error information to be returned in <i>uderr</i> will be discarded.</td> </tr> <tr> <td>TNOTSUPPORT</td> <td>This function is not supported by the underlying transport provider.</td> </tr> <tr> <td>TNOUDERR</td> <td>No unit data error indication currently exists on the specified transport endpoint.</td> </tr> <tr> <td>TSYSERR</td> <td>A system error has occurred during execution of this function, errno will be set to the specific error.</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 is 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.	TSYSERR	A system error has occurred during execution of this function, errno will be set to the specific error.
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 is 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.										
TSYSERR	A system error has occurred during execution of this function, errno will be set to the specific error.										

SEE ALSO **t_connect(3N), t_rcvudata(3N), t_sndudata(3N)**

Network Interfaces Programmer's Guide

NOTES This interface is safe in multithreaded applications.

NAME	t_snd – send data or expedited data over a connection
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lnsl [<i>library</i> ...] #include <tiuser.h> int t_snd(int <i>fildev</i>, char *<i>buf</i>, unsigned <i>nbytes</i>, int <i>flags</i>);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>This function is used to send either normal or expedited data. <i>fildev</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>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_NDELAY or O_NONBLOCK is set (using t_open(3N) or fcntl(2)), t_snd() will execute in asynchronous mode, and will fail immediately if there are flow control restrictions.</p> <p>Even when there are no flow control restrictions, t_snd() will wait if STREAMS internal resources are not available, regardless of the state of O_NDELAY or O_NONBLOCK.</p> <p>On successful completion, t_snd() returns the number of bytes accepted by the transport provider. Normally this will equal the number of bytes specified in <i>nbytes</i>. However, if O_NDELAY or O_NONBLOCK is set, it is possible that only part of the data will be accepted by the transport provider. In this case, t_snd() will set T_MORE for the data that was sent (see below) and will return a value less than <i>nbytes</i>. If <i>nbytes</i> is zero and sending of zero bytes is not supported by the underlying transport provider, t_snd() will return -1 with t_errno set to TBADDATA. A return value of zero indicates that the request to send a zero-length data message was sent to the provider.</p> <p>If T_EXPEDITED is 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>If T_MORE is set in <i>flags</i>, or is set as described above, an indication is sent to the transport provider that the transport service data unit (TSDU) or expedited transport service data unit (ETSDU) is being sent through multiple t_snd() calls. Each t_snd() with the T_MORE flag set indicates that another t_snd() will follow with more data for the current TSDU. The end of the TSDU (or ETSDU) is identified by a t_snd() 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 should be ignored.</p>

The size of each **TSDU** or **ETSDU** must not exceed the limits of the transport provider as returned by **t_open(3N)** or **t_getinfo(3N)**. If the size is exceeded, a **TSYSERR** with system error **EPROTO** will occur. However, the **t_snd()** may not fail because **EPROTO** errors may not be reported immediately. In this case, a subsequent call that accesses the transport endpoint will fail with the associated **TSYSERR**.

If **t_snd()** is issued from the **T_IDLE** state, the provider may silently discard the data. If **t_snd()** is issued from any state other than **T_DATAXFER**, **T_INREL**, or **T_IDLE**, the provider will generate a **TSYSERR** with system error **EPROTO** (which may be reported in the manner described above).

RETURN VALUES

On successful completion, **t_snd()** returns the number of bytes accepted by the transport provider. On failure **t_snd()** returns **-1**, **t_errno** is set to indicate the error, and possibly **errno** is set.

ERRORS

On failure, **t_errno** will be set to one of the following:

TBADDATA	<i>nbytes</i> is zero and sending zero bytes is not supported by the transport provider.
TBADDF	The specified file descriptor does not refer to a transport endpoint.
TFLOW	O_NDELAY or O_NONBLOCK was set, but the flow control mechanism prevented the transport provider from accepting data at this time.
TNOTSUPPORT	This function is not supported by the underlying transport provider.
TSYSERR	A system error (see intro(2)) has been detected during execution of this function.

SEE ALSO

fcntl(2), **t_getinfo(3N)**, **t_open(3N)**, **t_rcv(3N)**

Network Interfaces Programmer's Guide

NOTES

This interface is safe in multithreaded applications.

NAME	t_snddis – send user-initiated disconnect request								
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lnsl [<i>library</i> ...] #include <tiuser.h> int t_snddis(int <i>fildev</i>, struct t_call *<i>call</i>);</pre>								
MT-LEVEL	MT-Safe								
DESCRIPTION	<p>This function is used to initiate an abortive release on an already established connection or to reject a connect request. <i>fildev</i> identifies the local transport endpoint of the connection, and <i>call</i> specifies information associated with the abortive release. <i>call</i> points to a t_call structure that contains the following members:</p> <pre> struct netbuf addr; struct netbuf opt; struct netbuf udata; int sequence;</pre> <p>netbuf is described in t_connect(3N). The values in <i>call</i> have different semantics, depending on the context of the call to t_snddis(). When rejecting a connect request, <i>call</i> must be non-NULL and contain a valid value of sequence to identify uniquely the rejected connect indication to the transport provider. The addr and opt fields of <i>call</i> are ignored. In all other cases, <i>call</i> need only be used when data is being sent with the disconnect request. The addr, opt, and sequence fields of the t_call structure are ignored. If the user does not wish to send data to the remote user, the value of <i>call</i> may be NULL.</p> <p>udata 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 discon field of the <i>info</i> argument of t_open(3N) or t_getinfo(3N). If the <i>len</i> field of udata is zero, no data will be sent to the remote user.</p>								
RETURN VALUES	t_snddis() returns 0 on success. On failure t_snddis() returns -1, t_errno is set to indicate the error, and possibly errno is set.								
ERRORS	<p>On failure, t_errno will be set to one of the following:</p> <table border="0"> <tr> <td style="padding-right: 10px;">TBADDATA</td> <td>The amount of user data specified was not within the bounds allowed by the transport provider. The transport provider's outgoing queue will be flushed, so data may be lost.</td> </tr> <tr> <td>TBADF</td> <td>The specified file descriptor does not refer to a transport endpoint.</td> </tr> <tr> <td>TBADSEQ</td> <td>An invalid sequence number was specified, or a NULL call structure was specified when rejecting a connect request. The transport provider's outgoing queue will be flushed, so data may be lost.</td> </tr> <tr> <td>TLOOK</td> <td>An asynchronous event has occurred on this transport endpoint and requires immediate attention.</td> </tr> </table>	TBADDATA	The amount of user data specified was not within the bounds allowed by the transport provider. The transport provider's outgoing queue will be flushed, so data may be lost.	TBADF	The specified file descriptor does not refer to a transport endpoint.	TBADSEQ	An invalid sequence number was specified, or a NULL call structure was specified when rejecting a connect request. The transport provider's outgoing queue will be flushed, so data may be lost.	TLOOK	An asynchronous event has occurred on this transport endpoint and requires immediate attention.
TBADDATA	The amount of user data specified was not within the bounds allowed by the transport provider. The transport provider's outgoing queue will be flushed, so data may be lost.								
TBADF	The specified file descriptor does not refer to a transport endpoint.								
TBADSEQ	An invalid sequence number was specified, or a NULL call structure was specified when rejecting a connect request. The transport provider's outgoing queue will be flushed, so data may be lost.								
TLOOK	An asynchronous event has occurred on this transport endpoint and requires immediate attention.								

TNOTSUPPORT	This function is not supported by the underlying transport provider.
TOUTSTATE	The function was issued in the wrong sequence. The transport provider's outgoing queue may be flushed, so data may be lost.
TSYSERR	A system error has occurred during execution of this function, errno will be set to the specific error.

SEE ALSO [t_connect\(3N\)](#), [t_getinfo\(3N\)](#), [t_listen\(3N\)](#), [t_open\(3N\)](#)

Network Interfaces Programmer's Guide

NOTES This interface is safe in multithreaded applications.

NAME	t_sndrel – initiate an orderly release
SYNOPSIS	<pre>cc [flag ...] file ... -lnsl [library ...] #include <tiuser.h> int t_sndrel(int fildes);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>This function is used to initiate an orderly release of a transport connection and indicates to the transport provider that the transport user has no more data to send. <i>fildes</i> identifies the local transport endpoint where the connection exists. After issuing t_sndrel(), 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>This function is an optional service of the transport provider, and is only supported if the transport provider returned service type T_COTS_ORD on t_open() or t_getinfo().</p> <p>If t_sndrel() is issued from an invalid state, the provider will generate an EPROTO protocol error; however, this error may not occur until a subsequent reference to the transport endpoint.</p>
RETURN VALUES	<p>t_sndrel() returns 0 on success. On failure t_sndrel() returns -1, t_errno is set to indicate the error, and possibly errno is set.</p> <p>TSYSERR – A system error has occurred during execution of this function, errno will be set to the specific error.</p>
ERRORS	<p>On failure, t_errno will be set to one of the following:</p> <p>TBADF The specified file descriptor does not refer to a transport endpoint.</p> <p>TFLOW O_NDELAY or O_NONBLOCK was set, but the flow control mechanism prevented the transport provider from accepting the function at this time.</p> <p>TNOTSUPPORT This function is not supported by the underlying transport provider.</p>
SEE ALSO	<p>t_open(3N), t_rcvrel(3N) <i>Network Interfaces Programmer's Guide</i></p>
NOTES	This interface is safe in multithreaded applications.

NAME	t_sndudata – send a data unit
SYNOPSIS	<pre>cc [flag ...] file ... -lnsl [library ...] #include <tiuser.h> int t_sndudata(int <i>fildev</i>, struct t_unitdata *<i>unitdata</i>);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>This function is used in connectionless mode to send a data unit to another transport user. <i>fildev</i> identifies the local transport endpoint through which data will be sent, and <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>netbuf is described in t_connect(3N). In <i>unitdata</i>, addr specifies the protocol address of the destination user, opt identifies protocol-specific options that the user wants associated with this request, and udata 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 len field of opt to zero. In this case, the provider may use default options.</p> <p>If the len field of udata is zero, and the sending of zero bytes is not supported by the underlying transport provider, t_sndudata will return -1 with t_errno set to TBADDATA.</p> <p>By default, t_sndudata() 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_NDELAY or O_NONBLOCK is set (using t_open(3N) or fcntl(2)), t_sndudata() will execute in asynchronous mode and will fail under such conditions.</p> <p>If t_sndudata() is issued from an invalid state, or if the amount of data specified in udata exceeds the TSDU size as returned in the tsdu field of the <i>info</i> argument of t_open(3N) or t_getinfo(3N), the provider will generate an EPROTO protocol error. (See TSYSERR below.) If the state is invalid, this error may not occur until a subsequent reference is made to the transport endpoint.</p>
RETURN VALUES	t_sndudata() returns 0 on successful completion. On failure t_sndudata() returns -1 , t_errno is set to indicate the error, and possibly errno is set.
ERRORS	<p>On failure, t_errno will be set to one of the following:</p> <p>TBADDATA nbytes is zero and sending zero bytes is not supported by the transport provider.</p> <p>TBADF The specified file descriptor does not refer to a transport endpoint.</p> <p>TFLOW] O_NDELAY or O_NONBLOCK was set, but the flow control mechanism prevented the transport provider from accepting data at this time.</p>

TNOTSUPPORT This function is not supported by the underlying transport provider.
TSYSERR A system error has occurred during execution of this function, **errno** will be set to the specific error.

SEE ALSO **fcntl(2)**, **t_connect(3N)**, **t_getinfo(3N)**, **t_rcvudata(3N)**, **t_rcvuderr(3N)**
Network Interfaces Programmer's Guide

NOTES This interface is safe in multithreaded applications.

NAME	t_strerror – get error message string
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lnsl [<i>library</i> ...] #include <tiuser.h> char *t_strerror(int <i>errnum</i>);</pre>
MT-LEVEL	Unsafe
DESCRIPTION	<p>The t_strerror() function maps the error number in <i>errnum</i> that corresponds to a TLI error to a language-independent error message string and returns a pointer to that string. The string pointed to will not be modified by the program, but may be overwritten by a subsequent call to the t_strerror() function. The string is not terminated by a newline character. The language for the error message strings written by t_strerror() is implementation-defined. If an error code is unknown and the language is English, t_strerror() returns the string:</p> <pre> "<error>: error unknown"</pre> <p>where <error> is the error number supplied as input. In other languages, an equivalent text is provided.</p>
SEE ALSO	gettext(3l) , perror(3C) , setlocale(3C) , strerror(3C) , t_error(3N)
VALID STATES	All – apart from T_UNINIT .
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	t_sync – synchronize transport library														
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lnsl [<i>library</i> ...] #include <tiuser.h> int t_sync(int <i>fildev</i>);</pre>														
MT-LEVEL	MT-Safe														
DESCRIPTION	<p>For the transport endpoint specified by <i>fildev</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 a raw file descriptor (obtained using open(2), dup(2), or as a result of a fork(2) and exec(2)) to an initialized transport endpoint, assuming that file descriptor referenced a transport provider. This function also allows two cooperating processes to synchronize their interaction with a transport provider.</p> <p>For example, if a process fork(2)s a new process and issues an exec(2), 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 provider. t_sync() returns the current state of the provider 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 provider's state <i>after</i> a t_sync() is issued.</p> <p>If the provider is undergoing a state transition when t_sync() is called, the function will fail.</p>														
RETURN VALUES	<p>t_sync() returns the state of the transport provider on successful completion. t_sync() returns -1 on failure, t_errno is set to indicate the error, and possibly errno is set. The state returned may be one of the following:</p> <table border="0"> <tr> <td style="padding-right: 20px;">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 orderly release (waiting for an orderly release indication)</td> </tr> <tr> <td>T_INREL</td> <td>incoming orderly release (waiting for an orderly release request)</td> </tr> </table>	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)
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 will be set to one of the following: TBADF The specified file descriptor does not refer to a transport endpoint. TSTATECHNG The transport provider is undergoing a state change. TSYSERR A system error has occurred during execution of this function, errno will be set to the specific error.
SEE ALSO	dup(2), exec(2), fork(2), open(2) <i>Network Interfaces Programmer's Guide</i>
NOTES	This interface is safe in multithreaded applications.

NAME	t_unbind – disable a transport endpoint
SYNOPSIS	<pre>cc [flag ...] file ... -lnsl [library ...] #include <tiuser.h> int t_unbind(int fildes);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	The t_unbind() function disables the transport endpoint specified by <i>fildes</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.
RETURN VALUES	t_unbind() returns 0 on success. On failure t_unbind() returns -1, t_errno is set to indicate the error, and possibly errno is set.
ERRORS	On failure, t_errno will be 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. TOUTSTATE The function was issued in the wrong sequence. TSYSERR A system error has occurred during execution of this function, errno will be set to the specific error.
SEE ALSO	t_bind(3N) <i>Network Interfaces Programmer's Guide</i>
NOTES	This interface is safe in multithreaded applications.

NAME	tcsetpgrp – set foreground process group id of terminal
SYNOPSIS	<pre>#include <unistd.h> int tcsetpgrp(int <i>fildes</i>, pid_t <i>pgid</i>);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	tcsetpgrp() sets the foreground process group ID of the terminal specified by <i>fildes</i> to <i>pgid</i> . The file associated with <i>fildes</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 a value of 0. Otherwise, a value of -1 is returned and errno is set to indicate the error.
ERRORS	tcsetpgrp() fails if one or more of the following is true: EBADF The <i>fildes</i> argument is not a valid file descriptor. EINVAL The <i>fildes</i> argument is a terminal that does not support tcsetpgrp() , or <i>pgid</i> is not a valid process group ID. 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. EPERM <i>pgid</i> does not match the process group ID of an existing process in the same session as the calling process.
SEE ALSO	termio(7)

The **tcsetattr()** function sets the parameters associated with the terminal (unless support is required from the underlying hardware that is not available) from the **termios** structure referenced by *termios_p* as follows:

If *optional_actions* is **TCSANOW**, the change occurs immediately.

If *optional_actions* is **TCSADRAIN**, the change occurs after all output written to *fildev* has been transmitted. This function should be used when changing parameters that affect output.

If *optional_actions* is **TCSAFLUSH**, the change occurs after all output written to the object referred to by *fildev* has been transmitted, and all input that has been received but not read is discarded before the change is made.

The symbolic constants for the values of *optional_actions* are defined in **<termios.h>**.

Line Control

If the terminal is using asynchronous serial data transmission, the **tcsendbreak()** function causes transmission of a continuous stream of zero-valued bits for a specific duration. If *duration* is zero, it causes transmission of zero-valued bits for at least 0.25 seconds, and not more than 0.5 seconds. If *duration* is not zero, it behaves in a way similar to **tcdrain()**.

If the terminal is not using asynchronous serial data transmission, the **tcsendbreak()** function sends data to generate a break condition or returns without taking any action.

The **tcdrain()** function waits until all output written to the object referred to by *fildev* has been transmitted.

The **tcflush()** function discards data written to the object referred to by *fildev* but not transmitted, or data received but not read, depending on the value of *queue_selector*:

If *queue_selector* is **TCIFLUSH**, it flushes data received but not read.

If *queue_selector* is **TCOFLUSH**, it flushes data written but not transmitted.

If *queue_selector* is **TCIOFLUSH**, it flushes both data received but not read, and data written but not transmitted.

The **tcflow()** function suspends transmission or reception of data on the object referred to by *fildev*, depending on the value of *action*:

If *action* is **TCOOFF**, it suspends output.

If *action* is **TCOON**, it restarts suspended output.

If *action* is **TCIOFF**, the system transmits a STOP character, which causes the terminal device to stop transmitting data to the system.

If *action* is **TCION**, the system transmits a START character, which causes the terminal device to start transmitting data to the system.

Get and Set Baud Rate

The baud rate functions get and set the values of the input and output baud rates in the **termios** structure. The effects on the terminal device described below do not become effective until the **tcsetattr()** function is successfully called.

The input and output baud rates are stored in the **termios** structure. The values shown in the table are supported. The names in this table are defined in `<termios.h>`.

Name	Description	Name	Description
B0	Hang up	B600	600 baud
B50	50 baud	B1200	1200 baud
B75	75 baud	B1800	1800 baud
B110	110 baud	B2400	2400 baud
B134	134.5 baud	B4800	4800 baud
B150	150 baud	B9600	9600 baud
B200	200 baud	B19200	19200 baud
B300	300 baud	B38400	38400 baud

cfgetospeed() returns the output baud rate stored in the **termios** structure pointed to by *termios_p*.

cfsetospeed() sets the output baud rate stored in the **termios** structure pointed to by *termios_p* to *speed*. The zero baud rate, **B0**, is used to terminate the connection. If **B0** is specified, the modem control lines are no longer be asserted. Normally, this disconnects the line.

cfgetispeed() returns the input baud rate stored in the **termios** structure pointed to by *termios_p*.

cfsetispeed() sets the input baud rate stored in the **termios** structure pointed to by *termios_p* to *speed*. If the input baud rate is set to zero, the input baud rate is specified by the value of the output baud rate. Both **cfsetispeed()** and **cfsetospeed()** return a value of zero if successful and **-1** to indicate an error. This refers both to changes to baud rates not supported by the hardware, and to changes setting the input and output baud rates to different values if the hardware does not support this.

Get and Set Terminal Foreground Process Group ID

tcsetpgrp() sets the foreground process group ID of the terminal specified by *fildev* to *pgid*. The file associated with *fildev* must be the controlling terminal of the calling process and the controlling terminal must be currently associated with the session of the calling process. *pgid* must match a process group ID of a process in the same session as the calling process.

tcgetpgrp() returns the foreground process group ID of the terminal specified by *fildev*. **tcgetpgrp()** 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.

Get Terminal Session ID RETURN VALUES

tcgetsid() returns the session ID of the terminal specified by *fildev*.

On success, **tcgetpgrp()** returns the process group ID of the foreground process group associated with the specified terminal. Otherwise, it returns **-1** and sets **errno** to indicate the error.

On success, **tcgetsid()** returns the session ID associated with the specified terminal. Otherwise, it returns **-1** and sets **errno** to indicate the error.

On success, all other functions return a value of **0**. Otherwise, they return **-1** and set **errno** to indicate the error.

ERRORS

All of the functions fail if one of more of the following is true:

EBADF The *fildev* argument is not a valid file descriptor.

ENOTTY The file associated with *fildev* is not a terminal.

tcsetattr() also fails if the following is true:

EINVAL The *optional_actions* argument is not a proper value, or an attempt was made to change an attribute represented in the **termios** structure to an unsupported value.

tcsendbreak() also fails if the following is true:

EINVAL The device does not support the **tcsendbreak()** function.

tcdrain() also fails if one or more of the following is true:

EINTR A signal interrupted the **tcdrain()** function.

EINVAL The device does not support the **tcdrain()** function.

tcflush() also fails if the following is true:

EINVAL The device does not support the **tcflush()** function or the *queue_selector* argument is not a proper value.

tcflow() also fails if the following is true:

EINVAL The device does not support the **tcflow()** function or the *action* argument is not a proper value.

tcgetpgrp() also fails if the following is true:

ENOTTY the calling process does not have a controlling terminal, or *fildev* does not refer to the controlling terminal.

tcsetpgrp() also fails if the following is true:

EINVAL *pgid* is not a valid process group ID .

ENOTTY the calling process does not have a controlling terminal, or *fildev* does not refer to the controlling terminal, or the controlling terminal is no longer associated with the session of the calling process.

EPERM *pgid* does not match the process group of an existing process in the same session as the calling process.

tcgetsid() also fails if the following is true:

EACCES *fildev* is a terminal that is not allocated to a session.

SEE ALSO

setpgid(2), **getsid(2)**, **termio(7)**

NAME	thr_create – create a new thread of control
SYNOPSIS	<pre>#include <thread.h> int thr_create(void *stack_base, size_t stack_size, void *(*start_routine)(void *), void *arg, long flags, thread_t *new_thread); size_t thr_min_stack(void);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>thr_create() adds a new thread of control to the current process. The new thread begins execution by calling the function specified by <i>start_routine</i> with a single argument, <i>arg</i>. If <i>start_routine</i> returns, the thread exits with the exit status set to the value returned by <i>start_routine</i> (see thr_exit(3T)).</p> <p>The new thread will use the stack starting at the address specified by <i>stack_base</i> and continuing for <i>stack_size</i> bytes. <i>stack_size</i> must be greater than the value returned by thr_min_stack(). If <i>stack_base</i> is NULL then thr_create() allocates a stack for the new thread with at least <i>stack_size</i> bytes. If <i>stack_size</i> is zero then a default size is used. If <i>stack_size</i> is not zero then it must be greater than the value returned by thr_min_stack(). A stack of minimum size may not accommodate the stack frame for <i>start_function</i>. If a stack size is specified, it must take into account the requirements <i>start_function</i> and the functions that it may call in turn, in addition to the minimum requirement.</p> <p><i>flags</i> specifies additional attributes for the created thread. The value in <i>flags</i> is constructed from the bitwise inclusive OR of the following:</p> <p>THR_SUSPENDED The new thread is created suspended and will not execute <i>start_routine</i> until it is started by thr_continue().</p> <p>THR_DETACHED The new thread is created detached. Its thread ID and other resources may be re-used as soon as the thread terminates. A detached thread cannot be waited for via thr_join().</p> <p>THR_BOUND The new thread is created permanently bound to an LWP (i.e. it is a <i>bound thread</i>).</p> <p>THR_NEW_LWP The desired concurrency level for unbound threads is increased by one. This is similar to incrementing concurrency by one via thr_setconcurrency(3T). Typically, this adds a new LWP to the pool of LWPs running unbound threads.</p> <p>THR_DAEMON The thread is marked as a daemon. The process will exit when all non-daemon threads exit.</p> <p>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.</p> <p>When <i>new_thread</i> 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.</p>

The new thread inherits the calling thread's signal mask and priority. Pending signals are not inherited.

RETURN VALUES

Zero is returned when successful. A non-zero value indicates an error.

ERRORS

If any of the following conditions are detected, **thr_create()** fails and returns the corresponding value:

- | | |
|---------------|---|
| EAGAIN | A system limit is exceeded, e.g., too many LWPs were created. |
| ENOMEM | Not enough memory was available to create the new thread. |
| EINVAL | <i>stack_base</i> is not NULL and <i>stack_size</i> is less than the value returned by thr_min_stack() . |
| EINVAL | <i>stack_base</i> is NULL and <i>stack_size</i> is not zero and is less than the value returned by thr_min_stack() . |

EXAMPLES

This 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 examples 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, NULL, &orig_mask);

```

SEE ALSO

_lwp_create(2), **thr_continue(3T)**, **thr_exit(3T)**, **thr_join(3T)**, **thr_setconcurrency(3T)**

NOTES

Typically, thread stacks allocated by **thr_create()** begin on page boundaries and any specified 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.

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.

If there is no explicit synchronization, an unsuspended, detached thread can die and have its thread ID re-assigned to another new thread before its creator returns from **thr_create()**.

NAME	thr_exit – thread termination
SYNOPSIS	<pre>#include <thread.h> void thr_exit(void *status);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>thr_exit() terminates the calling thread. All thread-specific data bindings are released (see thr_keycreate(3T)). If the calling thread is not detached, then the thread's ID and the exit status specified by <i>status</i> are retained until it is waited for (see thr_join(3T)). Otherwise, <i>status</i> is ignored and the thread's ID may be reclaimed immediately.</p> <p>If the calling thread is the last non-daemon thread in the process (see thr_create(3T)), then the process terminates with a status of zero (see exit(3C)). If the initial thread returns from <i>main()</i> then the process exits with a status equal to the return value.</p>
SEE ALSO	exit(3C) , thr_create(3T) , thr_join(3T) , thr_keycreate(3T)

NAME	thr_join – wait for thread termination
SYNOPSIS	<pre>#include <thread.h> int thr_join(thread_t wait_for, thread_t *departed, void **status);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>thr_join() blocks the calling thread until the thread specified by <i>wait_for</i> terminates. The specified thread must be in the current process and must not be detached (see thr_create(3T)). If <i>wait_for</i> is (thread_t)0, then 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. If <i>status</i> is not NULL, it points to a location that is set to the exit status of the terminated thread if thr_join() returns successfully.</p> <p>Multiple threads cannot wait for the same thread to terminate; one thread will return successfully and the others will fail with an error of ESRCH.</p>
RETURN VALUES	Zero is returned when successful. A non-zero value indicates an error.
ERRORS	If any of the following conditions are detected, thr_join() fails and returns the corresponding value: ESRCH <i>wait_for</i> is not a valid, undetached thread in the current process. EDEADLK <i>wait_for</i> specifies the calling thread.
SEE ALSO	thr_create(3T)

NAME	thr_keycreate, thr_setspecific, thr_getspecific – thread-specific data
SYNOPSIS	<pre>#include <thread.h> int thr_keycreate(thread_key_t *keyp, void (*destructor)(void *value)); int thr_setspecific(thread_key_t key, void *value); int thr_getspecific(thread_key_t key, void **valuep);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>thr_keycreate() allocates a key that is used to identify data that is specific to each thread in the process. The key is global to all threads in the process. Once a key has been created each thread may bind a value to the key. The values are specific to the binding thread and are maintained for each thread independently. All threads initially have the value NULL associated with the key when it is created. When thr_keycreate() returns successfully the allocated key is stored in the location pointed to by <i>keyp</i>. The caller must ensure that storage and access to this key are properly synchronized.</p> <p>An optional destructor function, specified by <i>destructor</i>, may be associated with each key. If a key has a non-NULL destructor function and the thread has a non-NULL value associated with that key, the destructor function is called with the current associated value when the thread exits. The order in which the destructor functions are called for all the allocated keys is unspecified.</p> <p>thr_setspecific() binds <i>value</i> to the thread-specific data key, <i>key</i>, for the calling thread.</p> <p>thr_getspecific() stores the current value bound to <i>key</i> for the calling thread into the location pointed to by <i>valuep</i>.</p>
RETURN VALUES	Zero is returned when successful. A non-zero value indicates an error.
ERRORS	<p>If any of the following conditions are detected, thr_keycreate() fails and returns the corresponding value:</p> <p>EAGAIN The key name space is exhausted.</p> <p>If any of the following conditions are detected, thr_keycreate() or thr_setspecific() fails and returns the corresponding value:</p> <p>ENOMEM Not enough memory is available.</p> <p>If any of the following conditions are detected, thr_setspecific() or thr_getspecific() fails and returns the corresponding value:</p> <p>EINVAL <i>key</i> is invalid.</p>

EXAMPLES

This examples shows the use of thread-specific data in a function that can be called from more than one thread without special initialization.

```
static mutex_t keylock;
static thread_key_t key;
static int once = 0;

func()
{
    void *ptr;

    if (!once) {
        (void) mutex_lock(&keylock);
        if (!once) {
            (void) thr_keycreate(&key, free);
            once++;
        }
        (void) mutex_unlock(&keylock);
    }
    (void) thr_getspecific(key, (void *)&ptr);
    if (ptr == NULL) {
        ptr = malloc(SIZE);
        (void) thr_setspecific(key, ptr);
    }
}
```

SEE ALSO

thr_exit(3T)

NAME	thr_kill – send a signal to a thread
SYNOPSIS	<pre>#include <thread.h> #include <signal.h> int thr_kill(thread_t target_thread, int sig);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	thr_kill() sends the signal, <i>sig</i> , to the thread specified by <i>target_thread</i> . <i>target_thread</i> must be a thread within the same process as the calling thread. <i>sig</i> must be one from the list given in signal(5) . If <i>sig</i> is zero, error checking is performed but no signal is actually sent. This can be used to check the validity of <i>target_thread</i> .
RETURN VALUES	Zero is returned when successful. A non-zero value indicates an error.
ERRORS	If any of the following conditions detected, thr_kill() fails and returns the corresponding value: EINVAL <i>sig</i> is not a valid signal number. ESRCH <i>target_thread</i> cannot be found in the current process.
SEE ALSO	kill(2) , sigaction(2) , thr_sigsetmask(3T) , signal(5)

NAME	thr_min_stack – minimum stack space for a thread
SYNOPSIS	<pre>#include <thread.h> size_t thr_min_stack(void);</pre>
MT-LEVEL	MT-Safe
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() + <i><some increment></i>.</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>
SEE ALSO	thr_create(3T)

NAME	thr_self – get thread identifier
SYNOPSIS	#include <thread.h> thread_t thr_self(void)
MT-LEVEL	MT-Safe
DESCRIPTION	thr_self() returns the ID of the calling thread.
SEE ALSO	thr_create(3T)

NAME	thr_setconcurrency, thr_getconcurrency – get/set thread concurrency level
SYNOPSIS	<pre>#include <thread.h> int thr_setconcurrency(int new_level); int thr_getconcurrency(void);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>Unbound threads in a process (see thr_create(3T)) may or may not be required to be simultaneously active. 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 zero, 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>thr_setconcurrency() returns zero when successful. A non-zero value indicates an error code.</p> <p>thr_getconcurrency() always returns the current value for the desired concurrency level.</p>
ERRORS	<p>If any of the following conditions are detected, thr_setconcurrency() fails and returns the corresponding value:</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>
SEE ALSO	thr_create(3T) .

NAME	thr_setprio, thr_getprio – set/get a thread priority
SYNOPSIS	<pre>#include <thread.h> int thr_setprio(thread_t target_thread, int pri); int thr_getprio(thread_t target_thread, int *pri);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>Each thread has a priority which it inherits from its creator. thr_setprio() changes the priority of the thread, specified by <i>target_thread</i>, within the current process to the priority specified by <i>pri</i>. By default, threads are scheduled based on fixed priorities that range from zero, the least significant, to the largest integer. The <i>target_thread</i> will preempt lower priority threads, and will yield to higher priority threads.</p> <p>The function thr_getprio() stores the current priority for the thread specified by <i>target_thread</i> in the location pointed to by <i>pri</i>.</p>
RETURN VALUES	Zero is returned when successful. A non-zero value indicates error.
ERRORS	<p>If any of the following conditions are detected, thr_setprio() or thr_getprio() fails and returns the corresponding value:</p> <p>ESRCH <i>target_thread</i> cannot be found in the current process.</p> <p>If any of the following conditions are detected, thr_setprio() fails and returns the corresponding value:</p> <p>EINVAL The value of <i>pri</i> makes no sense for the scheduling class associated with the <i>target_thread</i>.</p>
SEE ALSO	thr_yield(3T)

NAME	thr_sigsetmask – change and/or examine calling thread's signal mask
SYNOPSIS	<pre>#include <thread.h> #include <signal.h> int thr_sigsetmask(int how, const sigset_t *set, sigset_t *oset);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>thr_sigsetmask() examines and/or changes the calling thread's signal mask. If the value of the argument <i>set</i> is not NULL, it points to a set of signals to be used to change the currently blocked set. The value of the argument <i>how</i> determines the manner in which the set is changed. <i>how</i> may have one of the following values:</p> <p>SIG_BLOCK <i>set</i> represent a set of signals to block. They are added to the current signal mask.</p> <p>SIG_UNBLOCK <i>set</i> represents a set of signals to unblock. These signals are deleted from the current signal mask.</p> <p>SIG_SETMASK <i>set</i> represents 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 space where the previous signal mask is stored. If the value of <i>set</i> is NULL, the value of <i>how</i> is not significant and the thread's signal mask is unchanged; thus, thr_sigsetmask() can be used to enquire about the currently blocked signals.</p>
RETURN VALUES	Zero is returned when successful. A non-zero value indicates an error.
ERRORS	<p>If any of the following conditions are detected, thr_sigsetmask() fails and returns the corresponding value:</p> <p>EINVAL <i>set</i> is not NULL and the value of <i>how</i> is not defined.</p> <p>EFAULT <i>set</i> or <i>oset</i> are not valid addresses</p>
SEE ALSO	sigaction(2) , sigprocmask(2)
NOTES	It is not possible to block those signals that cannot be ignored (see sigaction(2)). In addition, if using the threads library, it is not possible to block the signal SIGLWP, reserved by the threads library, and it is not possible to unblock the signal SIGWAITING, which is always blocked on all threads. This restriction is silently imposed by the threads library.

NAME	thr_suspend, thr_continue – suspend or continue thread execution
SYNOPSIS	<pre>#include <thread.h> int thr_suspend(thread_t target_thread); int thr_continue(thread_t target_thread);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>thr_suspend() 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>thr_continue() 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	Zero is returned when successful. A non-zero value indicates an error.
ERRORS	If any of the following conditions are detected, thr_suspend() or thr_continue() fails and returns the corresponding value: ESRCH <i>target_thread</i> cannot be found in the current process.
SEE ALSO	thr_create(3T)

NAME	thr_yield – yield execution to another thread
SYNOPSIS	#include <thread.h> void thr_yield(void);
MT-LEVEL	MT-Safe
DESCRIPTION	thr_yield() causes the current thread to yield its execution in favor of another thread with the same or greater priority.
SEE ALSO	thr_setprio(3T).

NAME timer_create – create a per-LWP timer

SYNOPSIS cc [*flag ...*] *file ...* -lposix4 [*library ...*]

```
#include <signal.h>
```

```
#include <time.h>
```

```
int timer_create(clockid_t clock_id, struct sigevent *evp, timer_t *timerid);
```

```
struct sigevent {
```

```
    int          sigev_notify    /* notification type */
```

```
    int          sigev_signo;    /* signal number */
```

```
    union signal sigev_value;   /* signal value */
```

```
};
```

```
union signal {
```

```
    int          sival_int;     /* integer value */
```

```
    void         *sival_ptr;   /* pointer value */
```

```
};
```

MT-LEVEL MT-Safe with exceptions

DESCRIPTION `timer_create()` creates a timer per-LWP using the specified clock, `clock_id`, as the timing base. This timer ID is unique and meaningful only within the calling LWP until the timer is deleted. This timer is initially disarmed upon return from `timer_create()`.

If `evp` is non-NULL:

then `evp` points to a **sigevent** structure, allocated by the application, which defines the asynchronous notification that will occur when the timer expires.

If the `sigev_notify` member of `evp` is `SIGEV_SIGNAL`, then the structure also contains the signal number and the application specific data value to be sent to the process. If `SA_SIGINFO` is set for the expiration signal, then the signal and application-defined value specified in the structure will be queued to the process on timer expiration. If `SA_SIGINFO` is not set for the expiration signal, then the signal specified in the structure will be sent to the LWP upon the timer expiration.

If the `sigev_notify` member is `SIGEV_NONE`, no notification will be sent.

If `evp` is NULL, and `SA_SIGINFO` is set for the expiration signal, then the default signal, `SIGALRM`, will be queued to the process and the signal data value will be set to the timer ID.

RETURN VALUES `timer_create()` returns `0` upon success and creates a `timer_t`, `timerid`, which can be passed to the timer calls; otherwise it returns `-1` and sets `errno` to indicate the error condition.

ERRORS

EAGAIN The system lacks sufficient signal queuing resources to honor the request. The calling process has already created all of the timers it is allowed by this implementation.

EINVAL The specified clock ID, *clock_id*, is not defined.

ENOSYS **timer_create()** is not supported by this implementation.

SEE ALSO **exec(2)**, **fork(2)**, **time(2)**, **clock_gettime(3R)**, **timer_delete(3R)**, **timer_settime(3R)**

NOTES Timers are not inherited by a child process across a **fork(2)** and can be disarmed and deleted by an **exec(2)**.

NAME	timer_delete – delete a per-LWP timer
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lposix4 [<i>library</i> ...] #include <time.h> int timer_delete(timer_t <i>timerid</i>);</pre>
MT-LEVEL	MT-Safe with exceptions
DESCRIPTION	timer_delete() deletes the specified timer, <i>timerid</i> , previously created by timer_create(3R) . If the timer is armed when timer_delete() is called, the behavior is as if the timer is automatically disarmed before removal.
RETURN VALUES	timer_delete() returns 0 upon success, otherwise it returns -1 and sets errno to indicate the error condition.
ERRORS	EINVAL <i>timerid</i> does not refer to a valid timer. ENOSYS timer_delete() is not supported by this implementation.
SEE ALSO	timer_create(3R)

NAME	timer_settime, timer_gettime, timer_getoverrun – high-resolution timer operations
SYNOPSIS	<pre>cc [flag ...] file ... -lposix4 [library ...] #include <time.h> int timer_settime(timer_t timerid, int flags, const struct itimerspec *value, struct itimerspec *ovalue); int timer_gettime(timer_t timerid, struct itimerspec *value); int timer_getoverrun(timer_t timerid); struct itimerspec { struct timespec it_interval; /* timer period */ struct timespec it_value; /* timer expiration */ }; struct timespec { time_t tv_sec; /* seconds */ long tv_nsec; /* and nanoseconds */ };</pre>
MT-LEVEL	Async-Safe
DESCRIPTION	<p>If <i>value->it_value</i> is non-zero, timer_settime() arms the timer, <i>timerid</i>, to next expire after the time designated by <i>value->it_value</i>. Upon expiration, an application-specified notification (see timer_create(3R)) or the default signal, SIGALRM, is queued for the calling LWP. If <i>timerid</i> was already armed when timer_settime() is called, this call resets the time until the next expiration to the value of <i>value->it_value</i>. If <i>value->it_value</i> is zero, then the timer is disarmed.</p> <p><i>value->it_value</i> may be expressed as either an absolute or relative time. If <i>flags</i> is set to TIMER_RELTIME, then the timer will initially expire relative to when the call is made. If <i>flags</i> is set to TIMER_ABSTIME, then the initial expiration will be relative to 00:00 Universal Coordinated Time, January 1, 1970. If the specified (absolute) time has already passed, timer_settime() succeeds and the expiration notification is made.</p> <p>If <i>value->it_interval</i> is non-zero, then <i>timerid</i> will be a “periodic” timer, to be reloaded to expire every <i>value->it_interval</i> seconds (nanoseconds). Otherwise, if <i>value->it_interval</i> is zero and <i>value->it_value</i> is non-zero, then <i>timerid</i> is a “one-shot” timer, which will expire only at the time specified by <i>value->it_value</i>.</p> <p>If <i>ovalue</i> is not NULL, and timer <i>timerid</i> had previously been used, then timer_settime() will store the remaining time until the previous timer expires in <i>ovalue->it_value</i>, and the previous reload interval in <i>ovalue->it_interval</i>. (If the previous timer was disarmed, <i>ovalue->it_value</i> will be set to zero). The values stored in <i>ovalue</i> by timer_settime() are the same values that would have been returned by a call to timer_gettime(timerid,...).</p> <p>timer_gettime() stores the amount of time until the specified timer, <i>timerid</i>, expires into <i>value->it_value</i>, and the timer’s reload value into <i>value->it_interval</i>.</p>

Only a single signal can be queued to the LWP for a given timer at any point in time. When a timer, for which a signal is still pending expires, (from a previous interval), no signal will be queued, and a “timer overrun count” will be incremented. When a timer expiration signal is delivered to an LWP, **timer_overrun()** may be used to determine the timer expiration overrun count for the specified timer. The overrun count returned contains the number of extra timer expirations which occurred between the time the signal was generated (queued) and when it was delivered, up to but not including a maximum of **{DELAYTIMER_MAX}**. If the number of such extra expirations is greater than or equal to **{DELAYTIMER_MAX}**, then the overrun count is set to **{DELAYTIMER_MAX}**. The value returned by **timer_getoverrun()** applies to the most recent expiration signal delivery for the timer.

RETURN VALUES

timer_settime(), and **timer_gettime()** return **0** upon success. If **timer_getoverrun()** succeeds, the number of extra timer expirations which occurred between the time the signal was queued and when it was delivered is returned. If these functions fail, they return **-1** and set **errno** to indicate the error condition.

ERRORS

EINVAL *timerid* does not correspond to a timer returned by **timer_create(3R)**.
The timer, *timerid*, had already been deleted by **timer_delete(3R)**.
A *value* structure specified a nanosecond value less than zero or greater than or equal to 1,000,000,000.

ENOSYS **timer_settime()**, **timer_gettime()**, or **timer_getoverrun()** is not supported by this implementation.

SEE ALSO

clock_gettime(3R), **timer_create(3R)**, **timer_delete(3R)**

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(tmsp) register struct tms *tmsp; </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 CLK_TCK, found in the header <limits.h>.</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	time(1) , time(2) , wait(2) , getrusage(3B)
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>times() has been superseded by getrusage(3B).</p>

NAME	tmpfile – create a temporary file
SYNOPSIS	#include <stdio.h> FILE *tmpfile(void);
MT-LEVEL	Safe
DESCRIPTION	tmpfile() creates a temporary file using a name generated by the tmpnam() routine and returns a corresponding FILE pointer. If the file cannot be opened, a NULL pointer is returned. The file is automatically deleted when the process using it terminates or when the file is closed. The file is opened for update (“w+”).
SEE ALSO	creat(2), open(2), unlink(2), fopen(3S), mktemp(3C), perror(3C), stdio(3S), tmpnam(3S)

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>
MT-LEVEL	See the NOTES section of this page.
DESCRIPTION	<p>These functions generate file names that can safely be used for a temporary file.</p> <p>tmpnam() always generates a file name using the path-prefix defined as P_tmpdir in the <stdio.h> header. If <i>s</i> is NULL, 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 <i>s</i> is not NULL, it is assumed to be the address of an array of at least L_tmpnam bytes, where L_tmpnam is a constant defined in <stdio.h>; tmpnam() places its result in that array and returns <i>s</i>.</p> <p>tmpnam_r() has the same functionality as tmpnam() except that if <i>s</i> is a NULL pointer, the function returns NULL.</p> <p>tmpnam() allows the user to control the choice of a directory. The argument <i>dir</i> points to the name of the directory in which the file is to be created. If <i>dir</i> is NULL or points to a string that is not a name for an appropriate directory, the path-prefix defined as P_tmpdir in the <stdio.h> header is used. If that directory is not accessible, /tmp will be used as a last resort. This entire sequence can be up-staged by providing an environment variable TMPDIR in the user's environment, whose value is the name of the desired temporary-file directory.</p> <p>Many applications prefer their temporary files to have certain favorite initial letter sequences in their names. Use the <i>pfx</i> argument for this. This argument may be NULL or point to a string of up to five characters to be used as the first few characters of the temporary-file name.</p> <p>tmpnam() uses malloc(3C) to get space for the constructed file name, and returns a pointer to this area. Thus, any pointer value returned from tmpnam() may serve as an argument to free(3C) (see malloc(3C)). If tmpnam() cannot return the expected result for any reason—for example, malloc(3C) failed—or none of the above mentioned attempts to find an appropriate directory was successful, a NULL pointer will be returned.</p> <p>tmpnam() fails if there is not enough space.</p>
SEE ALSO	creat(2) , unlink(2) , fopen(3S) , free(3C) , malloc(3C) , mktemp(3C) , tmpfile(3S)

NOTES

The **tmpnam_r()** 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.

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.

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.

tempnam() is safe in mutli-thread applications. **tmpnam()** is unsafe in multi-thread applications, **tmpnam_r()** should be used instead.

On Solaris systems, the default value for **P_tmpdir** is **var/tmp**.

NAME	trig, sin, cos, tan, asin, acos, atan, atan2 – trigonometric functions
SYNOPSIS	<pre>cc [flag ...] file ... -lm [library ...] #include <math.h> double sin(double x); double cos(double x); double tan(double x); double asin(double x); double acos(double x); double atan(double x); double atan2(double y, double x);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>sin(x), cos(x) and tan(x) return trigonometric functions of radian arguments. Trigonometric argument reduction is carried out with respect to the infinitely precise π.</p> <p>asin(x) returns the arc sine of x in the range $-\pi/2$ to $\pi/2$.</p> <p>acos(x) returns the arc cosine of x in the range 0 to π.</p> <p>atan(x) returns the arc tangent of x in the range $-\pi/2$ to $\pi/2$.</p> <p>atan2(y,x) and hypot(x,y) (see hypot(3M)) convert rectangular coordinates (x,y) to polar (r,θ); atan2(y,x) computes θ, the argument or phase, by computing an arc tangent of y/x in the range $-\pi$ to π.</p>
RETURN VALUES	For exceptional cases, matherr(3M) tabulates the values to be returned as dictated by various Standards.
SEE ALSO	hypot(3M) , matherr(3M)
DIAGNOSTICS	<p>In IEEE 754 mode (i.e. the -xlibmieee cc compilation option), these functions handle exceptional arguments in the spirit of ANSI/IEEE Std 754-1985. sin($\pm\infty$), cos($\pm\infty$) and tan($\pm\infty$) return NaN; asin(x) and acos(x) return NaN if $x > 1$. atan($\pm\infty$) returns $\pm\pi/2$.</p> <p>For atan2(),</p> <ul style="list-style-type: none"> • atan2($\pm 0, x$) returns ± 0 for $x > 0$ or $x = +0$; • atan2($\pm 0, x$) returns $\pm\pi$ for $x < 0$ or $x = -0$; • atan2($y, \pm 0$) returns $\pi/2$ for $y > 0$; • atan2($y, \pm 0$) returns $-\pi/2$ for $y < 0$; • atan2($\pm y, \infty$) returns ± 0 for finite $y > 0$; • atan2($\pm\infty, x$) returns $\pm\pi/2$ for finite x; • atan2($\pm y, -\infty$) returns $\pm\pi$ for finite $y > 0$; • atan2($\pm\infty, \infty$) returns $\pm\pi/4$; • atan2($\pm\infty, -\infty$) returns $\pm 3\pi/4$.

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>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>The file whose name is given by <i>path</i> or referenced by the descriptor <i>fildes</i> has its size set to <i>length</i> bytes.</p> <p>If the file was previously longer than <i>length</i>, bytes past <i>length</i> will no longer be accessible. If it was shorter, bytes from the EOF before the call to the EOF after the call will be read in as zeros. The effective user ID of the process must have write permission for the file, and for ftruncate() the file must be open for writing.</p>
RETURN VALUES	Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and errno is set to indicate the error.
ERRORS	<p>truncate() fails if one or more of the following are true:</p> <p>EACCES Search permission is denied on a component of the path prefix.</p> <p>EACCES Write permission is denied for the file referred to by <i>path</i>.</p> <p>EFAULT <i>path</i> points outside the process's allocated address space.</p> <p>EINTR A signal was caught during execution of the truncate routine.</p> <p>EINVAL <i>path</i> is not an ordinary file.</p> <p>EIO An I/O error occurred while reading from or writing to the file system.</p> <p>EISDIR The file referred to by <i>path</i> is a directory.</p> <p>ELOOP Too many symbolic links were encountered in translating <i>path</i>.</p> <p>EMFILE The maximum number of file descriptors available to the process has been reached.</p> <p>EMULTIHOP Components of <i>path</i> require hopping to multiple remote machines and file system type does not allow it.</p> <p>ENAMETOOLONG The length of a <i>path</i> component exceeds {NAME_MAX} characters, or the length of <i>path</i> exceeds {PATH_MAX} characters.</p> <p>ENFILE Could not allocate any more space for the system file table.</p> <p>ENOENT Either a component of the path prefix or the file referred to by <i>path</i> does not exist.</p> <p>ENOLINK <i>path</i> points to a remote machine and the link to that machine is no longer active.</p> <p>ENOTDIR A component of the path prefix of <i>path</i> is not a directory.</p>

EROFS The file referred to by *path* resides on a read-only file system.

ftruncate() fails if one or more of the following are true:

EAGAIN The file exists, mandatory file/record locking is set, and there are outstanding record locks on the file (see **chmod(2)**).

EBADF *fdes* is not a file descriptor open for writing.

EINTR A signal was caught during execution of the **ftruncate** routine.

EIO An I/O error occurred while reading from or writing to the file system.

ENOLINK *fdes* points to a remote machine and the link to that machine is no longer active.

EINVAL *fdes* does not correspond to an ordinary file.

SEE ALSO **chmod(2)**, **fcntl(2)**, **open(2)**

NAME	tsearch, tfind, tdelete, twalk – manage binary search trees
SYNOPSIS	<pre>#include <search.h> void *tsearch(const void *key, void **rootp, int (*compar)(const void *, const void *)); void *tfind(const void *key, void * const *rootp, int (*compar)(const void *, const void *)); void *tdelete(const void *key, void **rootp, int (*compar)(const void *, const void *)); void twalk(void *root, void(*action) (void *, VISIT, int));</pre>
MT-LEVEL	Safe
DESCRIPTION	<p>tsearch(), tfind(), tdelete(), and twalk() 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>tsearch() is used to build and access the tree. <i>key</i> 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. <i>rootp</i> 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>tdelete() 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>twalk() traverses a binary search tree. <i>root</i> 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 <code>typedef enum { preorder, postorder, endorder, leaf } VISIT;</code> (defined in the <code><search.h></code> header), 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</p>

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

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 **tfind()** and **tdelete()** if *rootp* is NULL on entry. If the datum is found, both **tsearch()** and **tfind()** return a pointer to it. If not, **tfind()** returns NULL, and **tsearch()** returns a pointer to the inserted item.

EXAMPLES

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\n",
            (*(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);
}
```

SEE ALSO [bsearch\(3C\)](#), [hsearch\(3C\)](#), [lsearch\(3C\)](#)

NOTES 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.

NAME	<code>ttyname</code> , <code>ttyname_r</code> , <code>isatty</code> – find name of a terminal
SYNOPSIS	<pre>#include <stdlib.h> char *ttyname(int <i>fildes</i>); char *ttyname_r(int <i>fildes</i>, char *<i>buf</i>, int <i>len</i>); int isatty(int <i>fildes</i>);</pre>
MT-LEVEL	See the NOTES section of this page.
DESCRIPTION	<p><code>ttyname()</code> returns a pointer to a string containing the null-terminated path name of the terminal device associated with file descriptor <i>fildes</i>.</p> <p><code>ttyname_r()</code> has the equivalent functionality to <code>ttyname()</code> except that a buffer <i>buf</i> with length <i>len</i> must be supplied by the caller to store the result. <i>buf</i> must be at least <code>_POSIX_PATH_MAX</code> in size (defined in <code><limits.h></code>).</p> <p><code>isatty()</code> returns 1 if <i>fildes</i> is associated with a terminal device, 0 otherwise.</p>
RETURN VALUES	<code>ttyname()</code> and <code>ttyname_r()</code> return a NULL pointer if <i>fildes</i> does not describe a terminal device in directory <i>/dev</i> .
ERRORS	<p><code>ttyname_r()</code> will fail if the following is true:</p> <p>ERANGE The size of the buffer is smaller than the result to be returned.</p>
FILES	<i>/dev/*</i>
SEE ALSO	<code>gettext(3I)</code> , <code>setlocale(3C)</code>
NOTES	<p>The <code>ttyname_r()</code> 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> <p>When compiling multi-thread applications, the <code>_REENTRANT</code> flag must be defined on the compile line. This flag should only be used in multi-thread applications.</p> <p>If the application is linked with <code>-lintl</code>, then messages printed from this function are in the native language specified by the <code>LC_MESSAGES</code> locale category; see <code>setlocale(3C)</code>.</p> <p>The return value points to static data whose content is overwritten by each call.</p> <p><code>ttyname()</code> is unsafe in multi-thread applications. <code>ttyname_r()</code> is MT-Safe, and should be used instead. <code>isatty()</code> is MT-Safe in multi-thread applications.</p>

NAME	ttyslot – find the slot in the utmp file of the current user
SYNOPSIS	#include <stdlib.h> int ttyslot(void);
MT-LEVEL	Safe
DESCRIPTION	ttyslot() 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	A value of -1 is returned 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.
FILES	/var/adm/utmp
SEE ALSO	getutent(3C) , ttyname(3C)

NAME	ualarm – schedule signal after interval in microseconds
SYNOPSIS	<i>/usr/ucb/cc [flag ...] file ...</i> unsigned ualarm(value, interval) register unsigned value, interval;
DESCRIPTION	ualarm() sends signal SIGALRM (see signal(3B)), to the caller in a number of microseconds given by the <i>value</i> argument. Unless caught or ignored, the signal terminates the caller. If the <i>interval</i> argument is non-zero, the SIGALRM signal will be sent to the caller every <i>interval</i> microseconds after the timer expires (for instance, after <i>value</i> microseconds have passed). Because of scheduling delays, resumption of execution when the signal is caught may be delayed an arbitrary amount. The longest specifiable delay time is $2^{31}-1$ microseconds. The return value is the amount of time previously remaining in the alarm clock.
SEE ALSO	alarm(2) , getitimer(2) , setitimer(2) , sigblock(3B) , sigvec(3B) , signal(3C) , sleep(3C) , usleep(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. ualarm() is a simplified interface to setitimer(2) ; (see getitimer(2)).

NAME	ungetc – push character back onto input stream
SYNOPSIS	#include <stdio.h> int ungetc(int c, FILE *stream);
MT-LEVEL	MT-Safe
DESCRIPTION	<p>ungetc() inserts the character specified by <i>c</i> (converted to an unsigned char) into the buffer associated with an input stream (see intro(3)). That character, <i>c</i>, will be returned by the next getc(3S) call on that stream. ungetc() returns <i>c</i>, and leaves the file corresponding to <i>stream</i> unchanged. A successful call to ungetc() clears the EOF indicator for <i>stream</i>.</p> <p>Four bytes of pushback are guaranteed.</p> <p>The value of the file position indicator for <i>stream</i> after reading or discarding all pushed-back characters will be the same as it was before the characters were pushed back.</p> <p>If <i>c</i> equals EOF, ungetc() does nothing to the buffer and returns EOF.</p> <p>fseek(), rewind() (both described on fseek(3S)), and fsetpos(3C) erase the memory of inserted characters for the stream on which they are applied.</p>
RETURN VALUES	ungetc() returns EOF if it cannot insert the character.
SEE ALSO	intro(3) , fseek(3S) , fsetpos(3C) , getc(3S) , setbuf(3S) , stdio(3S)

NAME	ungetwc – push a Process Code character back into input stream
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lw [<i>library</i> ...] #include <stdio.h> #include <wdec.h> int ungetwc(int c, FILE *stream);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>ungetwc() pushes back the Process Code character <i>c</i> onto an input stream. That character will be returned by the next getwc(3I) call on that stream. ungetwc() returns <i>c</i>, and leaves the file <i>stream</i> unchanged.</p> <p>One character of pushback is guaranteed provided something has been read from the stream and the stream is actually buffered. In the case that <i>stream</i> is stdin, one character may be pushed back onto the buffer without a previous read statement.</p> <p>If <i>c</i> equals EOF, ungetwc() does nothing to the buffer and returns EOF.</p> <p>An fseek(3S) erases all memory of pushed back characters.</p>
RETURN VALUES	ungetwc() returns EOF if it can't push a character back.
SEE ALSO	fseek(3S) , getwc(3I) , setbuf(3S) , ungetc(3S)

NAME	unlockpt – unlock a pseudo-terminal master/slave pair
SYNOPSIS	int unlockpt(int <i>fildev</i>);
MT-LEVEL	Safe
DESCRIPTION	The function unlockpt() clears a lock flag associated with the slave pseudo-terminal device associated with its master pseudo-terminal counterpart so that the slave pseudo-terminal device can be opened. <i>fildev</i> is a file descriptor returned from a successful open of a master pseudo-terminal device.
RETURN VALUES	Upon successful completion, the function unlockpt() returns 0 ; otherwise it returns -1 . A failure may occur if <i>fildev</i> is not an open file descriptor or is not associated with a master pseudo-terminal device.
SEE ALSO	open(2), grantpt(3C), ptsname(3C) <i>STREAMS Programmer's Guide</i>

NAME	usleep – suspend execution for interval in microseconds
SYNOPSIS	<i>/usr/ucb/cc [flag ...] file ...</i> int usleep (<i>useconds</i>) unsigned <i>useconds</i> ;
DESCRIPTION	<p>Suspend the caller for the number of microseconds specified by the argument. The actual suspension time may be an arbitrary amount longer because of other activity in the system, or because of the time spent in processing the call.</p> <p>The routine is implemented by setting an interval timer and pausing until it occurs. The previous state of this timer is saved and restored. If the sleep time exceeds the time to the expiration of the previous timer, the process sleeps only until the signal would have occurred, and the signal is sent a short time later.</p>
SEE ALSO	alarm (2), getitimer (2), sigblock (3B), sleep (3C), ualarm (3B)
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>A microsecond is .000001 seconds.</p>

NAME	vlfmt – display error message in standard format and pass to logging and monitoring services
MT-LEVEL	MT-safe
SYNOPSIS	<pre>#include <stdarg.h> #include <pfmt.h> int vlfmt(FILE *stream, long flags, char *format, va_list ap);</pre>
DESCRIPTION	<p>vlfmt() is the same as lfmt() except that instead of being called with a variable number of arguments, it is called with an argument list as defined by the <stdarg.h> header file. The <stdarg.h> header file defines the type va_list and a set of macros for advancing through a list of arguments whose number and types may vary. The argument ap to vlfmt() is of type va_list. This argument is used with the <stdarg.h> header file macros va_start(), va_arg() and va_end(). [see va_start(), va_arg(), and va_end() in stdarg(5)]. The EXAMPLE section below shows their use with vlfmt().</p> <p>The macro va_alist is used as the parameter list in a function definition as in the function called errlog() in the example below. The macro va_start(ap,), where ap is of type va_list, must be called before any attempt to traverse and access unnamed arguments. Calls to va_arg(ap, atype) 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 ap, which is the same object initialized by va_start. The argument atype is the type that the returned argument is expected to be. The va_end(ap) macro must be invoked when all desired arguments have been accessed. (The argument list in ap 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, arg1, arg2, ..., are given to vlfmt() in the argument ap.</p>
RETURN VALUE	<p>Upon success, vlfmt() returns the number of bytes transmitted. Upon failure, it returns a negative value:</p> <ul style="list-style-type: none"> -1 write error to <i>stream</i>. -2 cannot log and/or display at console.
EXAMPLE	<p>The following demonstrates how vlfmt() could be used to write an errlog() routine:</p> <pre>#include <pfmt.h> #include <stdarg.h> /* * errlog should be called like * errlog(log_info, format, arg1, ...); */ void errlog(long log_info, ...)</pre>

```
{
    va_list ap;
    char *format;

    va_start(ap, );
    format = va_arg(ap, char *);
    (void) vlfmt(stderr, log_info | MM_ERROR, format, ap);
    va_end(ap);
    (void) abort();
}
```

NOTES Since **vlfmt()** uses **gettxt(3C)**, it is recommended that **vlfmt()** not be used.

SEE ALSO **gettxt(3C)**, **lfmt(3C)**, **stdarg(5)**

NAME	vpfmt – display error message in standard format and pass to logging and monitoring services
MT-LEVEL	MT-safe
SYNOPSIS	<pre>#include <stdarg.h> #include <pfmt.h> int vpfmt(FILE *stream, long flags, char *format, va_list ap);</pre>
DESCRIPTION	<p>vpfmt() is the same as lfmt() except that instead of being called with a variable number of arguments, it is called with an argument list as defined by the <stdarg.h> header file. The <stdarg.h> header file defines the type va_list and a set of macros for advancing through a list of arguments whose number and types may vary. The argument ap to vpfmt() is of type va_list. This argument is used with the <stdarg.h> header file macros va_start(), va_arg() and va_end(). [see va_start(), va_arg(), and va_end() in stdarg(5)]. The EXAMPLE section below shows their use with vpfmt().</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 va_start(ap,), where ap is of type va_list, must be called before any attempt to traverse and access unnamed arguments. Calls to va_arg(ap, atype) 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 ap, which is the same object initialized by va_start. The argument atype is the type that the returned argument is expected to be. The va_end(ap) macro must be invoked when all desired arguments have been accessed. (The argument list in ap 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, arg1, arg2, ..., are given to vpfmt() in the argument ap.</p>
RETURN VALUE	<p>Upon success, lfmt() returns the number of bytes transmitted. Upon failure, it returns a negative value:</p> <p>-1 write error to <i>stream</i>.</p>
EXAMPLE	<p>The following demonstrates how vpfmt() could be used to write an error() routine:</p> <pre>#include <pfmt.h> #include <stdarg.h> /* * error should be called like * error(format, arg1, ...); */ void error(...) {</pre>

```
va_list ap;  
char *format;  
  
va_start(ap, );  
format = va_arg(ap, char *);  
(void) vpfmt(stderr, MM_ERROR, format, ap);  
va_end(ap);  
(void) abort();  
}
```

NOTES Since **vpfmt()** uses **gettext(3C)**, it is recommended that **vpfmt()** not be used.

SEE ALSO **pfmt(3C)**, **stdarg(5)**

NAME	vprintf, vfprintf, vsprintf – 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);</pre>
MT-LEVEL	See the NOTES section of this page.
DESCRIPTION	<p>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 the <stdarg.h> header.</p> <p>The <stdarg.h> header defines the type va_list 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 vprint family of routines is of type va_list. This argument is used with the <stdarg.h> header file macros va_start(), va_arg(), and va_end() (see stdarg(5)). The EXAMPLES section below shows 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 va_start(ap, parmN), where <i>ap</i> is of type va_list, and <i>parmN</i> is the rightmost parameter (just before ...), must be called before any attempt to traverse and access unnamed arguments is made. The va_end(ap) 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 given to vfprintf() in the argument <i>ap</i>.</p>
RETURN VALUES	vprintf() and vfprintf() return the number of characters transmitted, or return -1 if an error was encountered.
EXAMPLES	<p>The following demonstrates how vfprintf() could be used to write an error routine:</p> <pre>#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,); /* print out name of function causing error */</pre>

```
(void) fprintf(stderr, "ERR in %s: ", function_name);  
/* print out remainder of message */  
(void) fprintf(stderr, format, ap);  
va_end(ap);  
(void) abort;  
}
```

SEE ALSO [printf\(3S\)](#), [stdarg\(5\)](#)

NOTES [vprintf\(\)](#) and [fprintf\(\)](#) are Async-Safe in mutli-thread applications. [vsprintf\(\)](#) is MT-Safe in multi-thread applications.

NAME	vsyslog – log message with a varargs argument list
SYNOPSIS	<pre>#include <syslog.h> #include <varargs.h> int vsyslog(int priority, const char *message, va_list ap);</pre>
MT-LEVEL	Safe
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	<p>The following demonstrates how vsyslog() could be used to write an error routine.</p> <pre>#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_list) 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(); }</pre>
SEE ALSO	syslog(3) , varargs(5)

NAME	wait, wait3, wait4, waitpid, WIFSTOPPED, WIFSIGNALED, WIFEXITED – wait for process to terminate or stop
SYNOPSIS	<pre> /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; </pre>
DESCRIPTION	<p>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.</p> <p>If <i>status</i> 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 <i>status</i>. 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:</p> <ul style="list-style-type: none"> • 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 **SIGTIN**, **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 *status* or *rusage* arguments point to an illegal address.

waitpid() may set **errno** to:

ECHILD The process or process group specified by *pid* 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 *statusp* 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(3B)**, **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 wait ***', as in:

```
wait((int wait *)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_retrcode** 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	wconv, towupper, towlower – Process Code character conversion macros
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lw [<i>library</i> ...] #include <wdec.h> #include <wctype.h> int towlower(int <i>c</i>); int towupper(int <i>c</i>);</pre>
MT-LEVEL	MT-Safe with exceptions
CHARACTER CONVERSION MACROS	<p>These macros perform simple case conversions on Latin characters in Process Code, <i>wchar_t</i>, from the primary and supplementary codesets, by table lookup.</p> <p>towupper(<i>c</i>) converts the lower-case Latin character <i>c</i> to its upper-case equivalent. If <i>c</i> is not a lower-case Latin character <i>c</i> is returned.</p> <p>towlower(<i>c</i>) converts the upper-case character <i>c</i> to its lower-case equivalent. If <i>c</i> is not an upper-case Latin character <i>c</i> is returned.</p>
SEE ALSO	cctype(3C) , setlocale(3C) , iswalph(3I) , stdio(3S)
NOTES	towupper and towlower can be used safely in a multi-thread application, as long as setlocale(3C) is not being called to change the locale.

NAME	wscoll, wscoll – wide character string comparison using collating information
SYNOPSIS	<pre>cc [flag ...] file ... -lw [library ...] #include <wchar.h> int wscoll(const wchar_t *ws1, const wchar_t *ws2); int wscoll(const wchar_t *ws1, const wchar_t *ws2);</pre>
MT-LEVEL	MT-Safe
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 errno , but no return value is reserved to indicate an error.
ERRORS	<p>wscoll() and wscoll() 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>
SEE ALSO	setlocale(3C) , wscmp(3I) , wcsxfrm(3I)
NOTES	<p>Because no return value is reserved to indicate an error, an application wishing to check for error situations should set errno to 0, call either wscoll() or wscoll(), then check errno and if it is non-zero, assume an error has occurred.</p> <p>wcsxfrm(3I) and wscmp(3I) should be used for sorting large lists.</p>

NAME	wcsftime – convert date and time to wide character string
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lw [<i>library</i> ...] #include <wchar.h> size_t wcsftime(wchar_t *<i>wcs</i>, size_t <i>maxsize</i>, const char *<i>format</i>, const struct tm *<i>timptr</i>);</pre>
DESCRIPTION	<p>The wcsftime() function places wide-character codes into the array pointed to by <i>wcs</i> as controlled by the string pointed to by <i>format</i>.</p> <p>This function behaves as if the character string generated by the strftime(3C) function is passed to the mbstowcs(3C) function as the character string argument, and mbstowcs() places the result in the wide character string argument of the wcsftime() function, up to a limit of <i>maxsize</i> wide-character codes.</p> <p>If copying takes place between objects that overlap, the behaviour 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>wcftime() uses malloc(3C) and should malloc() fail, errno will be set by malloc().</p>
SEE ALSO	malloc(3C) , mbstowcs(3C) , strftime(3C)

NAME	wcstod, wstod, watof – convert wide character string to double-precision number
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lw [<i>library</i> ...] #include <wchar.h> double wcstod(const wchar_t *nptr, wchar_t **endptr); double wstod(const wchar_t *nptr, wchar_t **endptr); double watof(wchar_t *nptr);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>The wcstod() and wstod() functions convert the initial portion of the wide character string pointed to by <i>nptr</i> to double 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(3I)); 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 LC_NUMERIC). 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> <p>If the subject sequence is empty or does not have the expected form, no conversion is performed; the value of <i>nptr</i> is stored in the object pointed to by <i>endptr</i>, provided that <i>endptr</i> is not a null pointer.</p>

watof(str) is equivalent to **wstod(str, (wchar_t **)NULL)**.

RETURN VALUES

wcstod() and **wstod()** 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

wcstod() and **wstod()** will fail if:

ERANGE The value to be returned would cause overflow or underflow.

wcstod() and **wstod()** may fail if:

EINVAL No conversion could be performed.

SEE ALSO

iswspace(3I), **localeconv(3C)**, **scanf(3S)**, **setlocale(3C)**, **wcstol(3I)**

NOTES

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.

NAME	wcstol, wstol, watol, watoll, watoi – convert wide character string to long integer
SYNOPSIS	<pre>cc [flag ...] file ... -lw [library ...] #include <wchar.h> long int wcstol(const wchar_t *nptr, wchar_t **endptr, int base); 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>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>The wcstol() and wstol() functions convert the initial portion of the wide character string pointed to by <i>nptr</i> to long int 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(3I)), 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 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.</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</p>

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.

watol() is equivalent to **wstol(str, (wchar_t **)NULL, 10)**.

watoll() is the long-long (double long) version of **watol()**.

watoi() 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

wcstol() and **wstol()** will fail if:

EINVAL The value of *base* is not supported.

ERANGE The value to be returned is not representable.

wcstol() and **wstol()** may fail if:

EINVAL No conversion could be performed.

SEE ALSO

iswalph(3I), **iswspace(3I)**, **scanf(3S)**, **wcstod(3I)**,

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 **0**, assume an error has occurred.

Truncation from **long long** to **long** can take place upon assignment or by an explicit cast.

NAME	wcstoul – convert wide character string to unsigned long
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lw [<i>library</i> ...] #include <wchar.h> unsigned long int wcstoul(const wchar_t *nptr, wchar_t **endptr, int base);</pre>
MT-LEVEL	MT-Safe
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 iswspace(3I)); 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. 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 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>

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.

RETURN VALUE

Upon successful completion, **wcstoul()** returns 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, **{ULONG_MAX}** is returned and **errno** is set to **ERANGE**.

ERRORS

wcstoul() will fail if:

EINVAL The value of *base* is not supported.

The value to be returned is not representable.

wcstoul() function may fail if:

EINVAL No conversion could be performed.

SEE ALSO

iswalph(3I), **isspace**(3C), **scanf**(3S), **wcstod**(3I), **wcstol**(3I)

WARNINGS

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.

Unlike **wcstod**(3I) and **wcstol**(3I), **wcstoul()** must always return a non-negative number; so, using the return value of **wcstoul()** for out-of-range numbers with **wcstoul()**.

NAME wcstring, wscat, wscat, wcsncat, wsncat, wscmp, wscmp, wcsncmp, wsncmp, wscpy, wscpy, wcsncpy, wsncpy, wcslen, wslen, wcwidth, wcswidth, wcschr, wschr, wcsrchr, wsrchr, windex, wrindex, wcsprbk, wspbrk, wcsvcs, wcsspn, wssp, wscspn, wscspn, wstok, wstok – wide character string operations

SYNOPSIS cc [*flag* ...] *file* ... -lw [*library* ...]

```
#include <wchar.h>

wchar_t *wscat(wchar_t *ws1, const wchar_t *ws2);
wchar_t *wscat(wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcsncat(wchar_t *ws1, const wchar_t *ws2, size_t n);
wchar_t *wsncat(wchar_t *ws1, const wchar_t *ws2, size_t n);

int wscmp(const wchar_t *ws1, const wchar_t *ws2);
int wscmp(const wchar_t *ws1, const wchar_t *ws2);

int wcsncmp(const wchar_t *ws1, const wchar_t *ws2, size_t n);
int wsncmp(const wchar_t *ws1, const wchar_t *ws2, size_t n);

wchar_t *wscpy(wchar_t *ws1, const wchar_t *ws2);
wchar_t *wscpy(wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcsncpy(wchar_t *ws1, const wchar_t *ws2, size_t n);
wchar_t *wsncpy(wchar_t *ws1, const wchar_t *ws2, size_t n);

size_t wcslen(const wchar_t *ws);
size_t wslen(const wchar_t *ws);

int wcwidth(wint_t wc);

int wcswidth(const wchar_t *pwcs, size_t n);

wchar_t *wcschr(const wchar_t *ws, wint_t wc);
wchar_t *wschr(const wchar_t *ws, wint_t wc);

wchar_t *wcsrchr(const wchar_t *ws, wint_t wc);
wchar_t *wsrchr(const wchar_t *ws, wint_t wc);

wchar_t *windex(const wchar_t *ws, wchar_t wc);
wchar_t *wrindex(const wchar_t *ws, wchar_t wc);

wchar_t *wcsvprk(const wchar_t *ws1, const wchar_t *ws2);
wchar_t *wspbrk(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 wssp(const wchar_t *ws1, const wchar_t *ws2);

size_t wscspn(const wchar_t *ws1, const wchar_t *ws2);
size_t wscspn(const wchar_t *ws1, const wchar_t *ws2);

wchar_t *wstok(wchar_t *ws1, const wchar_t *ws2);
wchar_t *wstok(wchar_t *ws1, const wchar_t *ws2);
```

MT-LEVEL	MT-Safe
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, <i>ws</i> , <i>ws1</i> , and <i>ws2</i> 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 <i>ws2</i> (including the terminating null wide-character code) to the end of the wide character string pointed to by <i>ws1</i> . The initial wide-character code of <i>ws2</i> overwrites the null wide-character code at the end of <i>ws1</i> . If copying takes place between objects that overlap, the behaviour is undefined. Both functions return <i>s1</i> ; no return value is reserved to indicate an error.
wcsncat(), wcsncat()	The wcsncat() and wcscat() functions append not more than <i>n</i> wide-character codes (a null wide-character code and wide character codes that follow it are not appended) from the array pointed to by <i>ws2</i> to the end of the wide character string pointed to by <i>ws1</i> . The initial wide-character code of <i>ws2</i> overwrites the null wide-character code at the end of <i>ws1</i> . A terminating null wide-character code is always appended to the result. Both functions return <i>ws1</i> ; no return value is reserved to indicate an error.
wscmp(), wscmp()	The wscmp() and wscmp() functions compare the wide character string pointed to by <i>ws1</i> to the wide character string pointed to by <i>ws2</i> . 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 <i>ws1</i> is greater than, equal to, or less than the wide character string pointed to by <i>ws2</i> .
wcsncmp(), wcsncmp()	The wcsncmp() and wcscmp() functions compare not more than <i>n</i> wide-character codes (wide-character codes that follow a null wide character code are not compared) from the array pointed to by <i>ws1</i> to the array pointed to by <i>ws2</i> . 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> .
wscopy(), wscopy()	The wscopy() and wscopy() 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 behaviour is undefined. Both functions return <i>ws1</i> ; no return value is reserved to indicate an error.
wcsncpy(), wcsncpy()	The wcsncpy() and wcscpy() 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 behaviour is undefined. If the array pointed to by *ws2* is a wide character string that is shorter than *n* wide-character codes, null wide-character codes are appended to the copy in the array pointed to by *ws1*, until a total *n* wide-character codes are written. Both functions return *ws1*; no return value is reserved to indicate an error.

wcslen() , wslen()	The wcslen() and wslen() 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.
wcwidth()	The wcwidth() function determines the number of column positions required for the wide character <i>wc</i> . The value of <i>wc</i> 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. The function returns 0 if <i>wc</i> is a null wide-character code; the number of column positions to be occupied by the wide-character code <i>wc</i> ; or -1 if <i>wc</i> does not correspond to a printing wide-character code.
wcswidth()	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> . The function returns 0 if <i>pwcs</i> points to a null wide-character code; the number of column positions to be occupied by the wide character string pointed to by <i>pwcs</i> ; or -1 if any wide-character code in the wide character string pointed to by <i>pwcs</i> is not a printing wide-character code.
wcschr() , wschr()	The wcschr() and wschr() 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 wchar_t 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() , wsrchr()	The wcsrchr() and wsrchr() 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 wchar_t 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() , wrindex()	The windex() and wrindex() functions behave the same as wschr() and wsrchr() , 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 string pointed to by <i>ws2</i> . Both functions return <i>ws1</i> ; no return value is reserved to indicate an error.
wcscspnl(), wscspnl()	The wcscspnl() and wscspnl() 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 <i>ws1</i> ; no return value is reserved to indicate an error.
wcstok(), wstok()	<p>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>. 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. 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 wcstok() and wstok() return a null pointer. If such a wide-character code is found, it is the start of the first token.</p> <p>wcstok() and wstok() then search from that point for a wide-character code that <i>is</i> 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 <i>ws1</i>, 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. wcstok() and wstok() save a pointer to the following wide-character code, from which the next search for a token will start.</p> <p>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.</p> <p>The implementation will behave as if no function calls wcstok() or wstok(). 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.</p>
SEE ALSO	malloc(3C) , string(3C) , wstring(3I)

NAME	wcsxfrm, wsxfrm – wide character string transformation
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lw [<i>library</i> ...] #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>
MT-LEVEL	MT-Safe
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 wcscmp(3I) or wscmp(3I) 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 wcscoll(3I) or wscoll(3I) 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 behaviour is undefined.</p>
RETURN VALUES	<p>wcsxfrm() and wsxfrm() 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 (size_t)-1, and set errno to indicate the error.</p>
ERRORS	<p>wcsxfrm() and wsxfrm() 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>
NOTES	<p>The transformation function is such that two transformed wide character strings can be ordered by the wcscmp() or wscmp() functions as appropriate to collating sequence information in the program's locale (category LC_COLLATE).</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 errno to 0, call wcscoll() or wscoll(), then check errno and if it is non-zero, assume an error has occurred.</p>
SEE ALSO	wcscmp(3I) , wcscoll(3I) , wscmp(3I) , wscoll(3I)

NAME	wctype – define character class
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lw [<i>library</i> ...] #include <wchar.h> wctype_t wctype(const char *<i>charclass</i>);</pre>
MT-LEVEL	MT-Safe
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 — "alnum", "alpha", "blank", "cntrl", "digit", "graph", "lower", "print", "punct", "space", "upper", and "xdigit".</p> <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 wctype_t, which can be used as the second argument to subsequent calls of iswctype(3I). wctype() determines values of wctype_t 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>
RETURN VALUES	wctype() 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 wctype_t that can be used in calls to iswctype() .
SEE ALSO	iswctype(3I) , setlocale(3C)

NAME	wsprintf – formatted output conversion
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lw [<i>library</i> ...] #include <stdio.h> #include <wdec.h> int wsprintf(wchar_t *<i>s</i>, const char *<i>format</i>, /* <i>arg</i> */ ...);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>wsprintf() outputs a Process Code string ending with a Process Code (<i>wchar_t</i>) NULL character. It is the user's responsibility to allocate enough space for this <i>wchar_t</i> string. This returns the number of Process Code characters (excluding the NULL terminator) that have been written. The conversion specifications and behavior of wsprintf() are the same as the regular sprintf(3S) function except that the result is a Process Code string for wsprintf(), and on Extended Unix Code (EUC) character string for sprintf(3S).</p>
RETURN VALUES	Upon success, wsprintf() returns the number of characters printed. When an error condition is encountered, a negative value is returned.
SEE ALSO	wscanf(3I) , printf(3S) , scanf(3S) , sprintf(3S)

NAME	wsscanf – formatted input conversion
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lw [<i>library</i> ...] #include <stdio.h> #include <wdec.h> int wsscanf(wchar_t *s, const char *format, /* pointer */ ...);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>wsscanf() reads Process Code characters from the Process Code string <i>s</i>, interprets them according to the <i>format</i>, and stores the results in its arguments. wsscanf() expects, as arguments, a control string <i>format</i>, and a set of <i>pointer</i> arguments indicating where the converted input should be stored. 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 simply ignored.</p> <p>The conversion specifications and behavior of wsscanf() are the same as the regular sscanf(3S) function except that the source is a Process Code string for wsscanf(), and on Extended Unix Code (EUC) character string for sscanf(3S).</p>
RETURN VALUES	wsscanf() returns the number of characters matched. On error wsscanf() returns a negative value.
SEE ALSO	wsprintf(3I), printf(3S), scanf(3S)

NAME	wstring, wscasecmp, wncasecmp, wsdup, wscol – Process Code string operations
SYNOPSIS	<pre>cc [flag ...] file ... -lw [library ...] #include <wdec.h> int wscasecmp(const wchar_t *s1, const wchar_t *s2); int wncasecmp(const wchar_t *s1, const wchar_t *s2, int n); wchar_t *wsdup(const wchar_t *s); int wscol(const wchar_t *s);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	<p>These functions operate on Process Code 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, <i>s</i>, <i>s1</i>, and <i>s2</i> point to Process Code strings terminated by a wchar_t NULL.</p> <p>wscasecmp(), wncasecmp() The wscasecmp() function compares its arguments, ignoring case, and returns an integer greater than, equal to, or less than 0, depending upon whether <i>s1</i> is lexicographically greater than, equal to, or less than <i>s2</i>. wncasecmp() makes the same comparison but compares at most <i>n</i> 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 <i>s</i>. The space for the new string is obtained using malloc(3C). 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 <i>s</i>.</p>
SEE ALSO	malloc(3C) , string(3C) , wcstring(3I)

NAME	xdr – library routines for external data representation																																																																				
MT-LEVEL	Safe																																																																				
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	The following table lists XDR routines and the manual reference pages on which they are described:																																																																				
	<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> <tr><td>xdr_reference</td><td>xdr_complex(3N)</td></tr> <tr><td>xdr_setpos</td><td>xdr_admin(3N)</td></tr> <tr><td>xdr_short</td><td>xdr_simple(3N)</td></tr> <tr><td>xdr_sizeof</td><td>xdr_admin(3N)</td></tr> <tr><td>xdr_string</td><td>xdr_complex(3N)</td></tr> <tr><td>xdr_u_char</td><td>xdr_simple(3N)</td></tr> <tr><td>xdr_u_hyper</td><td>xdr_simple(3N)</td></tr> <tr><td>xdr_u_int</td><td>xdr_simple(3N)</td></tr> <tr><td>xdr_u_long</td><td>xdr_simple(3N)</td></tr> <tr><td>xdr_u_longlong_t</td><td>xdr_simple(3N)</td></tr> <tr><td>xdr_u_short</td><td>xdr_simple(3N)</td></tr> <tr><td>xdr_union</td><td>xdr_complex(3N)</td></tr> <tr><td>xdr_vector</td><td>xdr_complex(3N)</td></tr> <tr><td>xdr_void</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_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 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)

SEE ALSO**rpc(3N), xdr_admin(3N), xdr_complex(3N), xdr_create(3N), xdr_simple(3N)**

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
MT-LEVEL	Safe
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 that malloc(3C) be used to allocate these buffers or that the programmer insure that the buffer address is divisible evenly by four.</p> <p>#include <rpc/xdr.h></p> <p>bool_t xdr_control(XDR *xdrs, int req, void *info);</p> <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>, their argument types and what they do are:</p> <p>XDR_GET_BYTES_AVAIL xdr_bytesrec * return number of bytes left unconsumed in the stream and a flag indicating whether or not this is the last fragment.</p> <p>u_int xdr_getpos(const XDR *xdrs);</p> <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> <p>long *xdr_inline(XDR *xdrs, const int len);</p> <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;</p>

it exists for the sake of efficiency, and applications written for portability should not depend on this feature.

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, u_int 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*. On success this routine returns the number of bytes read, -1 on failure. 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 u_int 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.

SEE ALSO

malloc(3C), rpc(3N), xdr_complex(3N), xdr_create(3N), xdr_simple(3N)

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
MT-LEVEL	Safe
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 stream (see xdr_create(3N)).
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 that malloc() be used to allocate these buffers or that the programmer insure that the buffer address is divisible evenly by four.</p> <p>#include <rpc/xdr.h></p> <p>bool_t xdr_array(XDR *xdrs, caddr_t *arp, u_int *sizep, const u_int maxsize, const u_int elsize, const xdrproc_t elproc);</p> <p>xdr_array() translates between variable-length arrays and their corresponding external representations. The parameter <i>arp</i> is the address of the pointer to the array, while <i>sizep</i> is the address of the element count of the array; this element count cannot exceed <i>maxsize</i>. The parameter <i>elsize</i> is the size of each of the array's elements, and <i>elproc</i> is an XDR routine that translates between the array elements' C form and their external representation. If <i>arp</i> is null when decoding, xdr_array() allocates memory and <i>arp</i> points to it. This routine returns TRUE if it succeeds, FALSE otherwise.</p> <p>bool_t xdr_bytes(XDR *xdrs, char **sp, u_int *sizep, const u_int maxsize);</p> <p>xdr_bytes() translates between counted byte strings and their external representations. The parameter <i>sp</i> is the address of the string pointer. The length of the string is located at address <i>sizep</i>; strings cannot be longer than <i>maxsize</i>. If <i>sp</i> is null when decoding, xdr_bytes() allocates memory and <i>sp</i> points to it. This routine returns TRUE if it succeeds, FALSE otherwise.</p> <p>bool_t xdr_opaque(XDR *xdrs, caddr_t cp, const u_int cnt);</p> <p>xdr_opaque() translates between fixed size opaque data and its external representation. The parameter <i>cp</i> is the address of the opaque object, and <i>cnt</i> is its size in bytes. This routine returns TRUE if it succeeds, FALSE otherwise.</p>

bool_t xdr_pointer(XDR *xdrs, char **objpp, u_int 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, u_int 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** 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 u_int 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 Note: **xdr_string()** can be used to send an empty string (" "), but not a NULL string. **TRUE** if it succeeds, **FALSE** otherwise.

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). Returns **TRUE** if it succeeds, **FALSE** otherwise.

bool_t xdr_vector(XDR *xdrs, char *arrp, const u_int size, const u_int 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.

SEE ALSO **rpc(3N)**, **xdr_admin(3N)**, **xdr_create(3N)**, **xdr_simple(3N)**

NAME	xdr_create, xdr_destroy, xdrmem_create, xdrrec_create, xdrstdio_create – library routines for external data representation stream creation
MT-LEVEL	MT-Safe
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 XDR, CLIENT, and SVCXPRT data structures. Note that any buffers passed to the XDR routines must be properly aligned. It is suggested that malloc(3C) be used to allocate these buffers or that the programmer insure that the buffer address is divisible evenly by four.</p> <p>#include <rpc/xdr.h></p> <p>void xdr_destroy(XDR *xdrs);</p> <p>A macro that invokes the destroy routine associated with the XDR stream, <i>xdrs</i>. Destruction usually involves freeing private data structures associated with the stream. Using <i>xdrs</i> after invoking xdr_destroy() is undefined.</p> <p>void xdrmem_create(XDR *xdrs, const caddr_t addr, const u_int size, const enum xdr_op op);</p> <p>This routine initializes the XDR stream object pointed to by <i>xdrs</i>. The stream's data is written to, or read from, a chunk of memory at location <i>addr</i> whose length is no less than <i>size</i> bytes long. The <i>op</i> determines the direction of the XDR stream (either XDR_ENCODE, XDR_DECODE, or XDR_FREE).</p> <p>void xdrrec_create(XDR *xdrs, const u_int sendsz, const u_int 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));</p> <p>This routine initializes the read-oriented XDR stream object pointed to by <i>xdrs</i>. The stream's data is written to a buffer of size <i>sendsz</i>; a value of 0 indicates the system should use a suitable default. The stream's data is read from a buffer of size <i>recvsz</i>; it too can be set to a suitable default by passing a 0 value. When a stream's output buffer is full, <i>writeit</i> is called. Similarly, when a stream's input buffer is empty, <i>readit</i> is called. The behavior of these two routines is similar to the system calls read() and write() (see read(2) and write(2), respectively), except that an appropriate handle (<i>read_handle</i> or <i>write_handle</i>) is passed to the former routines as the first parameter instead of a file descriptor. Note: the XDR</p>

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*→*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.

SEE ALSO

read(2), **write(2)**, **malloc(3C)**, **rpc(3N)**, **xdr_admin(3N)**, **xdr_complex(3N)**, **xdr_simple(3N)**, **fclose(3S)**

NAME	xdr_simple, xdr_bool, xdr_char, xdr_double, xdr_enum, xdr_float, xdr_free, xdr_hyper, xdr_int, xdr_long, xdr_longlong_t, xdr_quadruple, xdr_short, xdr_u_char, xdr_u_hyper, xdr_u_int, xdr_u_long, xdr_u_longlong_t, xdr_u_short, xdr_void – library routines for external data representation
MT-LEVEL	Safe
DESCRIPTION	<p>XDR library routines allow C programmers to describe simple 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 require the creation of XDR streams (see xdr_create(3N)).</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 that malloc(3C) be used to allocate these buffers or that the programmer insure that the buffer address is divisible evenly by four.</p> <p>#include <rpc/xdr.h></p> <p>bool_t xdr_bool(XDR *xdrs, bool_t *bp);</p> <p>xdr_bool() translates between booleans (C integers) and their external representations. When encoding data, this filter produces values of either 1 or 0. This routine returns TRUE if it succeeds, FALSE otherwise.</p> <p>bool_t xdr_char(XDR *xdrs, char *cp);</p> <p>xdr_char() translates between C characters and their external representations. This routine returns TRUE if it succeeds, FALSE otherwise. Note: encoded characters are not packed, and occupy 4 bytes each. For arrays of characters, it is worthwhile to consider xdr_bytes(), xdr_opaque(), or xdr_string() (see xdr_complex(3N)).</p> <p>bool_t xdr_double(XDR *xdrs, double *dp);</p> <p>xdr_double() translates between C double precision numbers and their external representations. This routine returns TRUE if it succeeds, FALSE otherwise.</p> <p>bool_t xdr_enum(XDR *xdrs, enum_t *ep);</p> <p>xdr_enum() translates between C enums (actually integers) and their external representations. This routine returns TRUE if it succeeds, FALSE otherwise.</p> <p>bool_t xdr_float(XDR *xdrs, float *fp);</p> <p>xdr_float() translates between C floats and their external representations. This routine returns TRUE if it succeeds, FALSE otherwise.</p>

void xdr_free(xdrproc_t proc, char *objp);

Generic freeing routine. The first argument is the XDR routine for the object being freed. The second argument is a pointer to the object itself. Note: the pointer passed to this routine is not freed, but what it points to is freed (recursively, depending on the XDR routine).

bool_t xdr_hyper(XDR *xdrs, longlong_t *llp);

xdr_hyper() translates between ANSI C **long long** integers and their external representations. This routine returns **TRUE** if it succeeds, **FALSE** otherwise.

bool_t xdr_int(XDR *xdrs, int *ip);

xdr_int() translates between C integers and their external representations. This routine returns **TRUE** if it succeeds, **FALSE** otherwise.

bool_t xdr_long(XDR *xdrs, long *lp);

xdr_long() translates between C **long** integers and their external representations. This routine returns **TRUE** if it succeeds, **FALSE** otherwise.

bool_t xdr_longlong_t(XDR *xdrs, longlong_t *llp);

xdr_longlong_t() translates between ANSI C **long long** integers and their external representations. This routine returns **TRUE** if it succeeds, **FALSE** otherwise. This routine is identical to **xdr_hyper()**.

bool_t xdr_quadruple(XDR *xdrs, long double *pq);

xdr_quadruple() translates between IEEE quadruple precision floating point numbers and their external representations. This routine returns **TRUE** if it succeeds, **FALSE** otherwise.

bool_t xdr_short(XDR *xdrs, short *sp);

xdr_short() translates between C **short** integers and their external representations. This routine returns **TRUE** if it succeeds, **FALSE** otherwise.

bool_t xdr_u_char(XDR *xdrs, unsigned char *ucp);

xdr_u_char() translates between **unsigned** C characters and their external representations. This routine returns **TRUE** if it succeeds, **FALSE** otherwise.

bool_t xdr_u_hyper(XDR *xdrs, u_longlong_t *ullp);

xdr_u_hyper() translates between unsigned ANSI C **long long** integers and their external representations. This routine returns **TRUE** if it succeeds, **FALSE** otherwise.

bool_t xdr_u_int(XDR *xdrs, unsigned *up);

A filter primitive that translates between a C **unsigned** integer and its external representation. This routine returns **TRUE** if it succeeds, **FALSE** otherwise.

bool_t xdr_u_long(XDR *xdrs, unsigned long *ulp);

xdr_u_long() translates between C **unsigned long** integers and their external representations. This routine returns **TRUE** if it succeeds, **FALSE** otherwise.

bool_t xdr_u_longlong_t(XDR *xdrs, u_longlong_t *ullp);

xdr_u_longlong_t() translates between unsigned ANSI C **long long** integers and their external representations. This routine returns **TRUE** if it succeeds, **FALSE** otherwise. This routine is identical to **xdr_u_hyper()**.

bool_t xdr_u_short(XDR *xdrs, unsigned short *usp);

xdr_u_short() translates between C **unsigned short** integers and their external representations. This routine returns **TRUE** if it succeeds, **FALSE** otherwise.

bool_t xdr_void(void);

This routine always returns **TRUE**. It may be passed to RPC routines that require a function parameter, where nothing is to be done.

SEE ALSO

malloc(3C), rpc(3N), xdr_admin(3N), xdr_complex(3N), xdr_create(3N)

NAME	yp_update – change NIS information
SYNOPSIS	<pre>#include <rpcsvc/ypclnt.h> int yp_update(char *domain, char *map, unsigned ypop, char *key, int keylen, char *data, int datalen);</pre>
MT-LEVEL	Unsafe
DESCRIPTION	<p>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.</p> <p>This routine depends upon secure RPC, and will not work unless the network is running secure RPC.</p>
RETURN VALUES	<p>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.</p> <p>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.</p>
SEE ALSO	secure_rpc(3N) , ypclnt(3N)
NOTES	This interface is unsafe in multithreaded applications. Unsafe interfaces should be called only from the main thread.

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	cc [<i>flag</i> ...] <i>file</i> ... -lnsl [<i>library</i> ...] #include <rpcsvc/ypclnt.h> #include <rpcsvc/yp_prot.h>
MT-LEVEL	Unsafe
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 nisd(1). 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 ypserv process running on another machine on the network that has an earlier version of SunOS or by an NIS+ server, see rpc.nisd(1M), running in "YP-compatibility mode". Refer to the NOTES section in ypfiles(4) for implications of being an NIS client of an NIS+ server in "YP-compatibility mode", and to ypbind(1M), ypwhich(1), ypmatch(1), and ypcat(1) for commands to access NIS from a client machine. The package can be loaded from the standard library, /usr/lib/libnsl.so.1.</p> <p>All input parameter names begin with <i>in</i>. Output parameters begin with <i>out</i>. Output parameters of type char ** should be addresses of uninitialized character pointers. Memory is allocated by the NIS client package using malloc(3C), 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 (YPERR_xxxx) otherwise. Failure codes are described under ERRORS below.</p>
Routines	<p>yp_bind (char *indomain);</p> <p>To use the NIS name services, the client process must be "bound" to an NIS server that serves the appropriate domain using yp_bind(). Binding need not be done explicitly by user code; this is done automatically whenever an NIS lookup function is called. yp_bind() 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 yp_bind(), it should call yp_unbind() when it is done using NIS in order to free up resources.</p>

void yp_unbind(char *indomain);

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.

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);

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 {
    int (*foreach)();
    char *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	15	<i>/* access violation */</i>
YPERR_BADARGS	1	<i>/* args to function are bad */</i>
YPERR_BADDB	13	<i>/* yp database is bad */</i>
YPERR_BUSY	16	<i>/* database busy */</i>
YPERR_DOMAIN	3	<i>/* can't bind to server on this domain */</i>
YPERR_KEY	5	<i>/* no such key in map */</i>
YPERR_MAP	4	<i>/* no such map in server's domain */</i>
YPERR_NODOM	12	<i>/* local domain name not set */</i>
YPERR_NOMORE	8	<i>/* no more records in map database */</i>
YPERR_PMAP	9	<i>/* can't communicate with rpcbinder */</i>
YPERR_RESRC	7	<i>/* resource allocation failure */</i>
YPERR_RPC	2	<i>/* RPC failure – domain has been unbound */</i>
YPERR_YPBIND	10	<i>/* can't communicate with ypbind */</i>
YPERR_YPERR	6	<i>/* internal yp server or client error */</i>
YPERR_YPSESV	11	<i>/* can't communicate with ypserv */</i>
YPERR_VERS	14	<i>/* yp version mismatch */</i>

FILES

/usr/lib/libnsl.so.1

SEE ALSO

nis+(1), ypcat(1), ypmatch(1), ypwhich(1), rpcbind(1M), rpc.nisd(1M), ypbind(1M), sysinfo(2), malloc(3C), ypfiles(4)

NOTES

This interface is unsafe in multithreaded applications. Unsafe interfaces should be called only from the main thread.

Index

Special Characters

`_longjmp` — non-local goto, 3B-834

`_setjmp` — non-local goto, 3B-834

A

`abort` — terminate the process abnormally, 3C-90

`abs` — return absolute value of integer, 3C-91

`accept` — accept a connection on a socket, 3N-92

accounting

time accounting for current process — `times`, 3B-991

`acos` — trigonometric arccosine, 3M-995

`acosh` — inverse hyperbolic function, 3M-483

acquire and release stream lock — `flockfile`, 3S-325

`funlockfile`, 3S-325

add a string of `wchar_t` characters to a curses window and advance cursor — `curs_addwstr`, 3X-162

`addnwstr`, 3X-162

`addwstr`, 3X-162

`mvaddnwstr`, 3X-162

`mvaddwstr`, 3X-162

`mvwaddnwstr`, 3X-162

`mvwaddwstr`, 3X-162

`waddnwstr`, 3X-162

`waddwstr`, 3X-162

add a `wchar_t` character (with attributes) to a curses

window and advance cursor —

`curs_addwch`, 3X-159

`addwch`, 3X-159

`echowchar`, 3X-159

`mvaddwch`, 3X-159

`mvwaddwch`, 3X-159

`waddwch`, 3X-159

`wechowchar`, 3X-159

add string of `wchar_t` characters (and attributes) to a curses window — `curs_addwchstr`, 3X-161

`addwchnstr`, 3X-161

`addwchstr`, 3X-161

`mvaddwchnstr`, 3X-161

`mvaddwchstr`, 3X-161

`mvwaddwchnstr`, 3X-161

`mvwaddwchstr`, 3X-161

`waddwchnstr`, 3X-161

`waddwchstr`, 3X-161

additional severities

define — `addsev`, 3C-94

`addnwstr` — add a string of `wchar_t` characters to a curses window and advance cursor, 3X-162

`addsev` — define additional severities, 3C-94

`addseverity` — build a list of severity levels for an application for use with `fmtmsg`, 3C-95

`addwch` — add a `wchar_t` character (with attributes) to a curses window and advance cursor, 3X-159

`addwchnstr` — add string of `wchar_t` characters

(and attributes) to a curses window, 3X-161

addwchstr — add string of wchar_t characters
(and attributes) to a curses window, 3X-161

addwstr — add a string of wchar_t characters to a
curses window and advance cursor, 3X-162

adjcurspos — moving the cursor by character,
3X-163

advance — regular expression compile and match
routines, 3G-739

aio_cancel — cancel asynchronous I/O request,
3R-97

aio_error — retrieve error status of asynchronous
I/O operation, 3R-103

aio_fsync — asynchronous file synchronization,
3R-99

aio_read — asynchronous read and write opera-
tions, 3R-101

aio_return — retrieve return status of asynchro-
nous I/O operation , 3R-103

aio_suspend — wait for asynchronous I/O
request, 3R-106

aio_write — asynchronous read and write opera-
tions, 3R-101

aiocancel — cancel an asynchronous operation,
3-108

aioread — initiate asynchronous read, 3-109

aiowait — wait for completion of asynchronous
I/O operation, 3-111

aioread — initiate asynchronous write, 3-109

alarm
 schedule signal after interval in microseconds
 — ualarm, 3B-1003

ALE curses library, See curses library

alphasort — scan a directory, 3B-798

applications
 build a list of severity levels for use with
 fmtmsg — addseverity, 3C-95
 display a message on stderr or system console
 — fmtmsg, 3C-328
 get entries from symbol table — nlist, 3B-674

arithmetic
 compute the quotient and remainder — div,
 3C-250

arithmetic, *continued*

arithmetic, 48-bit integer
 generate uniformly distributed pseudo-random
 numbers — drand48, 3C-261

ascftime — convert date and time to string,
3C-893

asin — trigonometric arcsine, 3M-995

asinh — inverse hyperbolic function, 3M-483

assert — verify program assertion, 3X-112

asynchronous file synchronization
 — aio_sync, 3R-99

asynchronous I/O
 — aio_cancel, 3R-97
 — aiocancel, 3-108
 — aioread, 3-109
 — aiowait, 3-111
 — aiowrite, 3-109
 retrieve error status — aio_error, 3R-103
 retrieve return status — aio_return, 3R-103
 wait for request — aio_suspend, 3R-106

asynchronous read and write operations
 — aio_read, aio_write, 3R-101

asysmem — return physical memory information,
3-919

atan — trigonometric arctangent, 3M-995

atan2 — trigonometric arctangent, 3M-995

atanh — inverse hyperbolic function, 3M-483

atexit — add program termination routine,
3C-113

attroff — curses character and window attribute
control routines, 3X-164

attron — curses character and window attribute
control routines, 3X-164

attrset — curses character and window attribute
control routines, 3X-164

au_close — construct audit records, 3-114

au_open — construct audit records, 3-114

au_preselect — preselect an audit record, 3-115

au_to_arg — creating audit record tokens, 3-117

au_to_attr — creating audit record tokens, 3-117

au_to_data — creating audit record tokens, 3-117

au_to_groups — creating audit record tokens,

3-117

`au_to_in_addr` — creating audit record tokens, 3-117

`au_to_ipc` — creating audit record tokens, 3-117

`au_to_ipc_perm` — creating audit record tokens, 3-117

`au_to_iport` — creating audit record tokens, 3-117

`au_to_me` — creating audit record tokens, 3-117

`au_to_opaque` — creating audit record tokens, 3-117

`au_to_path` — creating audit record tokens, 3-117

`au_to_process` — creating audit record tokens, 3-117

`au_to_return` — creating audit record tokens, 3-117

`au_to_text` — creating audit record tokens, 3-117

`au_user_mask` — get user's binary preselection mask, 3-120

`au_write` — write audit records, 3-114

audit control file information

- `endac`, 3-378
- `getacdir`, 3-378
- `getacflg`, 3-378
- `getacinfo`, 3-378
- `getacmin`, 3-378
- `getacna`, 3-378
- `setac`, 3-378

audit record tokens, creating

- `au_to_attr`, 3-117
- `au_to_data`, 3-117
- `au_to_groups`, 3-117
- `au_to_in_addr`, 3-117
- `au_to_in_ipc`, 3-117
- `au_to_in_ipc_perm`, 3-117
- `au_to_iport`, 3-117
- `au_to_me`, 3-117
- `au_to_opaque`, 3-117
- `au_to_path`, 3-117
- `au_to_process`, 3-117
- `au_to_return`, 3-117
- `au_to_socket`, 3-117
- `au_to_subject`, 3-117

audit record tokens, creating, *continued*

- `au_to_text`, 3-117

audit record tokens, manipulating

- `au_close`, 3-114
- `au_open`, 3-114
- `au_preselect`, 3-115
- `au_write`, 3-114

`auth_destroy` — library routines for client side remote procedure call authentication, 3N-754

`authnone_create` — library routines for client side remote procedure call authentication, 3N-754

`authsys_create` — library routines for client side remote procedure call authentication, 3N-754

`authsys_create_default` — library routines for client side remote procedure call authentication, 3N-754

B

base-64 ASCII characters

- convert from long integer — `l64a`, 3C-89
- convert to long integer — `a64l`, 3C-89

`basename` — return the last element of path name, 3G-121

Basic Security Module functions

- `au_close`, 3-114
- `au_open`, 3-114
- `au_preselect`, 3-115
- `au_to_attr`, 3-117
- `au_to_data`, 3-117
- `au_to_groups`, 3-117
- `au_to_in_addr`, 3-117
- `au_to_ipc`, 3-117
- `au_to_ipc_perm`, 3-117
- `au_to_iport`, 3-117
- `au_to_me`, 3-117
- `au_to_opaque`, 3-117
- `au_to_path`, 3-117
- `au_to_process`, 3-117
- `au_to_return`, 3-117
- `au_to_socket`, 3-117
- `au_to_subject`, 3-117
- `au_to_text`, 3-117
- `au_user_mask`, 3-120

Basic Security Module functions, *continued*

- `au_write`, 3-114

`bcmp` — operates on variable length strings of bytes, 3B-130

`bcopy` — operates on variable length strings of bytes, 3B-130

Bessel functions

- `j0`, 3M-122
- `j1`, 3M-122
- `jn`, 3M-122
- `y0`, 3M-122
- `y1`, 3M-122
- `yn`, 3M-122

`bgets` — read stream up to next delimiter, 3G-123

binary search of sorted table

- `bsearch`, 3C-128

binary search trees, manage

- `tdelete`, 3C-998
- `tfind`, 3C-998
- `tsearch`, 3C-998
- `twalk`, 3C-998

`bind` — bind a name to a socket, 3N-124

bind an address to a transport endpoint — `t_bind`, 3N-925

`bindtextdomain` — select location of domain, 3I-460

bit and byte operations

- find first set bit — `ffs`, 3C-320

`bsdmalloc` — memory allocator, 3X-126

`bsearch` — binary search a sorted table, 3C-128

BSM, See Basic Security Module

`bstring` — bit and byte string operations, 3B-130

buffer

- split into fields — `bufsplit`, 3G-131

buffering, assign to stream

- `setbuffer`, 3B-832
- `setlinebuf`, 3B-832

byte order, convert values between host and network

- `byteorder`, 3N-132
- `htonl`, 3N-132
- `htons`, 3N-132
- `ntohl`, 3N-132

byte order, convert values between host and network, *continued*

- `ntohs`, 3N-132

`bzero` — operates on variable length strings of bytes, 3B-130

C

C Compilation

- close a shared object — `dlclose`, 3X-253
- get address of symbol in shared object — `dlsym`, 3X-257
- get diagnostic information — `dlerror`, 3X-254
- open a shared object — `dlopen`, 3X-255

`catclose` — close a message catalog, 3C-134

`catgets` — read a program message, 3C-133

`catopen` — open a message catalog, 3C-134

`cbirt` — cube root function, 3M-877

`ceil` — round to integral value, 3M-327

`cfgetispeed` — general terminal interface, 3-967, 3-968

`cfgetospeed` — general terminal interface, 3-967, 3-968

`cfsetispeed` — general terminal interface, 3-967, 3-968

`cfsetospeed` — general terminal interface, 3-967, 3-968

`cftime` — convert date and time to string, 3C-893

character based forms package

- `forms`, 3X-363

character based menus package

- `menus`, 3X-598

character based panels package

- `panels`, 3X-691

character classification and conversion

- `_tolower`, 3C-144
- `_toupper`, 3C-144
- `conv`, 3C-144
- `toascii`, 3C-144
- `tolower`, 3C-144
- `toupper`, 3C-144

character handling

- `ctype`, 3C-153
- `isalnum`, 3C-153

character handling, *continued*

- isalpha, 3C-153
- isascii, 3C-153
- iscntrl, 3C-153
- isdigit, 3C-153
- isgraph, 3C-153
- islower, 3C-153
- isprint, 3C-153
- ispunct, 3C-153
- isspace, 3C-153
- isupper, 3C-153
- isxdigit, 3C-153

client side remote procedure call authentication,
library routines for

- auth_destroy, 3N-754
- authnone_create, 3N-754
- authsys_create, 3N-754
- authsys_create_default, 3N-754
- rpc_clnt_auth, 3N-754

clnt_call — library routines for client side calls,
3N-756

clnt_control — library routines for dealing with
creation and manipulation of CLIENT handles,
3N-759

clnt_create — library routines for dealing with
creation and manipulation of CLIENT handles,
3N-759

clnt_create_vers — library routines for dealing
with creation and manipulation of CLIENT
handles, 3N-759

clnt_destroy — library routines for dealing with
creation and manipulation of CLIENT handles,
3N-759

clnt_dg_create — library routines for dealing
with creation and manipulation of CLIENT
handles, 3N-759

clnt_freeres — library routines for client side
calls, 3N-756

clnt_geterr — library routines for client side
calls, 3N-756

clnt_pcreateerror — library routines for deal-
ing with creation and manipulation of CLIENT
handles, 3N-759

clnt_perrno — library routines for client side
calls, 3N-756

clnt_perror — library routines for client side
calls, 3N-756

clnt_raw_create — library routines for dealing
with creation and manipulation of CLIENT
handles, 3N-759

clnt_spcreateerror — library routines for deal-
ing with creation and manipulation of CLIENT
handles, 3N-759

clnt_sperrno — library routines for client side
calls, 3N-756

clnt_sperror — library routines for client side
calls, 3N-756

clnt_tli_create — library routines for dealing
with creation and manipulation of CLIENT
handles, 3N-759

clnt_tp_create — library routines for dealing
with creation and manipulation of CLIENT
handles, 3N-759

clnt_vc_create — library routines for dealing
with creation and manipulation of CLIENT
handles, 3N-759

clock — report CPU time used, 3C-136

clock_getres — high-resolution clock operations,
3R-137

clock_gettime — high-resolution clock opera-
tions, 3R-137

clock_settime — high-resolution clock opera-
tions, 3R-137

closelog — close system log file, 3-916

code conversion allocation function —

iconv_open, 3-488

code conversion deallocation function —

iconv_close, 3-487

code conversion function — iconv, 3-485

command options

get option letter from argument vector —

getopt, 3C-421

command suboptions

parse suboptions from a string — getsubopt,
3C-457

commands

- open, close to and from a command —
p2open, p2close, 3G-681
- return stream to remote — rcmd, 3N-729

communications

- accept a connect request — t_accept, 3N-921
- accept a connection on a socket — accept, 3N-92
- acknowledge receipt of an orderly release indication — t_rcvrel, 3N-950
- allocate memory for argument structures — t_alloc, 3N-923
- bind a name to a socket — bind, 3N-124
- bind an address to a transport endpoint — t_bind, 3N-925
- close a transport endpoint — t_close, 3N-928
- create a pair of connected sockets — socket-pair, 3N-874
- create an endpoint for communication — socket, 3N-871
- disable a transport endpoint — t_unbind, 3N-965
- establish a connection with another transport user — t_connect, 3N-929
- establish a transport endpoint — t_open, 3N-941
- free allocated memory — t_free, 3N-933
- get name of peer connected to socket — get-peername, 3N-426
- get protocol-specific service information — t_getinfo, 3N-934
- get socket name — getsockname, 3N-449
- get the current state — t_getstate, 3N-937
- initiate a connection on a socket — connect, 3N-142
- initiate an orderly release of connection — t_sndrel, 3N-959
- listen for a connect request — t_listen, 3N-938
- listen for connections on a socket — listen, 3N-543
- look at the current event on a transport endpoint — t_look, 3N-940
- manage options for a transport endpoint — t_optmgmt, 3N-942

communications, *continued*

- produce error message — t_error, 3N-932
 - receive a data unit — t_rcvudata, 3N-951
 - receive a unit data error indication — t_rcvuderr, 3N-953
 - receive data or expedited data sent over a connection — t_rcv, 3N-944
 - receive the confirmation from a connect request — t_rcvconnect, 3N-946
 - retrieve information from disconnect — t_rcvdis, 3N-948
 - scatter data in order to test the network — spray, 3N-875
 - send a data unit — t_sndudata, 3N-960
 - send a message from a socket — send, sendto, sendmsg, 3N-829
 - send data or expedited data over a connection — t_snd, 3N-955
 - send user-initiated disconnect request — t_snddis, 3N-957
 - shut down part of a full-duplex connection — shutdown, 3N-849
 - synchronize transport library — t_sync, 3N-963
- compile — regular expression compile and match routines, 3G-739
- cond_broadcast() — signal a condition variable, 3T-139
- cond_destroy() — destroy a condition variable, 3T-139
- cond_init() — initialize a condition variable, 3T-139
- cond_signal() — signal a condition variable, 3T-139
- cond_wait() — wait for a condition variable, 3T-139
- cond_wait() — wait for a condition variable, 3T-139
- configuration script
- execute — doconfig, 3N-259
- connect — initiate a connection on socket, 3N-142
- control system log
- close system log — closelog, 3-916
 - set log priority mask — setlogmask, 3-916

control system log, *continued*
 start system log — `openlog`, 3-916
 write to system log — `syslog`, 3-916

convert date and time to string — `strftime`, 3C-893
`asctime`, 3C-893
`cftime`, 3C-893

convert date and time to wide character string — `wcsftime`, 3I-1021

convert floating-point number to string
 — `ecvt`, 3C-266
 — `fcvt`, 3C-266
 — `gcvt`, 3C-266

convert formatted input — `scanf`, 3S-799
`fscanf`, 3S-799
`sscanf`, 3S-799

convert monetary value to string — `strfmon`, 3C-889

convert numbers to strings
 — `econvert`, 3-264
 — `ecvt`, 3-264
 — `fconvert`, 3-264
 — `fcvt`, 3-264
 — `fprintf`, 3B-702
 — `gconvert`, 3-264
 — `gcvt`, 3-264
 — `printf`, 3B-702
 — `qeconvert`, 3-264
 — `qfconvert`, 3-264
 — `qgconvert`, 3-264
 — `seconvert`, 3-264
 — `sfconvert`, 3-264
 — `sgconvert`, 3-264
 — `sprintf`, 3B-702
 — `vfprintf`, 3B-702
 — `vprintf`, 3B-702
 — `vsprintf`, 3B-702

convert to `wchar_t` strings
`wsprintf`, 3I-1034

convert wide character string to double-precision number — `wcstod`, 3I-1022
`watof`, 3I-1022
`wstod`, 3I-1022

convert wide character string to unsigned long — `wcstoul`, 3I-1026

`copysign()` function, 3M-489

`cos` — trigonometric cosine, 3M-995

`cosh` — hyperbolic cosine, 3M-483

CPU time
 report for calling process — `clock`, 3C-136

CPU-use
 prepare execution profile — `monitor`, 3C-609

CRT handling and optimization package
 — `curses`, 3X-228

`cset` — get information on EUC codesets, 3I-147

`csetcol` — get information on EUC codesets, 3I-147

`csetlen` — get information on EUC codesets, 3I-147

`csetno` — get information on EUC codesets, 3I-147

`ctermid` — generate path name for controlling terminal, 3S-148

`ctermid_r` — generate path name for controlling terminal, 3S-148

cube root — `cbirt`, 3M-877

current working directory
 get pathname — `getcwd`, 3C-389

`curl_addwch` — add a `wchar_t` character (with attributes) to a `curses` window and advance cursor, 3X-159

`curl_addwchstr` — add string of `wchar_t` characters (and attributes) to a `curses` window, 3X-161

`curl_addwstr` — add a string of `wchar_t` characters to a `curses` window and advance cursor, 3X-162

`curl_alecompat` — moving the cursor by character, 3X-163

`curl_attr` — `curses` character and window attribute control routines, 3X-164
 Attributes, 3X-164

`curl_getwch` — get (or push back) `wchar_t` characters from `curses` terminal keyboard, 3X-180
 Function Keys, 3X-180

`curl_getwstr` — get `wchar_t` character strings from `curses` terminal keyboard, 3X-184

`curl_inswch` — insert a `wchar_t` character before

-
- the character under the cursor in a curses window, 3X-196
 - `curs_inswstr` — insert `wchar_t` string before character under the cursor in a curses window, 3X-197
 - `curs_inwch` — get a `wchar_t` character and its attributes from a curses window, 3X-198
 - `curs_inwchstr` — get a string of `wchar_t` characters (and attributes) from a curses window, 3X-199
 - `curs_inwstr` — get a string of `wchar_t` characters from a curses window, 3X-200
 - `curs_pad` — create and display curses pads, 3X-207
 - `curses` — CRT handling and optimization package, 3X-228
 - curses bell and screen flash routines
 - `beep`, 3X-166
 - `curs_beep`, 3X-166
 - `flash`, 3X-166
 - curses borders, horizontal and vertical lines, create
 - `border`, 3X-168
 - `box`, 3X-168
 - `curs_border`, 3X-168
 - `wborder`, 3X-168
 - `whline`, 3X-168
 - `wvline`, 3X-168
 - curses character and window attribute control routines
 - `attroff`, 3X-164
 - `attron`, 3X-164
 - `attrset`, 3X-164
 - `curs_attr`, 3X-164
 - `standend`, 3X-164
 - `standout`, 3X-164
 - `wattroff`, 3X-164
 - `wattron`, 3X-164
 - `wattrset`, 3X-164
 - `wstandend`, 3X-164
 - `wstandout`, 3X-164
 - curses color manipulation routines
 - `can_change_colors`, 3X-171
 - `color_content`, 3X-171
 - curses color manipulation routines, *continued*
 - `curs_color`, 3X-171
 - `has_colors`, 3X-171
 - `init_color`, 3X-171
 - `init_pair`, 3X-171
 - `pair_content`, 3X-171
 - `start_color`, 3X-171
 - curses cursor and window coordinates
 - `curs_getyx`, 3X-185
 - `getbegyx`, 3X-185
 - `getmaxyx`, 3X-185
 - `getparyx`, 3X-185
 - `getyx`, 3X-185
 - curses environment query routines
 - `baudrate`, 3X-217
 - `curs_termattrs`, 3X-217
 - `erasechar`, 3X-217
 - `has_ic`, 3X-217
 - `has_il`, 3X-217
 - `killchar`, 3X-217
 - `longname`, 3X-217
 - `termattrs`, 3X-217
 - `termname`, 3X-217
 - curses interfaces to termcap library
 - `curs_termcap`, 3X-219
 - `tgetent`, 3X-219
 - `tgetflag`, 3X-219
 - `tgetnum`, 3X-219
 - `tgetstr`, 3X-219
 - `tgoto`, 3X-219
 - `tputs`, 3X-219
 - curses interfaces to terminfo database
 - `curs_terminfo`, 3X-220
 - `del_curterm`, 3X-220
 - `mvcur`, 3X-220
 - `putp`, 3X-220
 - `restartterm`, 3X-220
 - `set_curterm`, 3X-220
 - `setterm`, 3X-220
 - `setupterm`, 3X-220
 - `tigetflag`, 3X-220
 - `tigetnum`, 3X-220
 - `tigetstr`, 3X-220
 - `tparm`, 3X-220
 - `tputs`, 3X-220

curses interfaces to terminfo database, *continued*

- vidattr, 3X-220
- vidputs, 3X-220

curses library, See also form library, menu library, or panel library

- adjcurspos, 3X-163
- curs_alecompat, 3X-163
- movenextch, 3X-163
- moveprevch, 3X-163
- wadjcurspos, 3X-163
- wmovenextch, 3X-163
- wmoveprevch, 3X-163

curses miscellaneous utility routines

- curs_util, 3X-224
- delay_output, 3X-224
- filter, 3X-224
- flushinp, 3X-224
- getwin, 3X-224
- keyname, 3X-224
- putwin, 3X-224
- unctrl, 3X-224
- use_env, 3X-224

curses pads, create and display — curs_pad, 3X-207

- newpad, 3X-207
- pechochar, 3X-207
- pechowchar, 3X-207
- pnoutrefresh, 3X-207
- prefresh, 3X-207
- subpad, 3X-207

curses refresh control routines

- curs_touch, 3X-223
- is_linetouched, 3X-223
- is_wintouched, 3X-223
- touchline, 3X-223
- touchwin, 3X-223
- untouchwin, 3X-223
- wtouchln, 3X-223

curses screen initialization and manipulation routines

- curs_initscr, 3X-188
- delscreen, 3X-188
- endwin, 3X-188
- initscr, 3X-188
- isendwin, 3X-188

curses screen initialization and manipulation routines, *continued*

- newterm, 3X-188
- set_term, 3X-188

curses screen, read/write from/to file

- curs_scr_dump, 3X-213
- scr_dump, 3X-213
- scr_init, 3X-213
- scr_restore, 3X-213
- scr_set, 3X-213

curses soft label routines

- curs_slk, 3X-215
- slk_attroff, 3X-215
- slk_attron, 3X-215
- slk_attrset, 3X-215
- slk_clear, 3X-215
- slk_init, 3X-215
- slk_label, 3X-215
- slk_noutrefresh, 3X-215
- slk_refresh, 3X-215
- slk_restore, 3X-215
- slk_set, 3X-215
- slk_touch, 3X-215

curses terminal input option control routines

- cbreak, 3X-190
- curs_inopts, 3X-190
- echo, 3X-190
- halfdelay, 3X-190
- intrflush, 3X-190
- keypad, 3X-190
- meta, 3X-190
- nocbreak, 3X-190
- nodelay, 3X-190
- noecho, 3X-190
- noqiflush, 3X-190
- noraw, 3X-190
- notimeout, 3X-190
- qiflush, 3X-190
- raw, 3X-190
- timeout, 3X-190
- typeahead, 3X-190
- wtimeout, 3X-190

curses terminal keyboard

- curs_getstr, 3X-179
- getstr, 3X-179

curses terminal keyboard, *continued*

- mvgetstr, 3X-179
- mvwgetstr, 3X-179
- wgetnstr, 3X-179
- wgetstr, 3X-179

curses terminal keyboard, get characters

- curs_getch, 3X-175
- getch, 3X-175
- mvgetch, 3X-175
- mvwgetch, 3X-175
- ungetch, 3X-175
- wgetch, 3X-175

curses terminal output option control routines

- clearok, 3X-204
- curs_outopts, 3X-204
- idcok, 3X-204
- idlok, 3X-204
- immedok, 3X-204
- leaveok, 3X-204
- nl, 3X-204
- nonl, 3X-204
- scrollok, 3X-204
- setscereg, 3X-204
- wsetscrreg, 3X-204

curses window background manipulation routines

- bkgd, 3X-167
- bkgdset, 3X-167
- curs_bkgd, 3X-167
- wbkgd, 3X-167
- wbkgdset, 3X-167

curses window cursor

- curs_move, 3X-203
- move, 3X-203
- wmove, 3X-203

curses window, add character and advance cursor

- addch, 3X-155
- curs_addch, 3X-155
- echochar, 3X-155
- mvwaddch, 3X-155
- mvwaddch, 3X-155
- waddch, 3X-155
- wechochar, 3X-155

curses window, add string of characters

- addchnstr, 3X-157
- addchstr, 3X-157

curses window, add string of characters, *continued*

- curs_addchstr, 3X-157
- mvaddchnstr, 3X-157
- mvaddchstr, 3X-157
- mvwaddchnstr, 3X-157
- mvwaddchstr, 3X-157
- waddchnstr, 3X-157
- waddchstr, 3X-157

curses window, add string of characters and advance cursor

- addnstr, 3X-158
- addstr, 3X-158
- curs_addstr, 3X-158
- mvaddnstr, 3X-158
- mvaddstr, 3X-158
- mvwaddstr, 3X-158
- waddnstr, 3X-158
- waddstr, 3X-158

curses window, clear all or part

- clear, 3X-170
- clrtoobot, 3X-170
- clrtoeol, 3X-170
- curs_clear, 3X-170
- erase, 3X-170
- wclear, 3X-170
- wclrtoobot, 3X-170
- wclrtoeol, 3X-170
- werase, 3X-170

curses window, convert formatted input

- curs_scanw, 3X-212
- mvscanw, 3X-212
- mvwscanw, 3X-212
- scanw, 3X-212
- vwscanw, 3X-212
- wscanw, 3X-212

curses window, delete and insert lines

- curs_deleteln, 3X-174
- deleteln, 3X-174
- insdelln, 3X-174
- insertln, 3X-174
- wdeleteln, 3X-174
- winsdelln, 3X-174
- winsertln, 3X-174

curses window, delete character under cursor

- curs_delch, 3X-173

curses window, delete character under cursor, *continued*

- `delch`, 3X-173
- `mvdelch`, 3X-173
- `mvwdelch`, 3X-173
- `wdelch`, 3X-173

curses window, get character and its attributes

- `curs_inch`, 3X-186
- `inch`, 3X-186
- `mvinch`, 3X-186
- `mvwinch`, 3X-186
- `winch`, 3X-186

curses window, get string of characters

- `curs_inchstr`, 3X-187
- `curs_instr`, 3X-195
- `inchstr`, 3X-187
- `innstr`, 3X-195
- `instr`, 3X-195
- `mvinchstr`, 3X-187
- `mvinchstr`, 3X-187
- `mvinnstr`, 3X-195
- `mvinstr`, 3X-195
- `mvwinchnstr`, 3X-187
- `mvwinchnstr`, 3X-187
- `mvwinstr`, 3X-195
- `winchnstr`, 3X-187
- `winchnstr`, 3X-187
- `winnstr`, 3X-195
- `winstr`, 3X-195

curses window, insert character before character under cursor

- `curs_insch`, 3X-193
- `insch`, 3X-193
- `mvinsch`, 3X-193
- `mvwinsch`, 3X-193
- `winsch`, 3X-193

curses window, insert string before character under cursor

- `curs_instr`, 3X-194
- `insnstr`, 3X-194
- `instr`, 3X-194
- `mvinsnstr`, 3X-194
- `mvinsstr`, 3X-194

curses window, insert string before character under cursor, *continued*

- `mvwinsnstr`, 3X-194
- `mvwinsstr`, 3X-194
- `winsnstr`, 3X-194
- `winsstr`, 3X-194

curses window, scroll

- `curs_scroll`, 3X-214
- `scrl`, 3X-214
- `scroll`, 3X-214
- `wscrl`, 3X-214

curses windows and lines, refresh

- `curs_refresh`, 3X-210
- `doupdate`, 3X-210
- `redrawwin`, 3X-210
- `refresh`, 3X-210
- `wnoutrefresh`, 3X-210
- `wredrawln`, 3X-210
- `wrefresh`, 3X-210

curses windows, create

- `curs_window`, 3X-226
- `delwin`, 3X-226
- `derwin`, 3X-226
- `dupwin`, 3X-226
- `mvderwin`, 3X-226
- `mvwin`, 3X-226
- `newwin`, 3X-226
- `subwin`, 3X-226
- `syncok`, 3X-226
- `wcursyncup`, 3X-226
- `wsyncdown`, 3X-226
- `wsyncup`, 3X-226

curses windows, overlap and manipulate

- `copywin`, 3X-206
- `curs_overlay`, 3X-206
- `overlay`, 3X-206
- `overwrite`, 3X-206

curses windows, print formatted output

- `curs_printw`, 3X-209
- `mvprintw`, 3X-209
- `mvwprintw`, 3X-209
- `printw`, 3X-209
- `vwprintw`, 3X-209
- `wprintw`, 3X-209

curses, low-level routines

curses, low-level routines, *continued*

- `curs_kernel`, 3X-201
- `curs_set`, 3X-201
- `def_prog_mode`, 3X-201
- `def_shell_mode`, 3X-201
- `getsyx`, 3X-201
- `napms`, 3X-201
- `reset_prog_mode`, 3X-201
- `reset_shell_mode`, 3X-201
- `resettty`, 3X-201
- `ripcoffline`, 3X-201
- `savetty`, 3X-201
- `setsyx`, 3X-201

`cuserid` — get character-string representation of login name of user, 3S-239

D

data base subroutines — `dbm`, 3B-240

- `dbmclose`, 3B-240
- `dbmopen`, 3B-240
- `dbmclose`, 3B-240
- `dbmdelete`, 3B-240
- `dbmfetch`, 3B-240
- `dbmfirstkey`, 3B-240
- `dbmnextkey`, 3B-240
- `dbmstore`, 3B-240

database subroutines

- `ndbm`, 3-630

date and time

- convert to string — `asctime`, 3C-149
- convert user format date and time — `getdate`, 3C-390
- `gettimeofday`, 3C-464

date and time conversion — `strptime`, 3C-902

`dbm` — data base subroutines, 3B-240

`dbm_clearerr` — reset the error condition, 3-630

`dbm_close` close database, 3-630

`dbm_delete` — delete a key and its associated contents, 3-630

`dbm_error` — return an error condition, 3-630

`dbm_fetch` — access data stored under a key, 3-630

`dbm_firstkey` — return first key in the database, 3-630

`dbm_nextkey` — return next key in the database,

3-630

`dbm_open` — open database, 3-630

`dbm_store` — place data under a key, 3-630

`dbmclose` — data base subroutines, 3B-240

`dbmopen` — data base subroutines, 3B-240

decimal record from double-precision floating — `double_to_decimal`, 3-321

decimal record from extended-precision floating — `extended_to_decimal`, 3-321

decimal record from quadruple-precision floating — `quadruple_to_decimal`, 3-321

decimal record from single-precision floating — `single_to_decimal`, 3-321

decimal record to double-precision floating — `decimal_to_double`, 3-242

decimal record to extended-precision floating — `decimal_to_extended`, 3-242

decimal record to quadruple-precision floating — `decimal_to_quadruple`, 3-242

decimal record to single-precision floating — `decimal_to_single`, 3-242

`decimal_to_double` — decimal record to double-precision floating, 3-242

`decimal_to_extended` — decimal record to extended-precision floating, 3-242

`decimal_to_quadruple` — decimal record to quadruple-precision floating, 3-242

`decimal_to_single` — decimal record to single-precision floating, 3-242

define character class — `wctype`, 3I-1033

define default catalog — `setcat`, 3C-833

define the label for `pfmt()` and `lfmt()`. — `setlabel`, 3C-840

`delete` — data base subroutines, 3B-240

device number

`manage` — `makedev`, `major`, `minor`, 3C-557

`dgettext` — message handling function, 3I-460

`dial` — establish an outgoing terminal line connection, 3N-243

`difftime` — computes the difference between two calendar times, 3C-245

directories

- create, remove them in a path — `mkdirp`, `rmdirp`, 3G-601
- get current working directory pathname — `getwd`, 3B-475
- get pathname of current working directory — `getcwd`, 3C-389

directory operations

- `alphasort`, 3B-798
- `closedir`, 3C-246
- `directory`, 3C-246
- `opendir`, 3C-246
- `readdir`, 3C-246
- `readdir_r`, 3C-246
- `rewinddir`, 3C-246
- `scandir`, 3B-798
- `seekdir`, 3C-246
- `telldir`, 3C-246

`dirname` — report parent directory name of file path name, 3G-249

display error message in standard format — `pfmt`, 3C-696

display error message in standard format and pass to logging and monitoring services — `lfmt`, 3C-535, 3C-1008, 3C-1010

`div` — compute quotient and remainder, 3C-250

division and remainder operations

- `div`, 3C-250
- `ldiv`, 3C-250

`dladdr` — translate address to symbolic information., 3X-251

`dlclose` — close a shared object, 3X-253

`dlerror` — get diagnostic information, 3X-254

`dlopen` — open a shared object, 3X-255

`dlsym` — get address of symbol in shared object, 3X-257

`dn_comp` — resolver routines, 3N-743

`dn_expand` — resolver routines, 3N-743

`doconfig` — execute a configuration script, 3N-259

`double_to_decimal` — decimal record from double-precision floating, 3-321

`dup2` — duplicate an open file descriptor, 3C-263

duplicate an open file descriptor — `dup2`, 3C-263

dynamic linking

close a shared object — `dlclose`, 3X-253

get address of symbol in shared object — `dlsym`, 3X-257

get diagnostic information — `dlerror`, 3X-254

open a shared object — `dlopen`, 3X-255

E

`echowchar` — add a `wchar_t` character (with attributes) to a curses window and advance cursor, 3X-159

`econvert` — convert number to ASCII, 3-264

`ecvt` — convert number to ASCII, 3-264

`edata` — last location in program, 3C-307

`elf` — object file access library, 3E-267

get entries from name list — `nlist`, 3E-675

`elf_begin` — process ELF object files, 3E-278

`elf_cntl` — control an elf file descriptor, 3E-283

`elf_end` — process ELF object files, 3E-278

`elf_errmsg` — error handling, 3E-284

`elf_errno` — error handling, 3E-284

`elf_fill` — set fill byte, 3E-285

`elf_flagdata` — manipulate flags, 3E-286

`elf_flagehdr` — manipulate flags, 3E-286

`elf_flagelf` — manipulate flags, 3E-286

`elf_flagphdr` — manipulate flags, 3E-286

`elf_flagphdr` — manipulate flags, 3E-286

`elf_flagshdr` — manipulate flags, 3E-286

`elf_getarhdr` — retrieve archive member header, 3E-288

`elf_getarsym` — retrieve archive symbol table, 3E-289

`elf_getbase` — get the base offset for an object file, 3E-290

`elf_getdata` — get section data, 3E-291

`elf_getident` — retrieve file identification data, 3E-295

`elf_getscn` — get section information, 3E-297

`elf_hash` — compute hash value, 3E-299

`elf_kind` — determine file type, 3E-300

`elf_ndxscn` — get section information, 3E-297

`elf_newdata` — get section data, 3E-291

elf_newscn — get section information, 3E-297
 elf_next — process ELF object files, 3E-278
 elf_nextscn — get section information, 3E-297
 elf_rand — process ELF object files, 3E-278
 elf_rawdata — get section data, 3E-291
 elf_rawfile — retrieve uninterpreted file contents, 3E-301
 elf_strptr — make a string pointer, 3E-302
 elf_update — update an ELF descriptor, 3E-303
 elf_version — coordinate ELF library and application versions, 3E-306
 elf_fsize — return the size of an object file type, 3E-272
 elf32_getehdr — retrieve class-dependent object file header, 3E-273
 elf_getphdr — retrieve class-dependent program header table, 3E-274
 elf32_getshdr — retrieve class-dependent section header, 3E-275
 elf32_newehdr — retrieve class-dependent object file header, 3E-273
 elf_newphdr — retrieve class-dependent program header table, 3E-274
 elf32_xlatetof — class-dependent data translation, 3E-276
 elf32_xlatetom — class-dependent data translation, 3E-276
 encryption
 determine whether a buffer of characters is encrypted — `isencrypt`, 3G-498
 encryption, password
 — `crypt`, 3C-146
 — `encrypt`, 3C-146
 — `setkey`, 3C-146
 end — last location in program, 3C-307
 endac — get audit control file information, 3-378
 endaclass — close `audit_class` database file, 3-380
 endaevent — close `audit_event` database file, 3-383
 endauuser — get `audit_user` database entry, 3-385
 endgrent — get group entry from database, 3C-398
 ednpwent — get password entry from user database, 3C-434
 endservent — get service entry, 3N-445
 endspent — get shadow password database entry, 3C-453
 endutent — access `utmp` file entry, 3C-467
 endutxent — access `utmpx` file entry, 3C-469
 environment name
 return value — `getenv`, 3C-396
 environment variables
 change or add value — `putenv`, 3C-716
 erf — error functions, 3M-308
 erfc — error functions, 3M-308
 error functions
 — `erf`, 3M-308
 — `erfc`, 3M-308
 error messages
 get string — `strerror`, 3C-887, 3N-962
 error messages, system
 print — `perror`, 3C-695
 establish a transport endpoint — `t_open`, 3N-941
 etext — last location in program, 3C-307
 Ethernet address mapping operations
 — `ethers`, 3N-309
 ethers — Ethernet address mapping operations, 3N-309
 EUC character
 convert EUC character from the stream to Process Code — `fgetwc`, 3I-474
 convert Process Code character to EUC — `fputwc`, 3I-720
 EUC character bytes
 — `euclen`, 3I-311
 EUC characters
 convert a string of EUC characters from the stream to Process Code — `getws`, 3I-477
 convert a string of EUC characters from the stream to Process Code `getwfs`, 3I-477
 convert a string of Process Code characters to EUC characters and put it on a stream

EUC characters, *continued*

fputws, 3I-721

EUC codeset, get information

— getwidth, 3I-476

EUC codesets, get information

— cset, 3I-147

— csetcol, 3I-147

— csetlen, 3I-147

— csetno, 3I-147

— wcsetno, 3I-147

EUC display width

— euccol, 3I-311

— eucscol, 3I-311

euccol — get EUC character display width, 3I-311

euclen — get EUC byte length, 3I-311

Euclidean distance function — hypot, 3M-484

eucscol — get EUC string display width, 3I-311

Executable and Linking Format, See *elf*

exit — terminate process, 3C-312

exit program

add routine — atexit, 3C-113

exp — exponential function, 3M-313

expm1 — exponential minus 1 function, 3M-313

exponential function — exp, 3M-313

exponential minus 1 function — expm1, 3M-313

Extended Unix Code, See *EUC*

extended_to_decimal — decimal record from
extended-precision floating, 3-321

external data representation

See *XDR*, 3N-1037

F

fabs() function, 3M-489

fattach — attach a STREAMS-based file descriptor to an object in the file system name space, 3C-314

fconvert — convert number to ASCII, 3-264

fcvt — convert number to ASCII, 3-264

fdatasync — synchronize a file's data, 3R-317

fdetach — detach a name from a STREAMS-based file descriptor, 3C-318

fetch — data base subroutines, 3B-240

ffs — find first set bit, 3C-320

fgetgrent — get group entry from file, 3C-398

fgetgrent_r — get group entry from file, 3C-398

fgetpos — reposition a file pointer in a stream,

3C-373

fgetpwent — get password entry from a file,

3C-434

fgetpwent_r — get password entry from a file,

3C-434

fgetspent — get shadow password database

entry, 3C-453

fgetspent_r — get shadow password database

entry(reentrant), 3C-453

fgetwc — get EUC character from stream, 3I-474

fgetws — em convert a string of EUC characters

from the stream to Process Code, 3I-477

FIFO

create a new one — mkfifo, 3C-602

file descriptor

duplicate an open one — dup2, 3C-263

get table size — getdtablesize, 3B-395

STREAMS-based, attach to an object in file system name space — fattach, 3C-314

STREAMS-based, detach a name — fdetach, 3C-318

test for a STREAMS file — isastream, 3C-497

file descriptors

apply or remove advisory lock on open file —

flock, 3B-323

file name

make a unique one — mktemp, 3C-603

file path names

returns the real file name — realpath,

3C-732

file pointer in a stream

reposition — fsetpos, fgetpos, 3C-373

file tree

recursively descend — ftw, 3C-376

file_to_decimal — decimal record from character stream, 3-899

files

allows sections of file to be locked — lockf,

3C-548

files, *continued*

- remove, 3C-742
 - report parent directory of file path name —
dirname, 3G-249
 - search for named file in named directories —
pathfind, 3G-693
 - set a file to a specified length — truncate,
3C-996
 - synchronize a file's in-memory state with that
on the physical medium — fsync,
3C-374
- firstkey — data base subroutines, 3B-240
- floating-point number**
convert to string — ecvt, 3C-266
- floating-point number, determine type**
— finite, 3C-499
— fpclass, 3C-499
— isnan, 3C-499
— isnand, 3C-499
— isnanf, 3C-499
— unordered, 3C-499
- floating-point number, double-precision**
convert string to — atof, 3C-906
- floating-point numbers, manipulate**
— frexp, 3C-369
— ldexp, 3C-369
— logb, 3C-369
— modf, 3C-369
— modff, 3C-369
— nextafter, 3C-369
- floating-point, manipulate**
— scalb, 3C-369
- flock — apply or remove an advisory lock on an
open file, 3B-323
- flockfile — acquire and release stream lock,
3S-325
- floor — round to integral value, 3M-327
- fmod() function, 3M-489
- fmtmsg — display a message on stderr or system
console, 3C-328
- fopen — open stream, 3B-333
- form library, See also curses library
- formatted input conversion — wsscanf, 3I-1035
- formatted output conversion

formatted output conversion, *continued*

- fprintf, 3B-702
 - printf, 3B-702
 - sprintf, 3B-702
 - vsprintf, 3B-702
 - vprintf, 3B-702
 - vsprintf, 3B-702
- forms — character based forms package, 3X-363
- forms field attributes, set and get**
— field_buffer, 3X-344
— field_status, 3X-344
— form_field_buffer, 3X-344
— set_field_buffer, 3X-344
— set_field_status, 3X-344
— set_max_field, 3X-344
- forms field characteristics**
— dynamic_field_info, 3X-345
— field_info, 3X-345
— form_field_info, 3X-345
- forms field data type validation**
— field_arg, 3X-351
— field_type, 3X-351
— form_field_validation, 3X-351
— set_field_type, 3X-351
- forms field option routines**
— field_opts, 3X-348
— field_opts_off, 3X-348
— field_opts_on, 3X-348
— form_field_opts, 3X-348
— set_field_opts, 3X-348
- forms field, off-screen data ahead or behind**
— data_ahead, 3X-338
— data_behind, 3X-338
— form_data, 3X-338
- forms fields, create and destroy**
— dup_field, 3X-347
— form_field_new, 3X-347
— free_field, 3X-347
— link_field, 3X-347
— new_field, 3X-347
- forms fieldtype routines**
— form_fieldtype, 3X-352
— free_fieldtype, 3X-352
— link_fieldtype, 3X-352
— new_fieldtype, 3X-352

-
- forms fieldtype routines, *continued*
 - set_fieldtype_arg, 3X-352
 - set_fieldtype_choice, 3X-352
 - forms option routines
 - form_opts, 3X-358
 - form_opts_off, 3X-358
 - form_opts_on, 3X-358
 - set_form_opts, 3X-358
 - forms pagination
 - form_new_page, 3X-357
 - new_page, 3X-357
 - set_new_page, 3X-357
 - forms window and subwindow association routines
 - form_sub, 3X-362
 - form_win, 3X-362
 - scale_form, 3X-362
 - set_form_sub, 3X-362
 - set_form_win, 3X-362
 - forms window cursor, position
 - form_cursor, 3X-337
 - pos_form_cursor, 3X-337
 - forms, application-specific routines
 - field_init, 3X-354
 - field_term, 3X-354
 - form_hook, 3X-354
 - form_init, 3X-354
 - form_term, 3X-354
 - set_field_init, 3X-354
 - set_field_term, 3X-354
 - set_form_init, 3X-354
 - set_form_term, 3X-354
 - forms, associate application data
 - field_userptr, 3X-350
 - form_field_userptr, 3X-350
 - form_userptr, 3X-361
 - set_field_userptr, 3X-350
 - set_form_userptr, 3X-361
 - forms, command processor
 - form_driver, 3X-339
 - forms, connect fields
 - field_count, 3X-342
 - form_field, 3X-342
 - form_fields, 3X-342
 - move_field, 3X-342
 - set_form_fields, 3X-342
 - forms, create and destroy
 - form_new, 3X-356
 - free_form, 3X-356
 - new_form, 3X-356
 - forms, format general appearance
 - field_just, 3X-346
 - form_field_just, 3X-346
 - set_field_just, 3X-346
 - forms, format general display attributes
 - field_back, 3X-343
 - field_fore, 3X-343
 - field_pad, 3X-343
 - form_field_attributes, 3X-343
 - set_field_back, 3X-343
 - set_field_fore, 3X-343
 - set_field_pad, 3X-343
 - forms, set current page and field
 - current_field, 3X-359
 - field_index, 3X-359
 - form_page, 3X-359
 - set_current_field, 3X-359
 - set_form_page, 3X-359
 - forms, write/erase from associated subwindows
 - form_post, 3X-360
 - post_form, 3X-360
 - unpost_form, 3X-360
 - fpgetmask — IEEE floating-point environment control, 3C-366
 - fpgetround — IEEE floating-point environment control, 3C-366
 - fpgetsticky — IEEE floating-point environment control, 3C-366
 - fprintf — formatted output conversion, 3B-702, 3S-706
 - fpsetmask — IEEE floating-point environment control, 3C-366
 - fpsetround — IEEE floating-point environment control, 3C-366
 - fpsetsticky — IEEE floating-point environment control, 3C-366
 - fputc — put wide character on stream, 3I-720
 - fputws — convert a string of Process Code characters to EUC characters and put it on a stream,

3I-721

- free — memory allocator, 3X-126
- freopen — open stream, 3B-333
- fscanf — convert formatted input, 3S-799
- fsetpos — reposition a file pointer in a stream, 3C-373
- fsync — synchronize a file's in-memory state with that on the physical medium, 3C-374
- ftime — get date and time, 3B-375
- ftruncate — set a file to a specified length, 3C-996
- ftw — walk a file tree, 3C-376
- func_to_decimal — decimal record from character function, 3-899
- funlockfile — acquire and release stream lock, 3S-325

G

- gamma — log gamma function, 3M-539
- gamma_r — log gamma function, 3M-539
- gconvert — convert number to ASCII, 3-264
- gcvt — convert number to ASCII, 3-264
- general terminal interface
 - cfgetispeed, 3-967
 - cfgetospeed, 3-967
 - cfsetispeed, 3-967
 - cfsetospeed, 3-967
 - tcdrain, 3-967
 - tcflow, 3-967
 - tcflush, 3-967
 - tcgetattr, 3-967
 - tcgetpgrp, 3-967
 - tcgetsid, 3-967
 - tcsendbreak, 3-967
 - tcsetattr, 3-967
 - tcsetpgrp, 3-967
 - termios, 3-967
- generate path name for controlling terminal
 - ctermid, 3S-148
 - ctermid_r, 3S-148
- generic transport name-to-address translation
 - netdir, 3N-633
 - netdir_free, 3N-633

generic transport name-to-address translation, *continued*

- netdir_getbyaddr, 3N-633
- netdir_getbyname, 3N-633
- netdir_mergeaddr, 3N-633
- netdir_options, 3N-633
- netdir_perror, 3N-633
- netdir_sperror, 3N-633
- taddr2uaddr, 3N-633
- uaddr2taddr, 3N-633
- get (or push back) wchar_t characters from curses terminal keyboard
 - curs_getwch, 3X-180
 - getwch, 3X-180
 - mvgetwch, 3X-180
 - mvwgetwch, 3X-180
 - ungetwch, 3X-180
 - wgetwch, 3X-180
- get a string of wchar_t characters (and attributes) from a curses window — curs_inwchstr, 3X-199
 - inwchnstr, 3X-199
 - inwchstr, 3X-199
 - mvinwchnstr, 3X-199
 - mvinwchstr, 3X-199
 - mvwinwchnstr, 3X-199
 - mvwinwchstr, 3X-199
 - winwchnstr, 3X-199
 - winwchstr, 3X-199
- get a string of wchar_t characters from a curses window — curs_inwstr, 3X-200
 - innwstr, 3X-200
 - inwstr, 3X-200
 - mvinnwstr, 3X-200
 - mvinwstr, 3X-200
 - mvinnwstr, 3X-200
 - mvwinwstr, 3X-200
 - winnwstr, 3X-200
 - winwstr, 3X-200
- get a wchar_t character and its attributes from a curses window — curs_inwch, 3X-198
 - inwch, 3X-198
 - mvinwch, 3X-198
 - mvwinwch, 3X-198
 - winwch, 3X-198

get error message string — `t_strerror`, 3N-962
 get service entry — `getservbyname`, 3N-445
 `endservent`, 3N-445
 `getservbyname_r`, 3N-445
 `getservbyport`, 3N-445
 `getservbyport_r`, 3N-445
 `getservent`, 3N-445
 `getservent_r`, 3N-445
 `setservent`, 3N-445
 get `wchar_t` character strings from curses terminal
 keyboard — `curl_getwstr`, 3X-184
 `getnwstr`, 3X-184
 `getwstr`, 3X-184
 `mvgetnwstr`, 3X-184
 `mvgetwstr`, 3X-184
 `mvwgetnwstr`, 3X-184
 `mvwgetwstr`, 3X-184
 `wgetnwstr`, 3X-184
 `wgetwstr`, 3X-184
`getacdir` — get audit control file information,
 3-378
`getacflg` — get audit control file information,
 3-378
`getacinfo` — get audit control file information,
 3-378
`getacmin` — get audit control file information,
 3-378
`getacna` — get audit control file information, 3-378
`getaclclassent` — get audit_class database entry,
 3-380
`getaclclassent` — get audit_class database entry,
 3-380
`getaclclassnam` — get audit_class database entry,
 3-380
`getaclclassnam` — get audit_class database entry,
 3-380
`getaclclassnam` — get audit_class database entry,
 3-380
`getauditflags()` — generate process audit state,
 3-397
`getauditflagsbin()` — convert audit flag
 specifications, 3-382
`getauditflagschar()` — convert audit flag
 specifications, 3-382
`getaeuevent` — get audit_event database entry,
 3-383
`getaeuevent_r` — get audit_event database entry,
 3-383
`getaeuevnam` — get audit_event database entry,
 3-383
`getaeuevnam_r` — get audit_event database entry,
 3-383
`getaeuevnonam` — get audit_event database entry,
 3-383
`getaeuevnum` — get audit_event database entry,
 3-383
`getaeuevnum_r` — get audit_event database entry,
 3-383
`getaeuserent` — get audit_user database entry,
 3-385
`getaeuserent_r` — get audit_user database entry,
 3-385
`getaeusernam` — get audit_user database entry,
 3-385
`getaeusernam_r` — get audit_user database entry,
 3-385
`getcwd` — get pathname of current working direc-
 tory, 3C-389
`getdate` — convert user format date and time,
 3C-390
 General Specifications, 3C-392
 Internal Format Conversion, 3C-391
 Modified Conversion Specifications, 3C-391
`getdtablesize` — get file descriptor table size,
 3B-395
`getenv` — return value for environment name,
 3C-396
`getgrent` — get group entry from database,
 3C-398
`getgrent_r` — get group entry from database,
 3C-398
`getgrgid` — get group entry from database,
 3C-398
`getgrgid_r` — get group entry from database,
 3C-398
`getgrnam` — get group entry from database,
 3C-398

getgrnam_r — get group entry from database, 3C-398
gethostid — get unique identifier of current host, 3B-406
gethostname — get name of current host, 3B-407
gethrtime — get high resolution real time, 3C-408
gethrvtime — get high resolution virtual time, 3C-408
getlogin — get login name, 3C-409
getlogin_r — get login name, 3C-409
getmntany — get mnttab file information, 3C-410
getmntent — get mnttab file information, 3C-410
getnwstr — get `wchar_t` character strings from curses terminal keyboard, 3X-184
getopt — get option letter from argument vector, 3C-421
getpagesize — get system page size, 3B-424
getpass — read a password, 3C-425
getpeername — get name of peer connected to socket, 3N-426
getpriority — get scheduling priority for process, process group or user, 3B-427
getpublickey — retrieve public or secret key, 3N-432
getpw — get passwd entry from UID, 3C-433
getpwent — get password entry from user database, 3C-434
getpwent_r — get password entry from user database, 3C-434
getpwnam — get password entry from user database, 3C-434
getpwnam_r — get password entry from user database, 3C-434
getpwuid — get password entry from user database, 3C-434
getpwuid_r — get password entry from user database, 3C-434
getrusage — get information about resource utilization, 3B-441
getsecretkey — retrieve public or secret key, 3N-432
getservbyname — get service entry, 3N-445
getservbyname_r — get service entry, 3N-445
getservbyport — get service entry, 3N-445
getservbyport_r — get service entry, 3N-445
getservent — get service entry, 3N-445
getservent_r — get service entry, 3N-445
getspent — get shadow password database entry, 3C-453
getspent_r — get shadow password database entry (reentrant), 3C-453
getspnam — get shadow password database entry, 3C-453
getspnam_r — get shadow password database entry (reentrant), 3C-453
getsubopt — parse suboptions from a string, 3C-457
gettext — message handling function, 3I-460
gettimeofday — get date and time, 3C-464, 3B-463
gettxt — retrieve a text string, 3C-465
getutent — access utmp file entry, 3C-467
getutid — access utmp file entry, 3C-467
getutline — access utmp file entry, 3C-467
getutmp — access utmpx file entry, 3C-469
getutmp_r — access utmpx file entry, 3C-469
getutxent — access utmpx file entry, 3C-469
getutxid — access utmpx file entry, 3C-469
getutxline — access utmpx file entry, 3C-469
getvfsany — get vfstab file entry, 3C-472
getvfssent — get vfstab file entry, 3C-472
getvfssfile — get vfstab file entry, 3C-472
getvfsspec — get vfstab file entry, 3C-472
getwc — get EUC character from stream, 3I-474
getwch — get (or push back) `wchar_t` characters from curses terminal keyboard, 3X-180
getwchar — get EUC character from stdin, 3I-474
getwd — get current working directory pathname, 3B-475
getwidth — get codeset information, 3I-476
getws — convert a string of EUC characters from the stream to Process Code, 3I-477
getwstr — get `wchar_t` character strings from curses terminal keyboard, 3X-184

gmatch — shell global pattern matching, 3G-478
grantpt — grant access to the slave pseudo-terminal device, 3C-479
group IDs
 set real and effective — setregid, 3B-843
 set terminal foreground process group id — tcsetpgrp, 3C-966
group IDs, supplementary
 initialize — initgroups, 3C-495
groups
 — endgrent, 3C-398
 — fgetgrent, 3C-398
 — fgetgrent_r, 3C-398
 — getgrent, 3C-398
 — getgrent_r, 3C-398
 — getgrid, 3C-398
 — getgrid_r, 3C-398
 — getgrnam, 3C-398
 — getgrnam_r, 3C-398
 — setgrent, 3C-398

H

halt system processor
 — reboot, 3B-733
hash-table search routine
 — hsearch, 3C-480
hasmntopt — get mnttab file information, 3C-410
hcreate — create hash table, 3C-480
hdestroy — destroy hash table, 3C-480
host ID
 get unique identifier of current host — gethostid, 3B-406
host machines, remote
 return information about users — rusers, rnusers, 3N-794
host name
 get name of current host — gethostname, 3B-407
 set name of current host — sethostname, 3B-407
hsearch — hash-table search routine, 3C-480
hyperbolic functions
 — acosh, 3M-483
 — asinh, 3M-483

hyperbolic functions, *continued*
 — atanh, 3M-483
 — cosh, 3M-483
 — sinh, 3M-483
 — tanh, 3M-483
hypot — Euclidean distance function, 3M-484

I

I/O asynchronous
 read and write operations — aio_read, aio_write, 3R-101
I/O multiplexing, synchronous
 — select, 3C-814
I/O package
 standard buffered I/O — stdio, 3S-879
I/O, asynchronous
 cancel request — aio_cancel, 3R-97
 file synchronization — aio_sync, 3R-99
 retrieve error status — aio_error, 3R-103
 retrieve return status — aio_return, 3R-103
 wait for request — aio_suspend, 3R-106
I/O, requests
 list — lio_listio, 3R-540
iconv — code conversion function, 3-485
iconv_close — code conversion deallocation function, 3-487
iconv_open — code conversion allocation function, 3-488
IEEE arithmetic
 convert floating-point number to string — ecvt, 3C-266
IEEE floating-point environment control
 — fpgetmasks, 3C-366
 — fpgetround, 3C-366
 — fpgetsticky, 3C-366
 — fpsetmask, 3C-366
 — fpsetround, 3C-366
 — fpsetsticky, 3C-366
ilogb() function, 3M-489
index — string operations, 3B-492
inet — Internet address manipulation, 3N-493
inet_addr — Internet address manipulation, 3N-493
inet_lnaof — Internet address manipulation,

3N-493
inet_makeaddr — Internet address manipulation, 3N-493
inet_netof — Internet address manipulation, 3N-493
inet_network — Internet address manipulation, 3N-493
inet_ntoa — Internet address manipulation, 3N-493
initgroups — initialize the supplementary group access list, 3C-495
initialize kernel statistics facility
 — **kstat_close**, 3K-525
 — **kstat_open**, 3K-525
initstate — routines for changing random number generators, 3B-727
innwstr — get a string of `wchar_t` characters from a curses window, 3X-200
input conversion
 convert from `wchar_t` string — **wsscanf**, 3I-1035
input/output package
 standard buffered I/O — **stdio**, 3S-879
insert a `wchar_t` character before the character under the cursor in a curses window —
 curlns, 3X-196
 inswch, 3X-196
 mvinswch, 3X-196
 mvwinswch, 3X-196
 winswch, 3X-196
insert `wchar_t` string before character under the cursor in a curses window — **curlns**, 3X-197
 insnwstr, 3X-197
 inswstr, 3X-197
 mvinsnwstr, 3X-197
 mvinswstr, 3X-197
 mvwinsnwstr, 3X-197
 mvwinswstr, 3X-197
 winsnwstr, 3X-197
 winswstr, 3X-197
insnwstr — insert `wchar_t` string before character under the cursor in a curses window, 3X-197
insque — insert element to a queue, 3C-496
inswch — insert a `wchar_t` character before the character under the cursor in a curses window, 3X-196
inswstr — insert `wchar_t` string before character under the cursor in a curses window, 3X-197
Internet address manipulation — **inet**, 3N-493
 inet_addr, 3N-493
 inet_lnaof, 3N-493
 inet_makeaddr, 3N-493
 inet_netof, 3N-493
 inet_network, 3N-493
 inet_ntoa, 3N-493
Interprocess Communication
 create a new FIFO — **mkfifo**, 3C-602
interprocess communication
 standard package — **ftok**, 3C-883
inwch — get a `wchar_t` character and its attributes from a curses window, 3X-198
inwchnstr — get a string of `wchar_t` characters (and attributes) from a curses window, 3X-199
inwchstr — get a string of `wchar_t` characters (and attributes) from a curses window, 3X-199
inwstr — get a string of `wchar_t` characters from a curses window, 3X-200
isatty — find name of a terminal, 3C-1001
isencrypt — determine whether a buffer of characters is encrypted, 3G-498
isenglish — Process Code character classification macros and functions, 3I-500
isideogram — Process Code character classification macros and functions, 3I-500
isnan() function, 3M-489
isnumber — Process Code character classification macros and functions, 3I-500
isphonogram — Process Code character classification macros and functions, 3I-500
isspecial — Process Code character classification macros and functions, 3I-500
iswalnum — Process Code character classification macros and functions, 3I-500
iswalpha — Process Code character classification

macros and functions, 3I-500
 iswascii — Process Code character classification
 macros and functions, 3I-500
 iswcntrl — Process Code character classification
 macros and functions, 3I-500
 iswctype — test character for specified class,
 3I-502
 iswdigit — Process Code character classification
 macros and functions, 3I-500
 iswgraph — Process Code character classification
 macros and functions, 3I-500
 iswlower — Process Code character classification
 macros and functions, 3I-500
 iswprint — Process Code character classification
 macros and functions, 3I-500
 iswpunct — Process Code character classification
 macros and functions, 3I-500
 iswspace — Process Code character classification
 macros and functions, 3I-500
 iswupper — Process Code character classification
 macros and functions, 3I-500
 iswxdigit — Process Code character classification
 macros and functions, 3I-500

J

j0 — Bessel function, 3M-122
 j1 — Bessel function, 3M-122
 jn — Bessel function, 3M-122

K

Kerberos authentication library

- kerberos, 3N-503
- krb_get_cred, 3N-503
- krb_kntoln, 3N-503
- krb_mk_err, 3N-503
- krb_mk_req, 3N-503
- krb_mk_safe, 3N-503
- krb_rd_err, 3N-503
- krb_rd_req, 3N-503
- krb_rd_safe, 3N-503
- krb_set_key, 3N-503

Kerberos authentication routines for RPC

Kerberos authentication routines for RPC, *continued*

- authkerb_getucrd, 3N-507
- authkerb_seccreate, 3N-507
- kerberos_rpc, 3N-507
- svc_kerb_reg, 3N-507

Kerberos authentication routines via network stream sockets

- krb_net_read, 3N-513
- krb_net_write, 3N-513
- krb_recauth, 3N-513
- krb_sendauth, 3N-513

Kerberos ticket cache file name

- krb_set_tkt_string, 3N-516

Kerberos utility routines

- krb_get_admhst, 3N-511
- krb_get_krbhst, 3N-511
- krb_get_lrealm, 3N-511
- krb_get_phost, 3N-511
- krb_realmofhost, 3N-511

kernel virtual memory functions

copy data from kernel image or running system

- kvm_read, 3K-533

copy data to kernel image or running system —

- kvm_write, 3K-533

get entries from kernel symbol table —

- kvm_nlist, 3K-530

get invocation argument for process —

- kvm_getcmd, 3K-527

get u-area for process — kvm_getu, 3K-527

kstat — kernel statistics facility, 3K-517

kstat_chain_update — update the kstat
header chain, 3K-523

kstat_close — initialize kernel statistics
facility, 3K-525

kstat_data_lookup — find a kstat by name,
3K-524

kstat_lookup — find a kstat by name,
3K-524

kstat_open — initialize kernel statistics facil-
ity, 3K-525

kstat_read — read or write kstat data,
3K-526

kstat_write — read or write kstat data,
3K-526

specify a kernel to examine — kvm_open,

kvm_close,
 kernel virtual memory functions, *continued*
 3K-531
 killpg — send signal to a process group, 3B-510
 kstat — kernel statistics facility, 3K-517
 kstat_chain_update — update the kstat header
 chain, 3K-523
 kstat_close — initialize kernel statistics facility,
 3K-525
 kstat_data_lookup — find a kstat by name,
 3K-524
 kstat_lookup — find a kstat by name, 3K-524
 kstat_open — initialize kernel statistics facility,
 3K-525
 kstat_read — read or write kstat data, 3K-526
 kstat_write — read or write kstat data, 3K-526
 kvm_close — specify kernel to examine, 3K-531
 kvm_getcmd — get invocation arguments for pro-
 cess, 3K-527
 kvm_getproc — read system process structures,
 3K-529
 kvm_getu — get u-area for process, 3K-527
 kvm_nextproc — read system process structures,
 3K-529
 kvm_nlist — get entries from kernel symbol table,
 3K-530
 kvm_open — specify kernel to examine, 3K-531
 kvm_read — copy data from kernel image or run-
 ning system, 3K-533
 kvm_setproc — read system process structures,
 3K-529
 kvm_write — copy data to kernel image or run-
 ning system, 3K-533

L

labs — return absolute value of long integer, 3C-91
 language information
 — nl_langinfo, 3C-673
 Latin characters, case conversion
 — towlower, 3I-1019
 — towupper, 3I-1019
 — wconv, 3I-1019

ldiv — compute quotient and remainder, 3C-250
 lfmt — display error message in standard format
 and pass to logging and monitoring services,
 3C-535

lgamma — log gamma function, 3M-539
 lgamma_r — log gamma function, 3M-539

library routines for client side calls

- clnt_call, 3N-756
- clnt_freeres, 3N-756
- clnt_geterr, 3N-756
- clnt_perrno, 3N-756
- clnt_perror, 3N-756
- clnt_sperrno, 3N-756
- clnt_sperror, 3N-756
- rpc_broadcast, 3N-756
- rpc_broadcast_exp, 3N-756
- rpc_call, 3N-756
- rpc_clnt_calls, 3N-756

library routines for dealing with creation and mani-
pulation of CLIENT handles —

- rpc_clnt_create, 3N-759
- clnt_control, 3N-759
- clnt_create, 3N-759
- clnt_create_vers, 3N-759
- clnt_destroy, 3N-759
- clnt_dg_create, 3N-759
- clnt_pcreateerror, 3N-759
- clnt_raw_create, 3N-759
- clnt_spcreateerror, 3N-759
- clnt_tli_create, 3N-759
- clnt_tp_create, 3N-759
- clnt_vc_create, 3N-759
- rpc_createerr, 3N-759

linear search and update routine

- lfind, 3C-550
- lsearch, 3C-550

lio_listio — list directed I/O, 3R-540

list directed I/O — lio_listio, 3R-540

listen — listen for connections on a socket,
3N-543

llabs — return absolute value of long long integer,
3C-91

lldiv — compute quotient and remainder, 3C-250

locale

locale, *continued*
 modify and query a program's locale — `setlocale`, 3C-841

localeconv — get numeric formatting information, 3C-544

lock
 apply or remove advisory lock on open file — `flock`, 3B-323

lock address space
 — `mlockall`, 3C-608

lock memory pages
 — `mlock`, 3C-606

lockf — allows sections of file to be locked, 3C-548

lockfile for user's mailbox
 — `maillock`, 3X-554

log — natural logarithm, 3M-313

log gamma function
 — `gamma`, 3M-539
 — `gamma_r`, 3M-539
 — `lgamma`, 3M-539
 — `lgamma_r`, 3M-539

log10 — logarithm, base 10, 3M-313

log1p — natural logarithm of 1 plus argument, 3M-313

logarithm, base 10 — `log10`, 3M-313

logarithm, natural — `log`, 3M-313

logb() function, 3M-491

login name
 — `getlogin`, 3C-409
 — `getlogin_r`, 3C-409

longjmp — non-local goto, 3B-834, 3C-837

M

madvise — provide advice to VM system, 3-552

maillock — manage lockfile for user's mailbox, 3X-554

makecontext — manipulate user contexts, 3C-556

malloc — memory allocator, 3X-126

manipulate sets of signals — `sigsetops`, 3C-860
`sigaddset`, 3C-860
`sigdelset`, 3C-860
`sigemptyset`, 3C-860
`sigfillset`, 3C-860

manipulate sets of signals — `sigsetops`, *continued*
`sigismember`, 3C-860

math library exception-handling — `matherr`, 3M-565

mathematical functions
 — `acos`, 3M-995
 — `asin`, 3M-995
 — `atan`, 3M-995
 — `atan2`, 3M-995
 — `cbrt`, 3M-877
 — `ceil`, 3M-327
 — `cos`, 3M-995
 — `exp`, 3M-313
 — `expm1`, 3M-313
 — `floor`, 3M-327
 — `gamma`, 3M-539
 — `gamma_r`, 3M-539
 hyperbolic arccosine — `acosh`, 3M-483
 hyperbolic arcsine — `asinh`, 3M-483
 hyperbolic arctangent — `atanh`, 3M-483
 hyperbolic cosine — `cosh`, 3M-483
 hyperbolic sine — `sinh`, 3M-483
 hyperbolic tangent — `tanh`, 3M-483
 — `hypot`, 3M-484
 — `j0`, 3M-122
 — `j1`, 3M-122
 — `jn`, 3M-122
 — `lgamma`, 3M-539
 — `lgamma_r`, 3M-539
 — `log`, 3M-313
 — `log10`, 3M-313
 — `log1p`, 3M-313
 — `pow`, 3M-313
 — `rint`, 3M-327
 — `sin`, 3M-995
 — `sqrt`, 3M-877
 — `tan`, 3M-995
 — `y0`, 3M-122
 — `y1`, 3M-122
 — `yn`, 3M-122

`matherr` — math library exception-handling, 3M-565

`mbstowcs` — multibyte string functions, 3C-572

`mctl` — memory management control, 3B-573

memory — memory operations, 3C-575

-
- memory — memory operations, *continued*
 - optimizing usage of user mapped memory —
madvise, 3-552
 - memory allocator — bsdmalloc, 3X-126
 - alloca, 3C-558
 - calloc, 3C-558, 3X-560, 3X-563
 - free, 3C-558, 3X-560, 3X-563, 3X-126
 - mallinfo, 3X-560
 - malloc, 3C-558, 3X-560, 3X-563, 3X-126
 - mallopt, 3X-560
 - memalign, 3C-558
 - realloc, 3C-558, 3X-560, 3X-563, 3X-126
 - valloc, 3C-558
 - memory lock or unlock
 - calling process — plock, 3C-699
 - memory management — mctl, 3B-573
 - copy a file into memory — copylist, 3G-145
 - get system page size — getpagesize, 3B-424
 - lock address space — mlockall, 3C-608
 - lock pages in memory — mlock, 3C-606
 - synchronize memory with physical storage —
msync, 3C-624
 - unlock address space — munlockall, 3C-608
 - unlock pages in memory — munlock, 3C-606
 - memory object, shared
 - open — shm_open, 3R-846
 - remove — shm_unlink, 3R-848
 - memory operations
 - memccpy, 3C-575
 - memchr, 3C-575
 - memcmp, 3C-575
 - memcpy, 3C-575
 - memmove, 3C-575
 - memory, 3C-575
 - memset, 3C-575
 - menu library, See also curses library
 - menus — character based menus package, 3X-598
 - menus cursor
 - menu_cursor, 3X-577
 - pos_menu_cursor, 3X-577
 - menus display attributes
 - menu_attributes, 3X-576
 - menu_back, 3X-576
 - menu_fore, 3X-576
 - menus display attributes, *continued*
 - menu_grey, 3X-576
 - menu_pad, 3X-576
 - set_menu_back, 3X-576
 - set_menu_fore, 3X-576
 - set_menu_grey, 3X-576
 - set_menu_pad, 3X-576
 - menus from associated subwindows, write/erase
 - menu_post, 3X-595
 - post_menu, 3X-595
 - unpost_menu, 3X-595
 - menus item name and description
 - item_description, 3X-584
 - item_name, 3X-584
 - menu_item_name, 3X-584
 - menus item options routines
 - item_opts, 3X-586
 - item_opts_off, 3X-586
 - item_opts_on, 3X-586
 - menu_item_opts, 3X-586
 - set_item_opts, 3X-586
 - menus item values, set and get
 - item_value, 3X-588
 - menu_item_value, 3X-588
 - set_item_value, 3X-588
 - menus item, visibility
 - item_visible, 3X-589
 - menu_item_visible, 3X-589
 - menus items, associate application data
 - item_userptr, 3X-587
 - menu_item_userptr, 3X-587
 - set_item_userptr, 3X-587
 - menus items, connect and disconnect
 - item_count, 3X-590
 - menu_items, 3X-590
 - set_menu_items, 3X-590
 - menus items, create and destroy
 - free_item, 3X-585
 - menu_item_new, 3X-585
 - new_item, 3X-585
 - menus items, get and set
 - current_item, 3X-583
 - item_index, 3X-583
 - menu_item_current, 3X-583
 - set_current_item, 3X-583

menus items, get and set, *continued*
 — `set_top_row`, 3X-583
 — `top_row`, 3X-583

menus mark string routines
 — `menu_mark`, 3X-591
 — `set_menu_mark`, 3X-591

menus options routines
 — `menu_opts`, 3X-593
 — `menu_opts_off`, 3X-593
 — `menu_opts_on`, 3X-593
 — `set_menu_opts`, 3X-593

menus pattern match buffer
 — `menu_pattern`, 3X-594
 — `set_menu_pattern`, 3X-594

menus subsystem, command processor
 — `menu_driver`, 3X-578

menus window and subwindow association routines
 — `menu_sub`, 3X-597
 — `menu_win`, 3X-597
 — `scale_menu`, 3X-597
 — `set_menu_sub`, 3X-597
 — `set_menu_win`, 3X-597

menus, application-specific routines
 — `item_init`, 3X-581
 — `item_term`, 3X-581
 — `menu_hook`, 3X-581
 — `menu_init`, 3X-581
 — `menu_term`, 3X-581
 — `set_item_init`, 3X-581
 — `set_item_term`, 3X-581
 — `set_menu_init`, 3X-581
 — `set_menu_term`, 3X-581

menus, associate application data
 — `menu_userptr`, 3X-596
 — `set_menu_userptr`, 3X-596

menus, create and destroy
 — `free_menu`, 3X-592
 — `menu_new`, 3X-592
 — `new_menu`, 3X-592

menus, rows and columns
 — `menu_format`, 3X-580
 — `set_menu_format`, 3X-580

message catalog
 — *message catalog, continued*
 `open/catalog` — `catopen`, `catclose`, 3C-134
 read a program message — `catgets`, 3C-133

message handling functions
 — `bindtextdomain`, 3I-460
 — `dcgettext`, 3I-460
 — `dgettext`, 3I-460
 — `gettext`, 3I-460
 — `textdomain`, 3I-460

message queue
 `close` — `mq_close`, 3R-611
 get attributes — `mq_getattr`, 3R-621
 notify process (or thread) — `mq_notify`, 3R-612
 `open` — `mq_open`, 3R-614
 receive a message from — `mq_receive`, 3R-617
 remove — `mq_unlink`, 3R-623
 send message to — `mq_send`, 3R-619
 set attributes — `mq_setattr`, 3R-621

messages
 display a message on `stderr` or system console — `fmtmsg`, 3C-328
 print system error messages — `perror`, 3C-695
 system signal messages — `psignal`, 3C-712

`mkdirp` — create directories in a path, 3G-601
`mkfifo` — create a new FIFO, 3C-602
`mktemp` — make a unique file name, 3C-603
`mktime` — converts a `tm` structure to a calendar time, 3C-604

mnttab file
 — `getmntany`, 3C-410
 — `getmntent`, 3C-410
 — `hasmntopt`, 3C-410
 — `putmntent`, 3C-410

`monitor` — prepare process execution profile, 3C-609

`movenextch` — moving the cursor by character, 3X-163

`moveprevch` — moving the cursor by character, 3X-163

`mq_close` — close a message queue, 3R-611

mq_getattr — set/get message queue attributes, 3R-621
mq_notify — notify process (or thread) that a message is available on a queue, 3R-612
mq_open — open a message queue, 3R-614
mq_receive — receive a message from a message queue, 3R-617
mq_send — send a message to a message queue, 3R-619
mq_setattr — set/get message queue attributes, 3R-621
mq_unlink — remove a message queue, 3R-623
msync — synchronize memory with physical storage, 3C-624
multibyte character handling
 — **mbchar**, 3C-570
 — **mblen**, 3C-570
 — **mbtowc**, 3C-570
 — **wctomb**, 3C-570
multibyte string functions — **mbstring**, 3C-572
 mbstowcs, 3C-572
 wcstombs, 3C-572
mutex_destroy() — destroy a mutual exclusion lock, 3T-626
mutex_init() — initialize a mutual exclusion lock, 3T-626
mutex_lock() — acquire a mutual exclusion lock, 3T-626
mutex_trylock() — acquire a mutual exclusion lock, 3T-626
mutex_unlock() — release a mutual exclusion lock, 3T-626
mvaddnwstr — add a string of `wchar_t` characters to a curses window and advance cursor, 3X-162
mvaddwch — add a `wchar_t` character (with attributes) to a curses window and advance cursor, 3X-159
mvaddwchnstr — add string of `wchar_t` characters (and attributes) to a curses window, 3X-161
mvaddwchstr — add string of `wchar_t` characters (and attributes) to a curses window, 3X-161
mvaddwstr — add a string of `wchar_t` characters to a curses window and advance cursor, 3X-162
mvgetnwstr — get `wchar_t` character strings from curses terminal keyboard, 3X-184
mvgetwch — get (or push back) `wchar_t` characters from curses terminal keyboard, 3X-180
mvgetwstr — get `wchar_t` character strings from curses terminal keyboard, 3X-184
mvinnwstr — get a string of `wchar_t` characters from a curses window, 3X-200
mvinsnwstr — insert `wchar_t` string before character under the cursor in a curses window, 3X-197
mvinswch — insert a `wchar_t` character before the character under the cursor in a curses window, 3X-196
mvinswstr — insert `wchar_t` string before character under the cursor in a curses window, 3X-197
mvinwch — get a `wchar_t` character and its attributes from a curses window, 3X-198
mvinwchnstr — get a string of `wchar_t` characters (and attributes) from a curses window, 3X-199
mvinwchstr — get a string of `wchar_t` characters (and attributes) from a curses window, 3X-199
mvinwstr — get a string of `wchar_t` characters from a curses window, 3X-200
mvwaddnwstr — add a string of `wchar_t` characters to a curses window and advance cursor, 3X-162
mvwaddwch — add a `wchar_t` character (with attributes) to a curses window and advance cursor, 3X-159
mvwaddwchnstr — add string of `wchar_t` characters (and attributes) to a curses window, 3X-161
mvwaddwchstr — add string of `wchar_t` characters (and attributes) to a curses window, 3X-161
mvwaddwstr — add a string of `wchar_t` characters to a curses window and advance cursor, 3X-162
mvwgetnwstr — get `wchar_t` character strings from curses terminal keyboard, 3X-184
mvwgetwch — get (or push back) `wchar_t` characters from curses terminal keyboard, 3X-180
mvwgetwstr — get `wchar_t` character strings from

curses terminal keyboard, 3X-184
 mvwinwstr — get a string of wchar_t characters from a curses window, 3X-200
 mvwinswstr — insert wchar_t string before character under the cursor in a curses window, 3X-197
 mvwinswch — insert a wchar_t character before the character under the cursor in a curses window, 3X-196
 mvwinswstr — insert wchar_t string before character under the cursor in a curses window, 3X-197
 mvwinwch — get a wchar_t character and its attributes from a curses window, 3X-198
 mvwinwchnstr — get a string of wchar_t characters (and attributes) from a curses window, 3X-199
 mvwinwchstr — get a string of wchar_t characters (and attributes) from a curses window, 3X-199
 mvwinwstr — get a string of wchar_t characters from a curses window, 3X-200

N

named pipe
 create a new one — mkfifo, 3C-602
 nanosleep — high resolution sleep, 3R-629
natural logarithm — log, 3M-313
natural logarithm of 1 plus argument — log1p, 3M-313
 ndbm — database subroutines, 3-630
 netdir — generic transport name-to-address translation, 3N-633
 netdir_free — generic transport name-to-address translation, 3N-633
 netdir_getbyaddr — generic transport name-to-address translation, 3N-633
 netdir_getbyname — generic transport name-to-address translation, 3N-633
 netdir_mergeaddr — generic transport name-to-address translation, 3N-633
 netdir_options — generic transport name-to-address translation, 3N-633

netdir_perror — generic transport name-to-address translation, 3N-633
 netdir_sperror — generic transport name-to-address translation, 3N-633
network configuration database entry
 — endnetconfig, 3N-415
 — freenetconfig, 3N-415
 — getnetconfig, 3N-415
 — getnetconfig, 3N-415
 — nc_perror, 3N-415
 — nc_sperror, 3N-415
 — setnetconfig, 3N-415
network configuration entry corresponding to NET-PATH
 — endnetpath, 3N-419
 — getnetpath, 3N-419
 — setnetpath, 3N-419
network entry
 — endnetent, 3N-412
 — getnetbyaddr, 3N-412
 — getnetbyaddr_r, 3N-412
 — getnetbyname, 3N-412
 — getnetbyname_r, 3N-412
 — getnetent, 3N-412
 — getnetent_r, 3N-412
 — setnetent, 3N-412
network group entry
 — endnetgrent, 3N-417
 — getnetgrent, 3N-417
 — getnetgrent_r, 3N-417
 — innetgr, 3N-417
 — setnetgrent, 3N-417
network host entry
 — endhostent, 3N-401
 — gethostbyaddr, 3N-401
 — gethostbyaddr_r, 3N-401
 — gethostbyname, 3N-401
 — gethostbyname_r, 3N-401
 — gethostent, 3N-401
 — gethostent_r, 3N-401
 — sethostent, 3N-401
network listener service
 format and send listener service request message — nlsrequest, 3N-678

network listener service, *continued*

- get client's data passed via the listener —
 - nlsgetcall, 3N-676
- get name of transport provider — nlsprovider, 3N-677

network protocol entry

- endprotoent, 3N-429
- getprotobyname, 3N-429
- getprotobyname_r, 3N-429
- getprotobynumber, 3N-429
- getprotobynumber_r, 3N-429
- getprotoent, 3N-429
- getprotoent_r, 3N-429
- setprotoent, 3N-429

newpad — create and display curses pads, 3X-207

nextafter() function, 3M-489

nextkey — data base subroutines, 3B-240

nftw — walk a file tree, 3C-376

nice — change priority of a process, 3B-637

NIS client interface

- yp_all, 3N-1051
- yp_bind, 3N-1051
- yp_first, 3N-1051
- yp_get_default_domain, 3N-1051
- yp_master, 3N-1051
- yp_match, 3N-1051
- yp_next, 3N-1051
- yp_order, 3N-1051
- yp_unbind, 3N-1051
- ypclnt, 3N-1051
- yperr_string, 3N-1051
- ypprot_err, 3N-1051

NIS+ database functions

- db_add_entry, 3N-638
- db_checkpoint, 3N-638
- db_create_table, 3N-638
- db_destroy_table, 3N-638
- db_first_entry, 3N-638
- db_free_result, 3N-638
- db_initialize, 3N-638
- db_list_entries, 3N-638
- db_next_entry, 3N-638
- db_remove_entry, 3N-638
- db_reset_next_entry, 3N-638
- db_standby, 3N-638

NIS+ database functions, *continued*

- db_table_exists, 3N-638
- db_unload_table, 3N-638
- nis_db, 3N-638

NIS+ error messages

- nis_error, 3N-642
- nis_lerror, 3N-642
- nis_perror, 3N-642
- nis_sperrno, 3N-642
- nis_sperror, 3N-642
- nis_sperror_r, 3N-642

NIS+ group manipulation functions

- nis_addmember, 3N-643
- nis_creategroup, 3N-643
- nis_destroygroup, 3N-643
- nis_groups, 3N-643
- nis_ismember, 3N-643
- nis_groups, 3N-643
- nis_removemember, 3N-643
- nis_verifygroup, 3N-643

NIS+ local names

- nis_freenames, 3N-664
- nis_getnames, 3N-664
- nis_local_directory, 3N-646
- nis_local_group, 3N-646
- nis_local_host, 3N-646
- nis_local_names, 3N-646
- nis_local_principal, 3N-646

NIS+ log administration functions

- nis_checkpoint, 3N-661
- nis_ping, 3N-661

NIS+ miscellaneous functions

- nis_freeservelist, 3N-662
- nis_freetags, 3N-662
- nis_getservlist, 3N-662
- nis_mkdir, 3N-662
- nis_rmdir, 3N-662
- nis_server, 3N-662
- nis_servstate, 3N-662
- nis_stats, 3N-662

NIS+ namespace functions

- nis_add, 3N-647
- nis_freeresult, 3N-647
- nis_lookup, 3N-647
- nis_modify, 3N-647

NIS+ namespace functions, *continued*

- `nis_names`, 3N-647
- `nis_remove`, 3N-647

NIS+ object formats

- `nis_objects`, 3N-652

NIS+ subroutines

- `nis_clone_object`, 3N-664
- `nis_destroy_object`, 3N-664
- `nis_dir_cmp`, 3N-664
- `nis_domain_of`, 3N-664
- `nis_leaf_of`, 3N-664
- `nis_name_of`, 3N-664
- `nis_print_object`, 3N-664
- `nis_subr`, 3N-664

NIS+ table functions

- `nis_add_entry`, 3N-666
- `nis_first_entry`, 3N-666
- `nis_list`, 3N-666
- `nis_modify_entry`, 3N-666
- `nis_next_entry`, 3N-666
- `nis_remove_entry`, 3N-666
- `nis_tables`, 3N-666

NIS, change information

- `yp_update`, 3N-1050

`nis_tables` — NIS+ table functions, 3N-666

`nis_tables` — NIS+ table functions, 3N-666

`nis_tables` — NIS+ table functions, 3N-666

`nlist` — get entries from symbol table, 3B-674

`non-local goto` — `setjmp`, 3B-834, 3C-837

- `_longjmp`, 3B-834
- `_setjmp`, 3B-834
- `longjmp`, 3B-834, 3C-837
- `siglongjmp`, 3C-837
- `sigsetjmp`, 3C-837

`numbers`, convert to strings — `econvert`, 3-264

O

`offsetof` — offset of structure member, 3C-680

`openlog` — initialize system log file, 3-916

output conversion

- `wsprintf` — convert to `wchar_t` string, 3I-1034

output conversion, formatted

- `fprintf`, 3B-702
- `printf`, 3B-702

output conversion, formatted, *continued*

- `sprintf`, 3B-702
- `vfprintf`, 3B-702
- `vprintf`, 3B-702
- `vsprintf`, 3B-702

P

`p2close` — close pipes to and from a command, 3G-681

`p2open` — open pipes to and from a command, 3G-681

page size, system

- `get` — `getpagesize`, 3B-424

panel library, See also `curses` library

`panels` — character based panels package, 3X-691

`panels` deck manipulation routines

- `bottom_panel`, 3X-687
- `hide_panel`, 3X-686
- `panel_hidden`, 3X-686
- `panel_show`, 3X-686
- `panel_top`, 3X-687
- `show_panel`, 3X-686
- `top_panel`, 3X-687

`panels` deck traversal primitives

- `panel_above`, 3X-683
- `panel_below`, 3X-683

`panels` panel, associate application data

- `panel_userptr`, 3X-689
- `set_panel_userptr`, 3X-689

`panels` panel, get or set current window

- `panel_window`, 3X-690
- `replace_panel`, 3X-690

`panels` virtual screen refresh routine

- `panel_update`, 3X-688
- `update_panel`, 3X-688

`panels` window on virtual screen, move

- `move_panel`, 3X-684
- `panel_move`, 3X-684

`panels`, create and destroy

- `del_panel`, 3X-685
- `new_panel`, 3X-685
- `panel_new`, 3X-685

password

- `getpass`, 3C-425

password databases
 lock the lock file — `lckpwn`, 3C-534
 unlock the lock file — `ulckpwn`, 3C-534

password encryption
 — `crypt`, 3C-146
 — `encrypt`, 3C-146
 — `setkey`, 3C-146

passwords
 get passwd entry from UID — `getpw`, 3C-433
 get password entry from a file — `fgetpwent`,
 3C-434
 write password file entry — `putpwent`,
 3C-717

passwords, shadow
 get shadow password database entry —
 `endspent`, 3C-453
 get shadow password database entry (reentr-
 ant) — `fgetspent_r`, 3C-453
 write shadow password file entry —
 `putspent`, 3C-719

path name
 return last element — `path name`, 3G-121

`pathfind` — search for named file in named direc-
 tories, 3G-693

`pclose` — initiate pipe to/from a process, 3S-700

`pechochar` — create and display curses pads,
 3X-207

`pechowchar` — create and display curses pads,
 3X-207

`perror` — print system error messages, 3C-695

`pfmt` — display error message in standard format,
 3C-696

pipes
 initiate to/from a process — `pclose`, 3S-700
 open, close to and from a command —
 `p2open`, `p2close`, 3G-681

`plock` — lock or unlock into memory process, text,
 or data, 3C-699

`pnoutrefresh` — create and display curses pads,
 3X-207

`popen` — initiate pipe to/from a process, 3S-700

`pow` — raise to power, 3M-313

power function `pow` — `pow`, 3M-313

`prefresh` — create and display curses pads,
 3X-207

print formatted output
 — `fprintf`, 3S-706
 — `printf`, 3S-706
 — `sprintf`, 3S-706

print formatted output of a variable argument list
 — `vfprintf`, 3S-1012
 — `vprintf`, 3S-1012
 — `vsprintf`, 3S-1012

`printf` — formatted output conversion, 3B-702,
 3S-706

Process Code character classification macros and
 functions
 — `isenglish`, 3I-500
 — `isideogram`, 3I-500
 — `isnumber`, 3I-500
 — `isphonogram`, 3I-500
 — `isspecial`, 3I-500
 — `iswalnum`, 3I-500
 — `iswalpha`, 3I-500
 — `iswascii`, 3I-500
 — `iswcntrl`, 3I-500
 — `iswdigit`, 3I-500
 — `iswgraph`, 3I-500
 — `iswlower`, 3I-500
 — `iswprint`, 3I-500
 — `iswpunct`, 3I-500
 — `iswspace`, 3I-500
 — `iswupper`, 3I-500
 — `iswxdigit`, 3I-500

Process Code character conversion macros
 — `towlower`, 3I-1019
 — `toupper`, 3I-1019
 — `wconv`, 3I-1019

Process Code string operations — `wstring`,
 3I-1036
 `wscasecmp`, 3I-1036
 `wscol`, 3I-1036
 `wsdup`, 3I-1036
 `wncasecmp`, 3I-1036

process statistics
 prepare execution profile — `monitor`, 3C-609

processes

processes, *continued*

- change priority — nice, 3B-637
- duplicate an open file descriptor — dup2, 3C-263
- generate path name for controlling terminal — ctermid, ctermid_r, 3S-148
- get character-string representation — cuserid, 3S-239
- initiate pipe to/from a process — popen, pclose, 3S-700
- manipulate user contexts — makecontext, swapcontext, 3C-556
- memory lock or unlock — plock, 3C-699
- prepare execution profile — monitor, 3C-609
- report CPU time used — clock, 3C-136
- send signal to a process group — killpg, 3B-510
- send signal to program — raise, 3C-724
- set terminal foreground process group id — tcsetpgrp, 3C-966
- suspend execution for interval — sleep, 3B-869, 3C-870
- terminate process — exit, 3C-312
- terminate the process abnormally — abort, 3C-90
- wait for process to terminate or stop — WIFEXITED, 3B-1015

profiling utilities

- prepare process execution profile — monitor, 3C-609

program assertion

- verify — assert, 3X-112

program messages

- open/close a message catalog — catopen, catclose, 3C-134
- read — catgets, 3C-133

programs

- last locations — end, etext, edata, 3C-307

pseudo-terminal device

- get name of the slave pseudo-terminal device — ptsname, 3C-713
- grant access to the slave pseudo-terminal device — grantpt, 3C-479

psiginfo — system signal messages, 3C-712

psignal — system signal messages, 3B-711, 3C-712

ptsname — get name of the slave pseudo-terminal device, 3C-713

publickey — retrieve public or secret key, 3N-432

putenv — change or add value to environment, 3C-716

putmntent — get mnttab file information, 3C-410

putpwent — write password file entry, 3C-717

putspent — write shadow password file entry, 3C-719

pututline — access utmp file entry, 3C-467

pututxline — access utmpx file entry, 3C-469

putwc — put wide character on stream, 3I-720

putwchar — put wide character on stdout, 3I-720

putws — convert a string of Process Code characters to EUC characters and put it on a stream, 3I-721

Q

qeconvert — convert number to ASCII, 3-264

qfconvert — convert number to ASCII, 3-264

qgconvert — convert number to ASCII, 3-264

qsort — quick sort, 3C-722

quadruple_to_decimal — decimal record from quadruple-precision floating, 3-321

queues

- insert/remove element from a queue — insque, remque, 3C-496

R

rac_drop() — remote asynchronous calls, 3N-766

rac_poll() — remote asynchronous calls, 3N-766

rac_recv() — remote asynchronous calls, 3N-766

rac_send() — remote asynchronous calls, 3N-766

raise — send signal to program, 3C-724

rand — simple random number generator, 3B-725, 3C-726

random — random number generator, 3B-727

random number generator

- changing generators — initstate, setstate, 3B-727
- drand48, 3C-261
- erand48, 3C-261

random number generator, *continued*
 — jrand48, 3C-261
 — lcong48, 3C-261
 — lrand48, 3C-261
 — mrand48, 3C-261
 — nrand48, 3C-261
 — rand, 3B-725
 — random, 3B-727
 — seed48, 3C-261
 — srand48, 3C-261
 random number generator, simple
 — rand, 3C-726
 — srand, 3C-726
 rcmd — execute command remotely, 3N-729
 read and write a disk's VTOC — read_vtoc, 3X-731
 write_vtoc, 3X-731
 read or write kstat data
 — kstat_read, 3K-526
 — kstat_write, 3K-526
 read system process structures
 — kvm_getproc, 3K-529
 — kvm_nextproc, 3K-529
 — kvm_setproc, 3K-529
 read_vtoc — read and write a disk's VTOC, 3X-731
 real and effective user IDs
 set — setreuid, 3B-844
 realloc — memory allocator, 3X-126
 realpath — returns the real file name, 3C-732
 reboot — reboot system or halt processor, 3B-733
 receive a message from a socket — recv, 3N-734
 recvfrom, 3N-734
 recvmsg, 3N-734
 recv — receive a message from a socket, 3N-734
 recvfrom — receive a message from a socket, 3N-734
 recvmsg — receive a message from a socket, 3N-734
 regcmp — compile regular expression, 3G-736
 regex — execute regular expression, 3G-736
 regexpr — regular expression compile and match routines, 3G-739
 regular expression compile and match routines
 — advance, 3G-739
 — compile, 3G-739
 — regexpr, 3G-739
 — step, 3G-739
 regular expression handler
 — re_comp, 3B-738
 — re_exec, 3B-738
 — regex, 3B-738
 regular expressions
 compile and execute — regcmp, regex, 3G-736
 remainder() function, 3M-489
 remote command, return stream to
 — rcmd, 3N-729
 remote procedure calls, library routines for — rpc, 3N-746
 remote system
 return information about users — rusers, rnusers, 3N-794
 write to — rstat, 3N-793, 3N-795
 remove — remove file, 3C-742
 remque — remove element from a queue, 3C-496
 res_init — resolver routines, 3N-743
 res_mkquery — resolver routines, 3N-743
 res_send — resolver routines, 3N-743
 resolver — resolver routines, 3N-743
 resolver routines — resolver, 3N-743
 dn_comp, 3N-743
 dn_expand, 3N-743
 res_init, 3N-743
 res_mkquery, 3N-743
 res_send, 3N-743
 resource utilization
 get information — getrusage, 3B-441
 retrieve archive symbol table — elf_getarsym, 3E-289
 retrieve public or secret key — getpublickey, 3N-432
 getsecretkey, 3N-432
 publickey, 3N-432
 return physical memory information — system, 3-919

return physical memory information — `systemem`,
continued
`asystemem`, 3-919

return stream to a remote command — `rexec`,
 3N-745

`rexec` — return stream to a remote command,
 3N-745

`rindex` — string operations, 3B-492

`rint` — round to integral value, 3M-327

`rmdirp` — remove directories in a path, 3G-601

`rnusers` — return information about users on
 remote machines, 3N-794

`rpc` — library routines for remote procedure calls,
 3N-746

RPC
 data transmission using XDR routines — `xdr`,
 3N-1037

RPC bind service library routines
 — `rpc_getmaps`, 3N-791
 — `rpcb_getaddr`, 3N-791
 — `rpcb_gettime`, 3N-791
 — `rpcb_rmtcall`, 3N-791
 — `rpcb_set`, 3N-791
 — `rpcb_unset`, 3N-791
 — `rpcbind`, 3N-791

RPC entry
 — `endrpccent`, 3N-438
 — `getrpcbyname`, 3N-438
 — `getrpcbyname_r`, 3N-438
 — `getrpcbynumber`, 3N-438
 — `getrpcbynumber_r`, 3N-438
 — `getrpccent`, 3N-438
 — `getrpccent_r`, 3N-438
 — `setrpccent`, 3N-438

RPC library routine for manipulating global RPC
 attributes for client and server applications
 — `rpc_control`, 3N-764

RPC library routines for creation and manipulation
 of server handles
 — `rpc_svc_create`, 3N-782
 — `svc_create`, 3N-782
 — `svc_destroy`, 3N-782
 — `svc_dg_create`, 3N-782
 — `svc_fd_create`, 3N-782

RPC library routines for creation and manipulation
 of server handles, *continued*
 — `svc_raw_create`, 3N-782
 — `svc_tli_create`, 3N-782
 — `svc_tp_create`, 3N-782
 — `svc_vc_create`, 3N-782

RPC library routines for registering servers
 — `rpc_reg`, 3N-787
 — `rpc_svc_reg`, 3N-787
 — `svc_auth_reg`, 3N-787
 — `svc_reg`, 3N-787
 — `svc_unreg`, 3N-787
 — `xprt_register`, 3N-787
 — `xprt_unregister`, 3N-787

RPC library routines for server side errors
 — `rpc_svc_err`, 3N-785
 — `svcerr_auth`, 3N-785
 — `svcerr_decode`, 3N-785
 — `svcerr_noproc`, 3N-785
 — `svcerr_noprog`, 3N-785
 — `svcerr_progvers`, 3N-785
 — `svcerr_systemerr`, 3N-785
 — `svcerr_weakauth`, 3N-785

RPC obsolete library routines
 — `authdes_create`, 3N-769
 — `authunix_create_default`, 3N-769
 — `callrpc`, 3N-769
 — `clnt_broadcast`, 3N-769
 — `clntraw_create`, 3N-769
 — `clnttcp_create`, 3N-769
 — `clntudp_bufcreate`, 3N-769
 — `clntudp_create`, 3N-769
 — `get_myaddress`, 3N-769
 — `pmap_getmaps`, 3N-769
 — `pmap_getport`, 3N-769
 — `pmap_rmtcall`, 3N-769
 — `pmap_set`, 3N-769
 — `pmap_unset`, 3N-769
 — `registerrpc`, 3N-769
 — `rpc_soc`, 3N-769
 — `svc_fds`, 3N-769
 — `svc_getcaller`, 3N-769
 — `svc_getreq`, 3N-769
 — `svc_register`, 3N-769
 — `svc_unregister`, 3N-769

RPC obsolete library routines, *continued*

- `svcf_create`, 3N-769
- `svccraw_create`, 3N-769
- `svctcp_create`, 3N-769
- `svcudp_bufcreate`, 3N-769
- `svcudp_create`, 3N-769
- `xdr_authunix_parms`, 3N-769

rpc routines

- `rac_drop()` — remote asynchronous calls, 3N-766
- `rac_poll()` — remote asynchronous calls, 3N-766
- `rac_recv()` — remote asynchronous calls, 3N-766
- `rac_send()` — remote asynchronous calls, 3N-766

RPC servers library routines

- `rpc_svc_calls`, 3N-778
- `svc_dg_enablecache`, 3N-778
- `svc_done`, 3N-778
- `svc_exit`, 3N-778
- `svc_fdset`, 3N-778
- `svc_freeargs`, 3N-778
- `svc_getargs`, 3N-778
- `svc_getreq_common`, 3N-778
- `svc_getreq_poll`, 3N-778
- `svc_getreqset`, 3N-778
- `svc_getrpccaller`, 3N-778
- `svc_pollset`, 3N-778
- `svc_run`, 3N-778
- `svc_sendreply`, 3N-778

RPC using Kerberos authentication routines

- `authkerb_getucred`, 3N-507
- `authkerb_seccreate`, 3N-507
- `kerberos_rpc`, 3N-507
- `svc_kerb_reg`, 3N-507

RPC, secure library routines

- `authdes_getucred`, 3N-810
- `authdes_seccreate`, 3N-810
- `getnetname`, 3N-810
- `host2netname`, 3N-810
- `key_decryptsession`, 3N-810
- `key_encryptsession`, 3N-810
- `key_gendes`, 3N-810
- `key_secretkey_is_set`, 3N-810

RPC, secure library routines, *continued*

- `key_setsecret`, 3N-810
- `netname2host`, 3N-810
- `netname2user`, 3N-810
- `secure_rpc`, 3N-810
- `user2netname`, 3N-810

RPC, XDR library routines

- `rpc_xdr`, 3N-789
- `xdr_accepted_reply`, 3N-789
- `xdr_authsys_parms`, 3N-789
- `xdr_callhdr`, 3N-789
- `xdr_callmsg`, 3N-789
- `xdr_opaque_auth`, 3N-789
- `xdr_rejected_reply`, 3N-789
- `xdr_replymsg`, 3N-789

`rpc_broadcast` — library routines for client side calls, 3N-756

`rpc_broadcast_exp` — library routines for client side calls, 3N-756

`rpc_call` — library routines for client side calls, 3N-756

`rpc_clnt_auth` — library routines for client side remote procedure call authentication, 3N-754

`rpc_clnt_calls` — library routines for client side calls, 3N-756
Routines, 3N-756

`rpc_clnt_create` — library routines for dealing with creation and manipulation of CLIENT handles, 3N-759
Routines, 3N-759

`rpc_createerr` — library routines for dealing with creation and manipulation of CLIENT handles, 3N-759

`rresvport` — get privileged socket, 3N-729

`rstat` — get performance data from remote kernel, 3N-793

`ruserok` — authenticate user, 3N-729

`rusers` — return information about users on remote machines, 3N-794
`xdr_utmpidlearr`, 3N-794

`rw_rdlock()` — acquire a read lock, 3T-796

`rw_tryrdlock()` — acquire a read lock, 3T-796

`rw_trywrlock()` — acquire a write lock, 3T-796

`rw_unlock()` — unlock a readers/writer lock, 3T-796
`rw_wrlock()` — acquire a write lock, 3T-796
`rwall` — write to specified remote machines, 3N-795
`rwlock_destroy()` — destroy a readers/writer lock, 3T-796
`rwlock_init()` — initialize a readers/writer lock, 3T-796

S

`scalb()` function, 3M-491
`scalbn()` function, 3M-489
scan a directory

- `alphasort`, 3B-798
- `scandir`, 3B-798

`scandir` — scan a directory, 3B-798
`scanf` — convert formatted input, 3S-799
`sched_get_priority_max` — get scheduling parameter limits, 3R-804
`sched_get_priority_min` — get scheduling parameter limits, 3R-804
`sched_getparam` — get scheduling parameters, 3R-805
`sched_getscheduler` — get scheduling policy and scheduling parameters, 3R-807
`sched_rr_get_interval` — get scheduling parameter limits, 3R-804
`sched_setparam` — set scheduling parameters, 3R-805
`sched_setscheduler` — set scheduling policy and scheduling parameters, 3R-807
`sched_yield` — yield processor, 3R-809
scheduling parameters

- set/get — `sched_setparam`, `sched_getparam`, 3R-805

scheduling parameters list

- `sched_get_priority_max`, 3R-804
- `sched_get_priority_min`, 3R-804
- `sched_rr_get_interval`, 3R-804

scheduling policy and parameters

- set/get — `sched_setscheduler`,

- `sched_getscheduler`,

scheduling policy and parameters, *continued* 3R-807
scheduling priority

- change priority of a process — `nice`, 3B-637
- get/set for process, process group or user
 - `getpriority`, `setpriority`, 3B-427

search functions

- linear search and update routine — `lsearch`, `lfind`, 3C-550
- manage hash search tables — `hsearch`, 3C-480

search routines

- binary search a sorted table — `bsearch`, 3C-128

`seconvert` — convert number to ASCII, 3-264
`secure`, RPC

- See `RPC,secure`, 3N-810

`select` — synchronous I/O multiplexing, 3C-814
`sem_close` — close a named semaphore, 3R-816
`sem_destroy` — destroy an unnamed semaphore, 3R-817
`sem_getvalue` — get the value of a semaphore, 3R-818
`sem_init` — initialize an unnamed semaphore, 3R-819
`sem_open` — initialize/open a named semaphore, 3R-820
`sem_post` — increment the count of a semaphore, 3R-822
`sem_trywait` — acquire or wait for a semaphore, 3R-824
`sem_unlink` — remove a named semaphore, 3R-823
`sem_wait` — acquire or wait for a semaphore, 3R-824
`sema_destroy()` — destroy a semaphore, 3T-827
`sema_init()` — initialize a semaphore, 3T-827
`sema_post()` — increment a semaphore, 3T-827
`sema_trywait()` — decrement a semaphore, 3T-827
`sema_wait()` — decrement a semaphore, 3T-827

semaphore

acquire or wait for — `sem_wait`,
`sem_trywait`, 3R-824

close a named one — `sem_close`, 3R-816

destroy an unnamed one — `sem_destroy`,
3R-817

get the value — `sem_getvalue`, 3R-818

increment the count — `sem_post`, 3R-822

initialize an unnamed one — `sem_init`,
3R-819

initialize/open a named one — `sem_open`,
3R-820

remove a named one — `sem_unlink`, 3R-823

send — send message from a socket, 3N-829

sendmsg — send message from a socket, 3N-829

sendto — send message from a socket, 3N-829

Service Access Facility library function
— `doconfig`, 3N-259

setac — get audit control file information, 3-378

setaclass — rewind `audit_class` database file,
3-380

setauser — rewind `audit_event` database file,
3-383

setauser — get `audit_user` database entry, 3-385

setcat — define default catalog, 3C-833

setgrent — get group entry from database,
3C-398

sethostname — set name of current host, 3B-407

setjmp — non-local goto, 3B-834, 3C-837

setlabel — define the label for `pfmt()` and `lfmt()`,
3C-840

setlocale — modify and query a program's
locale, 3C-841

setlogmask — set log priority mask, 3-916

setpriority — set scheduling priority for pro-
cess, process group or user, 3B-427

setpwnam — get password entry from user data-
base, 3C-434

setregid — set real and effective group IDs,
3B-843

setreuid — set real and effective user IDs, 3B-844

setservent — get service entry, 3N-445

setspent — get shadow password database entry,

3C-453

setstate — routines for changing random
number generators, 3B-727

settimeofday — set date and time, 3C-464,
3B-463

setutent — access `utmp` file entry, 3C-467

setutxent — access `utmpx` file entry, 3C-469

severity levels, applications
build a list for use with `fntmsg` — `addsever-`
`ity`, 3C-95

sfconvert — convert number to ASCII, 3-264

sgconvert — convert number to ASCII, 3-264

shared memory object
open — `shm_open`, 3R-846
remove — `shm_unlink`, 3R-848

shared object
close — `dlclose`, 3X-253
get address of symbol — `dlsym`, 3X-257
get diagnostic information — `dlerror`, 3X-254
open — `dlopen`, 3X-255

shell command
issue one — `system`, 3S-920

shell global pattern matching — `gmatch`, 3G-478

`shm_open` — open a shared memory object, 3R-846

`shm_unlink` — remove a shared memory object,
3R-848

shutdown — shut down part of a full-duplex con-
nection, 3N-849

`sig2str` — translation between signal name and
signal number, 3C-884

`sigaddset` — manipulate sets of signals, 3C-860

`sigdelset` — manipulate sets of signals, 3C-860

`sigemptyset` — manipulate sets of signals, 3C-860

`sigfillset` — manipulate sets of signals, 3C-860

`sigfpe()` function, 3-851

`sighold` — adds `sig` to the calling process's sig-
nal mask, 3C-857

`sigignore` — sets the disposition of `sig` to
`SIG_IGN`, 3C-857

`siginterrupt` — allow signals to interrupt func-
tions, 3B-854

`sigismember` — manipulate sets of signals, 3C-860

siglongjmp — non-local goto, 3C-837
signal — modify signal disposition, 3C-857, 3B-855
 queue one to a process — `sigqueue`, 3R-859
 schedule after interval in microseconds — `ualarm`, 3B-1003
 suspend execution for interval in microseconds — `usleep`, 3B-1007
 wait for queued signals — `sigwaitinfo`, `sigtimedwait`, 3R-867
signal management
 simplified, for application processes — `signal`, 3C-857
signal messages
 get error message string — `strsignal`, 3C-905
signal messages, system
 — `psignal`, 3B-711, 3C-712
signals, block
 — `sigblock`, 3B-850
 — `sigmask`, 3B-850
 — `sigpause`, 3B-850
 — `sigsetmask`, 3B-850
signals, software
 — `gsignal`, 3C-878
 — `ssignal`, 3C-878
significand() function, 3M-491
sigpause — removes `sig` from the calling process's signal mask and suspends the calling process until a signal is received, 3C-857
sigqueue — queue a signal to a process, 3R-859
sigrelse — removes `sig` from the calling process's signal mask, 3C-857
sigset — modify signal disposition, 3C-857
sigsetjmp — non-local goto, 3C-837
sigsetops — manipulate sets of signals, 3C-860
sigstack — set and/or get signal stack context, 3B-861
sigtimedwait — wait for queued signals, 3R-867
sigvec — software signal facilities, 3B-863
sigwaitinfo — wait for queued signals, 3R-867
sin — trigonometric sine, 3M-995
single_to_decimal — decimal record from single-precision floating, 3-321
sinh — hyperbolic sine, 3M-483
sleep — suspend execution for interval, 3B-869, 3C-870
 high resolution — `nanosleep`, 3R-629
 suspend execution for interval in microseconds — `usleep`, 3B-1007
socket — create an endpoint for communication, 3N-871
 accept a connection — `accept`, 3N-92
 bind a name — `bind`, 3N-124
 get name — `getsockname`, 3N-449
 get name of connected peer — `getpeername`, 3N-426
 get options — `getsockopt`, 3N-450
 initiate a connection — `connect`, 3N-142
 listen for connections — `listen`, 3N-543
 send message from — `send`, `sendto`, `sendmsg`, 3N-829
 set options — `setsockopt`, 3N-450
 shut down part of a full-duplex connection — `shutdown`, 3N-849
socketpair — create a pair of connected sockets, 3N-874
software signals
 — `gsignal`, 3C-878
 — `ssignal`, 3C-878
sort
 quick — `qsort`, 3C-722
sprintf — formatted output conversion, 3B-702
spray — scatter data in order to test the network, 3N-875
sprintf — print formatted output, 3S-706
sqrt — square root function, 3M-877
square root — `sqrt`, 3M-877
srand — reset simple random number generator, 3B-725
srandom — random number generator, 3B-727
sscanf — convert formatted input, 3S-799
stack space minimum for a thread — `thr_min_stack`, 3T-979
standend — curses character and window attribute control routines, 3X-164

standout — curses character and window attribute control routines, 3X-164

stdin
 convert EUC character from the stream to Process Code — `getwchar`, 3I-474
 push a Process Code character back into input stream — `ungetwc`, 3I-1005

stdio — standard buffered input/output package, 3S-879

stdout
 convert Process Code character to EUC — `putwchar`, 3I-720

step — regular expression compile and match routines, 3G-739

string collation
 — `strcoll`, 3C-886

store — data base subroutines, 3B-240

strfind — string manipulations, 3G-888

str2sig — translation between signal name and signal number, 3C-884

strcadd — copy strings, compressing or expanding C language escape codes, 3G-885

strccpy — copy strings, compressing or expanding C language escape codes, 3G-885

streadd — copy strings, compressing or expanding C language escape codes, 3G-885

stream
 convert a string of EUC characters from the stream to Process Code — `getws`, 3I-477
 convert a string of EUC characters from the stream to Process Code — `getwfs`, 3I-477
 convert a string of Process Code characters to EUC characters and put it on a stream — `fputws`, 3I-721
 convert EUC character from the stream to Process Code — `fgetwc`, 3I-474
 convert Process Code character to EUC — `fputwc`, 3I-720
 open — `fopen`, 3B-333
 push a Process Code character back into input stream — `ungetwc`, 3I-1005
 push character back onto input stream — `ungetc`,
 stream, *continued*
 3S-1004

stream status inquiries
 — `clearerr`, 3S-319
 — `feof`, 3S-319
 — `ferror`, 3S-319
 — `fileno`, 3S-319

stream, assign buffering
 — `setbuf`, 3S-831
 — `setvbuf`, 3S-831

stream, get character or word
 — `fgetc`, 3S-387
 — `getc`, 3S-387
 — `getc_unlocked`, 3S-387
 — `getchar`, 3S-387
 — `getchar_unlocked`, 3S-387
 — `getw`, 3S-387

stream, get string
 — `fgets`, 3S-444
 — `gets`, 3S-444

stream, put a string
 — `fputs`, 3S-718
 — `puts`, 3S-718

stream, put character or word
 — `fputc`, 3S-714
 — `putc`, 3S-714
 — `putc_unlocked`, 3S-714
 — `putchar`, 3S-714
 — `putchar_unlocked`, 3S-714
 — `putw`, 3S-714

STREAMS
 accept a connection on a socket — `accept`, 3N-92
 attach a STREAMS-based file descriptor to an object in the file system name space — `fattach`, 3C-314
 bind a name to a socket — `bind`, 3N-124
 buffered binary input/output — `fread`, 3S-368
 close a stream — `fclose`, 3S-316
 create a pair of connected sockets — `socketpair`, 3N-874
 create an endpoint for communication — `socket`, 3N-871

STREAMS, *continued*

- detach a name from a STREAMS-based file descriptor — `fdetach`, 3C-318
- determine whether a buffer of characters is encrypted — `isencrypt`, 3G-498
- flush a stream — `fflush`, 3S-316
- get and set socket options — `getsockopt`, `setsockopt`, 3N-450
- get name of peer connected to socket — `getpeername`, 3N-426
- get socket name — `getsockname`, 3N-449
- initiate a connection on a socket — `connect`, 3N-142
- listen for connections on a socket — `listen`, 3N-543
- open a stream with a file descriptor — `fdoopen`, 3S-335
- open and associate stream with file — `fopen`, 3S-335
- open stream with name of file — `freopen`, 3S-335
- read stream up to next delimiter — `bgets`, 3G-123
- reposition a file pointer — `fseek`, 3S-371
- return to remote command — `rcmd`, 3N-729
- send a message from a socket — `send`, `sendto`, `sendmsg`, 3N-829
- shut down part of a full-duplex connection — `shutdown`, 3N-849
- split buffer into fields — `bufsplit`, 3G-131
- test file descriptor for a STREAMS file — `isastream`, 3C-497
- `strecpy` — copy strings, compressing or expanding C language escape codes, 3G-885
- `strfind` — string manipulations, 3G-888
- `strfmon` — convert monetary value to string, 3C-889
- `strftime` — convert date and time to string, 3C-893
- string manipulations — `strfind`, 3G-888
 - `str`, 3G-888
 - `strfind`, 3G-888
 - `strrspn`, 3G-888
 - `strtrns`, 3G-888
- string operation

string operation, *continued*

- get error message string — `strerror`, 3C-887
- get TLI error message string — `t_strerror`, 3N-962

string operations

- bit and byte — `bstring`, 3B-130
 - `index`, 3B-492
 - `rindex`, 3B-492
 - `strcasecmp`, 3C-896
 - `strcat`, 3C-896
 - `strchr`, 3C-896
 - `strcmp`, 3C-896
 - `strcpy`, 3C-896
 - `strcspn`, 3C-896
 - `strdup`, 3C-896
 - `string`, 3C-896
 - `strlen`, 3C-896
 - `strncasecmp`, 3C-896
 - `strncat`, 3C-896
 - `strncmp`, 3C-896
 - `strncpy`, 3C-896
 - `strpbrk`, 3C-896
 - `strrchr`, 3C-896
 - `strspn`, 3C-896
 - `strstr`, 3C-896
 - `strtok`, 3C-896
 - `strtok`, 3C-896

string transformation

- `strxfrm`, 3C-909

string, convert to double-precision floating-point number

- `atof`, 3C-906
- `strtod`, 3C-906

string, convert to integer

- `atoi`, 3C-907
- `atol`, 3C-907
- `atol`, 3C-907
- `strtol`, 3C-907
- `strtol`, 3C-907
- `strtoul`, 3C-907
- `strtoul`, 3C-907

- `string_to_decimal` — decimal record from character string, 3-899

strings

- copy, compressing or expanding C language

- escape
- strings, *continued*
 - codes, 3G-885
- strings, convert from numbers — `econvert`, 3-264
- `strptime` — date and time conversion, 3C-902
- `strfind` — string manipulations, 3G-888
- `strfind` — string manipulations, 3G-888
- `subpad` — create and display curses pads, 3X-207
- `swab` — swap bytes, 3C-910
- `swapcontext` — manipulate user contexts, 3C-556
- symbol table
 - get entries — `nlist`, 3B-674
- synchronize a file's data
 - `fdatasync`, 3R-317
- synchronous I/O multiplexing
 - `select`, 3C-814
- `sys_siglist` — system signal messages list, 3B-711
- `syscall` — indirect system call, 3B-911
- `sysconf` — get configurable system variables, 3C-912
- `syslog` — write message to system log, 3-916
- `systemem` — return physical memory information, 3-919
- `system` — issue shell command, 3S-920
- system error messages
 - print — `perror`, 3C-695
- system log
 - log message with variable argument list — `vsyslog`, 3-1014
- system log, control — `syslog`, 3-916
- system signal messages
 - `psignal`, 3C-712
- system variables
 - get configurable ones — `sysconf`, 3C-912

T

- `t_accept` — accept a connect request, 3N-921
- `t_alloc` — allocate memory for argument structures, 3N-923
- `t_bind` — bind an address to a transport endpoint, 3N-925
- `t_close` — close a transport endpoint, 3N-928

- `t_connect` — establish a connection with another transport user, 3N-929
- `t_error` — produce error message, 3N-932
- `t_free` — free allocated memory, 3N-933
- `t_getinfo` — get protocol-specific service information, 3N-934
- `t_getstate` — get the current state, 3N-937
- `t_listen` — listen for a connect request, 3N-938
- `t_look` — look at the current event on a transport endpoint, 3N-940
- `t_open` — establish a transport endpoint, 3N-941
- `t_optmgmt` — manage options for a transport endpoint, 3N-942
- `t_rcv` — manage options for a transport endpoint, 3N-944
- `t_rcvconnect` — receive the confirmation from a connect request, 3N-946
- `t_rcvdis` — retrieve information from disconnect, 3N-948
- `t_rcvrel` — acknowledge receipt of an orderly release indication, 3N-950
- `t_rcvudata` — receive a data unit, 3N-951
- `t_rcvuderr` — receive a unit data error indication, 3N-953
- `t_snd` — send data or expedited data over a connection, 3N-955
- `t_snddis` — send user-initiated disconnect request, 3N-957
- `t_sndrel` — initiate an orderly release of connection, 3N-959
- `t_sndudata` — send a data unit, 3N-960
- `t_strerror` — get error message string, 3N-962
- `t_sync` — synchronize transport library, 3N-963
- `t_unbind` — disable a transport endpoint, 3N-965
- `taddr2uaddr` — generic transport name-to-address translation, 3N-633
- `tan` — trigonometric tangent, 3M-995
- `tanh` — hyperbolic tangent, 3M-483
- `tcdrain` — general terminal interface, 3-967, 3-968
- `tcflow` — general terminal interface, 3-967, 3-968
- `tcflush` — general terminal interface, 3-967, 3-968
- `tcgetattr` — general terminal interface, 3-967

tcgetpgrp — general terminal interface, 3-967, 3-969
 tcgetsid — general terminal interface, 3-967, 3-969
 tcsendbreak — general terminal interface, 3-967, 3-968
 tcsetattr — general terminal interface, 3-967
 tcsetpgrp — general terminal interface, 3-967, 3-969
 tempnam — create a name for a temporary file, 3S-993
 terminal
 find name — ttyname, 3C-1001
 find the slot in the utmp file of the current user — ttyslot, 3C-1002
 terminal attributes
 get — tcsetattr, 3-967
 set — tcsetattr, 3-967
 terminal device, slave pseudo
 get name — ptsname, 3C-713
 grant access — grantpt, 3C-479
 terminal ID
 generate path name for controlling terminal — ctermid, ctermid_r, 3S-148
 terminal interface
 baud rate — cfgetispeed, 3-968
 get terminal foreground process group — tcgetgrp, 3-969
 get terminal session — tcgetsid, 3-969
 line control — tcdrain, 3-968
 set terminal foreground process group — tcsetpgrp, 3-969
 terminal line
 establish an outgoing connection — dial, 3N-243
 terminals
 set terminal foreground process group id — tcsetpgrp, 3C-966
 termios — general terminal interface, 3-967
 Get and Set Baud Rate, 3-968
 Get and Set Terminal Attributes, 3-967
 Get and Set Terminal Foreground Process Group ID, 3-969
 Get Terminal Session ID, 3-969
 termios — general terminal interface, *continued*
 Line Control, 3-968
 test character for specified class — iswctype, 3I-502
 text processing utilities
 compile and execute regular expressions — regcmp, regex, 3G-736
 quick sort — qsort, 3C-722
 regular expression handler — regex, 3B-738
 text string
 — gettext, 3C-465
 textdomain — select domain of messages, 3I-460
 thr_continue — continue thread execution, 3T-984
 thr_create — create a thread of control, 3T-971
 thr_exit — thread termination, 3T-974
 thr_getconcurrency — get thread concurrency level, 3T-981
 thr_setprio — get thread priority, 3T-982
 thr_getspecific — get thread-specific data, 3T-976
 thr_join — wait for thread termination, 3T-975
 thr_keycreate — create a thread-specific data key, 3T-976
 thr_kill — send a signal to a thread, 3T-978
 thr_min_stack — minimum stack space for a thread, 3T-979
 thr_self — get thread identifier, 3T-980
 thr_setconcurrency — set thread concurrency level, 3T-981
 thr_setprio — set thread priority, 3T-982
 thr_setspecific — set thread-specific data, 3T-976
 thr_sigsetmask — change and/or examine calling thread's signal mask, 3T-983
 thr_suspend — suspend thread execution, 3T-984
 thr_yield — yield execution to another thread, 3T-985
 time
 computes the difference between two calendar times — difftime, 3C-245
 time accounting

time accounting, *continued*
 for current process — `times`, 3B-991

time and date
 convert to string — `asctime`, 3C-149
 convert user format date and time — `get-date`, 3C-390
`get` — `ftime`, 3B-375
 — `settimeofday`, 3C-464

time of day
`get` and `set` — `gettimeofday`, `settimeofday`, 3B-463

time, calendar
 convert from a `tm` structure — `mktime`, 3C-604

`timer_create` — create a per-process, per-LWP, or per-thread timer, 3R-986

`timer_delete` — delete a per-process, per-LWP, or per-thread timer, 3R-988

`timer_getoverrun` — high-resolution timer operations, 3R-989

`timer_gettime` — high-resolution timer operations, 3R-989

`timer_settime` — high-resolution timer operations, 3R-989

`times` — get process times, 3B-991

`tmpfile` — create a temporary file, 3S-992

`tmpnam` — create a name for a temporary file, 3S-993

`towlower` — Process Code character conversion macros, 3I-1019

`toupper` — Process Code character conversion macros, 3I-1019

translate address to symbolic information. — `dladdr`, 3X-251

translation between signal name and signal number — `str2sig`, 3C-884
`sig2str`, 3C-884

transport functions
 accept a connect request — `t_accept`, 3N-921
 acknowledge receipt of an orderly release indication — `t_rcvrel`, 3N-950
 allocate memory for argument structures — `t_alloc`, 3N-923

transport functions, *continued*
 bind an address to a transport endpoint — `t_bind`, 3N-925
 close a transport endpoint — `t_close`, 3N-928
 disable a transport endpoint — `t_unbind`, 3N-965
 establish a connection with another transport user — `t_connect`, 3N-929
 establish a transport endpoint — `t_open`, 3N-941
 free allocated memory — `t_free`, 3N-933
 get protocol-specific service information — `t_getinfo`, 3N-934
 get the current state — `t_getstate`, 3N-937
 initiate an orderly release of connection — `t_sndrel`, 3N-959
 listen for a connect request — `t_listen`, 3N-938
 look at the current event on a transport endpoint — `t_look`, 3N-940
 manage options for a transport endpoint — `t_optmgmt`, 3N-942
 produce error message — `t_error`, 3N-932
 receive a data unit — `t_rcvudata`, 3N-951
 receive a unit data error indication — `t_rcvuderr`, 3N-953
 receive data or expedited data sent over a connection — `t_rcv`, 3N-944
 receive the confirmation from a connect request — `t_rcvconnect`, 3N-946
 retrieve information from disconnect — `t_rcvdis`, 3N-948
 send a data unit — `t_sndudata`, 3N-960
 send data or expedited data over a connection — `t_snd`, 3N-955
 send user-initiated disconnect request — `t_snddis`, 3N-957
 synchronize transport library — `t_sync`, 3N-963

trigonometric functions
 — `acos`, 3M-995
 — `asin`, 3M-995
 — `atan`, 3M-995
 — `atan2`, 3M-995
 — `cos`, 3M-995

trigonometric functions, *continued*

- `sin`, 3M-995
- `tan`, 3M-995

`truncate` — set a file to a specified length, 3C-996

`ttyname` — find name of a terminal, 3C-1001

`ttyname_r` — find name of a terminal, 3C-1001

`ttyslot` — find the slot in the `utmp` file of the current user, 3C-1002

U

`uaddr2taddr` — generic transport name-to-address translation, 3N-633

`ualarm` — schedule signal after interval in microseconds, 3B-1003

`ungetwc` — push a Process Code character back into input stream, 3I-1005

`ungetwch` — get (or push back) `wchar_t` characters from curses terminal keyboard, 3X-180

unlock a pseudo-terminal master/slave pair — `unlockpt`, 3C-1006

unlock address space

- `munlockall`, 3C-608

unlock memory pages

- `munlock`, 3C-606

`unlockpt` — unlock a pseudo-terminal master/slave pair, 3C-1006

`updwtmp` — access `utmpx` file entry, 3C-469

`updwtmp` — access `utmpx` file entry, 3C-469

user contexts

- manipulate — `makecontext`, 3C-556

user ID

- get character-string representation — `cuserid`, 3S-239

users

- return information from remote machines — `rusers`, `rnusers`, 3N-794

`usleep` — suspend execution for interval in microseconds, 3B-1007

`utmp` file

- access entry — `getutent`, 3C-467
- find the slot of current user — `ttyslot`, 3C-1002

`utmpname` — access `utmp` file entry, 3C-467

`utmpx` file

- access entry — `getutxent`, 3C-469

`utmpxname` — access `utmpx` file entry, 3C-469

V

`vfprintf` — formatted output conversion, 3B-702

`vfstab` file

- `getvfstent`, 3C-472

virtual memory

- optimizing usage of user mapped memory — `madvise`, 3-552

`vlfmt` — display error message in standard format and pass to logging and monitoring services, 3C-1008

`vpfmt` — display error message in standard format and pass to logging and monitoring services, 3C-1010

`vprintf` — formatted output conversion, 3B-702

`vsprintf` — formatted output conversion, 3B-702

`vsyslog()` — log message with variable argument list, 3-1014

VTOC, disk's

- read a disk's VTOC — `read_vtoc`, 3X-731

- write a disk's VTOC — `write_vtoc`, 3X-731

W

`waddnwstr` — add a string of `wchar_t` characters to a curses window and advance cursor, 3X-162

`waddwch` — add a `wchar_t` character (with attributes) to a curses window and advance cursor, 3X-159

`waddwchnstr` — add string of `wchar_t` characters (and attributes) to a curses window, 3X-161

`waddwchstr` — add string of `wchar_t` characters (and attributes) to a curses window, 3X-161

`waddwstr` — add a string of `wchar_t` characters to a curses window and advance cursor, 3X-162

`wadjcurspos` — moving the cursor by character, 3X-163

`wait3` — wait for process to terminate or stop, 3B-1015

`watof` — convert wide character string to double-

precision number, 3I-1022
 watoi — convert wide character string to long integer, 3I-1024
 watol — convert wide character string to long integer, 3I-1024
 watoll — convert wide character string to long integer, 3I-1024
 wattroff — curses character and window attribute control routines, 3X-164
 wattron — curses character and window attribute control routines, 3X-164
 wattrset — curses character and window attribute control routines, 3X-164
 wchar_t string
 number conversion — wscanf, 3I-1035
 wconv — Process Code character conversion macros, 3I-1019
 wscat — wide character string operations, 3I-1029
 wcschr — wide character string operations, 3I-1030
 wscmp — wide character string operations, 3I-1029
 wscoll — wide character string comparison using collating information, 3I-1020
 wscpy — wide character string operations, 3I-1029
 wscspn — wide character string operations, 3I-1031
 wsetno — get information on EUC codesets, 3I-147
 wcsftime — convert date and time to wide character string, 3I-1021
 wcslen — wide character string operations, 3I-1030
 wcsncat — wide character string operations, 3I-1029
 wcsncmp — wide character string operations, 3I-1029
 wcsncpy — wide character string operations, 3I-1029
 wcsrchr — wide character string operations, 3I-1030
 wcsrchr — wide character string operations, 3I-1030
 wcsspn — wide character string operations, 3I-1031
 wcstod — convert wide character string to double-precision number, 3I-1022
 wcstok — wide character string operations, 3I-1031
 wcstol — convert wide character string to long integer, 3I-1024
 wcstombs — multibyte string functions, 3C-572
 wcstoul — convert wide character string to unsigned long, 3I-1026
 wcstring — wide character string operations, 3I-1029
 wcsvcs — wide character string operations, 3I-1031
 wcswidth — wide character string operations, 3I-1030
 wcsxfrm — wide character string transformation, 3I-1032
 wctype — define character class, 3I-1033
 wcwidth — wide character string operations, 3I-1030
 wechowchar — add a wchar_t character (with attributes) to a curses window and advance cursor, 3X-159
 wgetnwstr — get wchar_t character strings from curses terminal keyboard, 3X-184
 wgetwch — get (or push back) wchar_t characters from curses terminal keyboard, 3X-180
 wgetwstr — get wchar_t character strings from curses terminal keyboard, 3X-184
 wide character string comparison using collating information
 — wscoll, 3I-1020
 — wscoll, 3I-1020
 wide character string operations
 — wscat, 3I-1029
 — wcschr, 3I-1030
 — wscmp, 3I-1029
 — wscpy, 3I-1029
 — wscspn, 3I-1031
 — wcslen, 3I-1030
 — wcsncat, 3I-1029
 — wcsncmp, 3I-1029
 — wcsncpy, 3I-1029
 — wcsrchr, 3I-1030
 — wcsspn, 3I-1031

wide character string operations, *continued*

- `wcstok`, 3I-1031
- `wcstring`, 3I-1029
- `wcswcs`, 3I-1031
- `wcswidth`, 3I-1030
- `wcwidth`, 3I-1030
- `windex`, 3I-1030
- `wrindex`, 3I-1030

wide character string to long integer, convert

- `watoi`, 3I-1024
- `watol`, 3I-1024
- `watoll`, 3I-1024
- `wcstol`, 3I-1024
- `wstol`, 3I-1024

wide character string transformation — `wcsxfrm`, 3I-1032

`wsxfrm`, 3I-1032

`windex` — wide character string operations, 3I-1030

`winnwstr` — get a string of `wchar_t` characters from a curses window, 3X-200

`winsnwstr` — insert `wchar_t` string before character under the cursor in a curses window, 3X-197

`winswch` — insert a `wchar_t` character before the character under the cursor in a curses window, 3X-196

`winswstr` — insert `wchar_t` string before character under the cursor in a curses window, 3X-197

`winwch` — get a `wchar_t` character and its attributes from a curses window, 3X-198

`winwchnstr` — get a string of `wchar_t` characters (and attributes) from a curses window, 3X-199

`winwchstr` — get a string of `wchar_t` characters (and attributes) from a curses window, 3X-199

`winwstr` — get a string of `wchar_t` characters from a curses window, 3X-200

`wmovenextch` — moving the cursor by character, 3X-163

`wmoveprevch` — moving the cursor by character, 3X-163

working directory

get pathname — `getwd`, 3B-475

`wrindex` — wide character string operations,

3I-1030

`write_vtoc` — read and write a disk's VTOC, 3X-731

`wscasecmp` — Process Code string operations, 3I-1036

`wscol` — Process Code string operations, 3I-1036

`wscoll` — wide character string comparison using collating information, 3I-1020

`wsdup` — Process Code string operations, 3I-1036

`wscncasecmp` — Process Code string operations, 3I-1036

`wsprintf` — formatted output conversion, 3I-1034

`wsscanf` — formatted input conversion, 3I-1035

`wstandend` — curses character and window attribute control routines, 3X-164

`wstandout` — curses character and window attribute control routines, 3X-164

`wstod` — convert wide character string to double-precision number, 3I-1022

`wstol` — convert wide character string to long integer, 3I-1024

`wsxfrm` — wide character string transformation, 3I-1032

X

XDR library routines

- `xdr`, 3N-1037
- `xdr_admin`, 3N-1039
- `xdr_control`, 3N-1039
- `xdr_getpos`, 3N-1039
- `xdr_inline`, 3N-1039
- `xdr_setpos`, 3N-1039
- `xdr_sizeof`, 3N-1039
- `xdrrec_endofrecord`, 3N-1039
- `xdrrec_eof`, 3N-1039
- `xdrrec_readbytes`, 3N-1039
- `xdrrec_skiprecord`, 3N-1039

XDR library routines for complex data structures

- `xdr_array`, 3N-1042
- `xdr_bytes`, 3N-1042
- `xdr_complex`, 3N-1042
- `xdr_opaque`, 3N-1042
- `xdr_pointer`, 3N-1042

XDR library routines for complex data structures,

continued

- xdr_reference, 3N-1042
- xdr_string, 3N-1042
- xdr_union, 3N-1042
- xdr_vector, 3N-1042
- xdr_wrapstring, 3N-1042

XDR library routines for RPC

- rpc_xdr, 3N-789
- xdr_accepted_reply, 3N-789
- xdr_authsys_parms, 3N-789
- xdr_callhdr, 3N-789
- xdr_callmsg, 3N-789
- xdr_opaque_auth, 3N-789
- xdr_rejected_reply, 3N-789
- xdr_replymsg, 3N-789

XDR library routines for simple data structures

- xdr_bool, 3N-1047
- xdr_char, 3N-1047
- xdr_double, 3N-1047
- xdr_enum, 3N-1047
- xdr_float, 3N-1047
- xdr_free, 3N-1047
- xdr_hyper, 3N-1047
- xdr_int, 3N-1047
- xdr_long, 3N-1047
- xdr_longlong_t, 3N-1047
- xdr_quadruple, 3N-1047
- xdr_short, 3N-1047
- xdr_simple, 3N-1047
- xdr_u_char, 3N-1047
- xdr_u_hyper, 3N-1047
- xdr_u_int, 3N-1047
- xdr_u_long, 3N-1047
- xdr_u_longlong_t, 3N-1047
- xdr_u_short, 3N-1047
- xdr_void, 3N-1047

XDR stream creation library routines

- xdr_create, 3N-1045
- xdr_destroy, 3N-1045
- xdrmem_create, 3N-1045
- xdrrec_create, 3N-1045
- xdrstdio_create, 3N-1045

xdr_statstime — get performance data from remote kernel, 3N-793

xdr_statsvar — get performance data from remote kernel, 3N-793

Y

- y0 — Bessel function, 3M-122
- y1 — Bessel function, 3M-122
- yn — Bessel function, 3M-122