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 6 contains available games and demos.

- Section 7 describes various special files that refer to specific hardware peripherals, and device drivers. STREAMS software drivers, modules and the STREAMS-generic set of system calls are also described.

- Section 9 provides reference information needed to write device drivers in the kernel operating systems environment. It describes two device driver interface specifications: the Device Driver Interface (DDI) and the Driver–Kernel Interface (DKI).

- Section 9E describes the DDI/DKI, DDI-only, and DKI-only entry-point routines a developer may include in a device driver.

- Section 9F describes the kernel functions available for use by device drivers.

- Section 9S describes the data structures used by drivers to share information between the driver and the kernel.

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

**NAME**

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

**SYNOPSIS**

This section shows the syntax of commands or functions. When a command or file does not exist in the standard path, its full pathname is shown. 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.
- {} Braces. The options and/or arguments enclosed within braces are interdependent, such that everything enclosed must be treated as a unit.

**PROTOCOL**

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

**DESCRIPTION**

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

**IOCTL**

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

OPERANDS

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

OUTPUT

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

RETURN VALUES

If the man page documents functions that return values, this section lists these values and describes the conditions under which they are returned. If a function can return only constant values, such as 0 or −1, these values are listed in tagged paragraphs. Otherwise, a single paragraph describes the return values of each function. Functions declared 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.

EXIT STATUS

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

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

**ATTRIBUTES**

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

**SEE ALSO**

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

**DIAGNOSTICS**

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

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:

(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.

(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 libc(4), cc(1B) for other overrides, and the “C Com-
plaintion System” chapter of the ANSI C Programmer’s Guide for a discussion.
Some functions behave differently in standard-conforming environments. This
behavior is noted on the individual manual pages. See standards(5).

(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. See libelf(4).

(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.

(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. See libkvm(4).

(3M) These functions constitute the math library, libm. This library is implemented as
a shared object, libm.so, and as an archive, libm.a, but is not automatically
linked by the C compilation system. Specify –lm on the cc command line to link
with this library. See libm(4).

(3N) These functions constitute the Network Service Library, libnsl. It is imple-
mented as a shared object, libnsl.so, and as an archive, libnsl.a, but is not
automatically linked by the C compilation system. Specify –lnsl on the cc command line to link with this library. See libnsl(4).

modified 4 Apr 1997  SunOS 5.6  3-5
Some of the functions documented in man3n incorporate other network libraries, including:

- `libsocket` (see `libsocket(4)`),
- `libresolv` (see `libresolv(4)`),
- `librpcsvc` (see `librpcsvc(4)`),
- `libnisdb` (see `libnisdb(4)`),
- `librac` (see `librac(4)`),
- `libxfn` (see `libxfn(4)`), and
- `libkrb` (see `libkrb(4)`).

Many base networking functions are also available in the X/Open Networking Interfaces library, `libxnet`. See section (3XN) below for more information on the `libxnet` interfaces.

Under all circumstances, the use of the Sockets API is recommended over the XTI and TLI APIs. If portability to other XPGV4v2 systems is a requirement, the application must use the `libxnet` interfaces. If portability is not required, the sockets interfaces in `libsocket` and `libnsl` are recommended over those in `libxnet`. Between the XTI and TLI APIs, the XTI interfaces (available with `libxnet`) are recommended over the TLI interfaces (available with `libnsl`).

(3R) These functions constitute the POSIX.4 Realtime library, `libposix4`. It is implemented only as a shared object, `libposix4.so`, and is not automatically linked by the C compilation system. Specify `−lpix4` on the `cc` command line to link with this library. See `libposix4(4)`.

(3S) These functions constitute the “standard I/O package” (see `stdio(3S)`). They can be compiled using the the standard C library, `libc`, which is automatically linked by the C compilation system. The standard C library is implemented as a shared object, `libc.so`, and as an archive, `libc.a`. See `libc(4)`.

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

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

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

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

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

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

(3X) Specialized libraries. These functions are contained in libraries including, but not limited to,
- libadm (see libadm(4)),
- libbsdmalloc,
- libcrypt,
- libcurses,
- libdl (see libdl(4)),
- libform,
- libmail,
- libmalloc,
- libmapmalloc (see libmapmalloc(4)),
- libmenu, and
- libpanel.

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

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

**DEFINITIONS**

A character is any bit pattern able to fit into a byte on the machine.

*Exception*: in some international languages, a “character” may require more than one byte, and is represented in multi-bytes.

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

MT-Level of Libraries

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

FILES

```text
INCDIR  usually /usr/include
LIBDIR  usually /usr/ccs/lib
LIBDIR/libc.so
LIBDIR/libc.a
LIBDIR/libgen.a
LIBDIR/libm.a
LIBDIR/libsfm.sa
/usr/lib/libc.so.1
```

SEE ALSO


`Linker and Libraries Guide`

`Profiling Tools`

`ANSI C Programmer’s Guide`

DIAGNOSTICS

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

NOTES ON MULTITHREAD APPLICATIONS

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

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

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

```
cc [flags] file... −D_POSIX_PTHREAD_SEMANTICS −lpthread
```

For Solaris behavior, compile as follows:

```
cc [flags] file... −D_REENTRANT −lpthread
```
When building a singlethreaded application, the above flags should be undefined. This generates a binary that is executable on previous Solaris releases, which do not support multithreading.

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

MT-Safe interfaces are denoted in the 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.

**REALTIME APPLICATIONS**

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

**NOTES**

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

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{a64l(3C)}</td>
<td>convert between long integer and base-64 ASCII string</td>
</tr>
<tr>
<td>\texttt{abort(3C)}</td>
<td>terminate the process abnormally</td>
</tr>
<tr>
<td>\texttt{abs(3C)}</td>
<td>return absolute value of integer</td>
</tr>
<tr>
<td>\texttt{accept(3N)}</td>
<td>accept a connection on a socket</td>
</tr>
<tr>
<td>\texttt{accept(3XN)}</td>
<td>accept a new connection on a socket</td>
</tr>
<tr>
<td>\texttt{aclcheck(3)}</td>
<td>check the validity of an ACL</td>
</tr>
<tr>
<td>\texttt{aclfrommode(3)}</td>
<td>See \texttt{acltomode(3)}</td>
</tr>
<tr>
<td>\texttt{aclfrompbits(3)}</td>
<td>See \texttt{acltopbits(3)}</td>
</tr>
<tr>
<td>\texttt{aclfromtext(3)}</td>
<td>See \texttt{acltotext(3)}</td>
</tr>
<tr>
<td>\texttt{aclsort(3)}</td>
<td>sort an ACL</td>
</tr>
<tr>
<td>\texttt{acltomode(3)}</td>
<td>convert an ACL to/from permission bits</td>
</tr>
<tr>
<td>\texttt{acltopbits(3)}</td>
<td>convert an ACL to/from permission bits</td>
</tr>
<tr>
<td>\texttt{acltotext(3)}</td>
<td>convert an internal representation to/from external representation</td>
</tr>
<tr>
<td>\texttt{acos(3M)}</td>
<td>arc cosine function</td>
</tr>
<tr>
<td>\texttt{acosh(3M)}</td>
<td>inverse hyperbolic functions</td>
</tr>
<tr>
<td>\texttt{addch(3X)}</td>
<td>See \texttt{curs_addch(3X)}</td>
</tr>
<tr>
<td>\texttt{addch(3XC)}</td>
<td>add a character (with rendition) to a window</td>
</tr>
<tr>
<td>\texttt{addchnstr(3X)}</td>
<td>See \texttt{curs_addchnstr(3X)}</td>
</tr>
<tr>
<td>\texttt{addchnstr(3XC)}</td>
<td>See \texttt{addchnstr(3XC)}</td>
</tr>
<tr>
<td>\texttt{addchstr(3X)}</td>
<td>See \texttt{curs_addchstr(3X)}</td>
</tr>
<tr>
<td>\texttt{addchstr(3XC)}</td>
<td>copy a character string (with renditions) to a window</td>
</tr>
<tr>
<td>\texttt{addnstr(3X)}</td>
<td>See \texttt{curs_addnstr(3X)}</td>
</tr>
<tr>
<td>\texttt{addnstr(3XC)}</td>
<td>add a multi-byte character string (without rendition) to a window</td>
</tr>
<tr>
<td>\texttt{addnwstr(3X)}</td>
<td>See \texttt{curs_addnwstr(3X)}</td>
</tr>
<tr>
<td>\texttt{addnwstr(3XC)}</td>
<td>add a wide-character string to a window</td>
</tr>
<tr>
<td>\texttt{addsev(3C)}</td>
<td>define additional severities</td>
</tr>
</tbody>
</table>
addseverity(3C) build a list of severity levels for an application for use with fmtmsg
addstr(3X) See curs_addstr(3X)
addstr(3XC) See addnstr(3XC)
addwch(3X) See curs_addwch(3X)
add_wch(3XC) add a complex character (with rendition) to a window
addwcharstr(3X) See curs_addwcharstr(3X)
add_wchnstr(3XC) copy a string of complex characters (with renditions) to a window
addwcharstr(3X) See curs_addwcharstr(3X)
add_wchnstr(3XC) See addnstr(3XC)
addwcharstr(3X) See curs_addwcharstr(3X)
addwchar(3XC) add a complex character (with rendition) to a window
addwcharstr(3X) See curs_addwcharstr(3X)
addwcharstr(3XC) See addnstr(3XC)
addwcharstr(3X) See curs_addwcharstr(3X)
addwcharstr(3XC) See addnstr(3XC)
adjcurspos(3X) See curs_alecompat(3X)
advance(3G) See regexpr(3G)
aiocancel(3) cancel an asynchronous operation
aio_cancel(3R) cancel asynchronous I/O request
aio_error(3R) See aio_return(3R)
aio_fsync(3R) asynchronous file synchronization
aioread(3) read or write asynchronous I/O operations
aioread(3R) asynchronous read and write operations
aioread64(3) See aioread(3)
aio_return(3R) retrieve return or error status of asynchronous I/O operation
aiosuspend(3R) wait for asynchronous I/O request
aiowait(3) wait for completion of asynchronous I/O operation
aiowrite(3) See aioread(3)
aiowrite(3R) See aioread(3R)
aiowrite64(3) See aioread(3)
alloc(3C) See malloc(3C)
alphasort(3B) See scandir(3B)
arc(3) See plot(3)
ascftime(3C) See strftime(3C)
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>asctime(3C)</td>
<td>See ctime(3C)</td>
</tr>
<tr>
<td>asctime_r(3C)</td>
<td>See ctime(3C)</td>
</tr>
<tr>
<td>asin(3M)</td>
<td>arc sine function</td>
</tr>
<tr>
<td>asinh(3M)</td>
<td>See acosh(3M)</td>
</tr>
<tr>
<td>assert(3C)</td>
<td>verify program assertion</td>
</tr>
<tr>
<td>asysmem(3)</td>
<td>See sysmem(3)</td>
</tr>
<tr>
<td>atan(3M)</td>
<td>arc tangent function</td>
</tr>
<tr>
<td>atan2(3M)</td>
<td>arc tangent function</td>
</tr>
<tr>
<td>atanh(3M)</td>
<td>See acosh(3M)</td>
</tr>
<tr>
<td>atexit(3C)</td>
<td>add program termination routine</td>
</tr>
<tr>
<td>atof(3C)</td>
<td>See strtod(3C)</td>
</tr>
<tr>
<td>atoi(3C)</td>
<td>See strtol(3C)</td>
</tr>
<tr>
<td>atol(3C)</td>
<td>See strtol(3C)</td>
</tr>
<tr>
<td>atoll(3C)</td>
<td>See strtol(3C)</td>
</tr>
<tr>
<td>attr_get(3XC)</td>
<td>control window attributes</td>
</tr>
<tr>
<td>attroff(3X)</td>
<td>See curs_attr(3X)</td>
</tr>
<tr>
<td>attroff(3XC)</td>
<td>change foreground window attributes</td>
</tr>
<tr>
<td>attr_off(3XC)</td>
<td>See attr_get(3XC)</td>
</tr>
<tr>
<td>attron(3X)</td>
<td>See curs_attr(3X)</td>
</tr>
<tr>
<td>attr_on(3XC)</td>
<td>See attr_get(3XC)</td>
</tr>
<tr>
<td>attron(3XC)</td>
<td>See attr_get(3XC)</td>
</tr>
<tr>
<td>attrset(3X)</td>
<td>See curs_attr(3X)</td>
</tr>
<tr>
<td>attr_set(3XC)</td>
<td>See attr_get(3XC)</td>
</tr>
<tr>
<td>attrset(3XC)</td>
<td>See attroff(3XC)</td>
</tr>
<tr>
<td>au_close(3)</td>
<td>See au_open(3)</td>
</tr>
<tr>
<td>au_open(3)</td>
<td>construct and write audit records</td>
</tr>
<tr>
<td>au_preselect(3)</td>
<td>preselect an audit event</td>
</tr>
<tr>
<td>authdes_create(3N)</td>
<td>See rpc_soc(3N)</td>
</tr>
<tr>
<td>authdes_getucred(3N)</td>
<td>See secure_rpc(3N)</td>
</tr>
<tr>
<td>authdes_seccreate(3N)</td>
<td>See secure_rpc(3N)</td>
</tr>
<tr>
<td>auth_destroy(3N)</td>
<td>See rpc_clnt_auth(3N)</td>
</tr>
<tr>
<td>authkerb_getucred(3N)</td>
<td>See kerberos_rpc(3N)</td>
</tr>
<tr>
<td>authkerb_seccreate(3N)</td>
<td>See kerberos_rpc(3N)</td>
</tr>
<tr>
<td>authnone_create(3N)</td>
<td>See rpc_clnt_auth(3N)</td>
</tr>
<tr>
<td>authsys_create(3N)</td>
<td>See rpc_clnt_auth(3N)</td>
</tr>
</tbody>
</table>
authsys_create_default(3N) See rpc_clnt_auth(3N)
authunix_create(3N) See rpc_soc(3N)
authunix_create_default(3N) See rpc_soc(3N)
au_to(3) create audit record tokens
au_to_arg(3) See au_to(3)
au_to_attr(3) See au_to(3)
au_to_data(3) See au_to(3)
au_to_groups(3) See au_to(3)
au_to_in_addr(3) See au_to(3)
au_to_ipc(3) See au_to(3)
au_to_ipc_perm(3) See au_to(3)
au_to_iport(3) See au_to(3)
au_to_me(3) See au_to(3)
au_to_opaque(3) See au_to(3)
au_to_path(3) See au_to(3)
au_to_process(3) See au_to(3)
au_to_return(3) See au_to(3)
au_to_socket(3) See au_to(3)
au_to_text(3) See au_to(3)
au_user_mask(3) get user’s binary preselection mask
au_write(3) See au_open(3)
basename(3C) return the last element of a path name
baudrate(3X) See curs_termattrs(3X)
baudrate(3XC) return terminal baud rate
bcmp(3C) See bstring(3C)
bcopy(3C) See bstring(3C)
beep(3X) See curs_beep(3X)
beep(3XC) activate audio-visual alarm
bgets(3G) read stream up to next delimiter
bind(3G) bind a name to a socket
bind(3N) bind a name to a socket
bindtextdomain(3C) See gettext(3C)
bkgd(3X) See curs_bkgd(3X)
bkgd(3XC) set the background character (and rendition) of window
 Intro (3)  C Library Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bkgdset(3X)</td>
<td>See curses_bkgd(3X)</td>
</tr>
<tr>
<td>bkgdset(3XC)</td>
<td>See bkgd(3XC)</td>
</tr>
<tr>
<td>bkgrnd(3XC)</td>
<td>set or get the background character (and rendition) of window using a complex character</td>
</tr>
<tr>
<td>bkgrndset(3XC)</td>
<td>See bkgrd(3XC)</td>
</tr>
<tr>
<td>border(3X)</td>
<td>See curses_border(3X)</td>
</tr>
<tr>
<td>border(3XC)</td>
<td>add a single-byte border to a window</td>
</tr>
<tr>
<td>border_set(3XC)</td>
<td>use complex characters (and renditions) to draw borders</td>
</tr>
<tr>
<td>bottom_panel(3X)</td>
<td>See panel_top(3X)</td>
</tr>
<tr>
<td>box(3)</td>
<td>See plot(3)</td>
</tr>
<tr>
<td>box(3X)</td>
<td>See curses_border(3X)</td>
</tr>
<tr>
<td>box(3XC)</td>
<td>See border(3XC)</td>
</tr>
<tr>
<td>box_set(3XC)</td>
<td>See border_set(3XC)</td>
</tr>
<tr>
<td>bsdmalloc(3X)</td>
<td>memory allocator</td>
</tr>
<tr>
<td>bsd_signal(3C)</td>
<td>simplified signal facilities</td>
</tr>
<tr>
<td>bsearch(3C)</td>
<td>binary search a sorted table</td>
</tr>
<tr>
<td>bstring(3C)</td>
<td>bit and byte string operations</td>
</tr>
<tr>
<td>bfunsplit(3G)</td>
<td>split buffer into fields</td>
</tr>
<tr>
<td>byteorder(3N)</td>
<td>convert values between host and network byte order</td>
</tr>
<tr>
<td>bzero(3C)</td>
<td>See bstring(3C)</td>
</tr>
<tr>
<td>calloc(3C)</td>
<td>See malloc(3C)</td>
</tr>
<tr>
<td>calloc(3X)</td>
<td>See malloc(3X)</td>
</tr>
<tr>
<td>calloc(3X)</td>
<td>See mapmalloc(3X)</td>
</tr>
<tr>
<td>calloc(3X)</td>
<td>See watchmalloc(3X)</td>
</tr>
<tr>
<td>callrpc(3N)</td>
<td>See rpc_soc(3N)</td>
</tr>
<tr>
<td>cancellation(3T)</td>
<td>overview of concepts related to POSIX thread cancellation</td>
</tr>
<tr>
<td>can_change_color(3X)</td>
<td>See curses_color(3X)</td>
</tr>
<tr>
<td>can_change_color(3XC)</td>
<td>manipulate color information</td>
</tr>
<tr>
<td>catclose(3C)</td>
<td>See catopen(3C)</td>
</tr>
<tr>
<td>catgets(3C)</td>
<td>read a program message</td>
</tr>
<tr>
<td>catopen(3C)</td>
<td>open/close a message catalog</td>
</tr>
<tr>
<td>cbreak(3X)</td>
<td>See curses_inopts(3X)</td>
</tr>
</tbody>
</table>
cbreak(3XC)
cbrt(3M)
ceil(3M)
cfgetispeed(3)
cfgetospeed(3)
cfree(3X)
cfree(3X)
cfsetispeed(3)
cfsetspeed(3)
cftime(3C)
chgat(3XC)
circle(3)
clear(3X)
clear(3XC)
clearerr(3S)
clearok(3X)
clearok(3XC)
clnt_broadcast(3N)
clnt_call(3N)
clnt_control(3N)
clnt_create(3N)
clnt_create_timed(3N)
clnt_create_vers(3N)
clnt_create_vers_timed(3N)
clnt_destroy(3N)
clnt_dg_create(3N)
clnt_freeres(3N)
clnt_geterr(3N)
clnt_pcreateerror(3N)
clnt_perror(3N)
clnt_raw_create(3N)
clntraw_create(3N)
clnt_spcreateerror(3N)
set input mode controls
cube root function
ceiling value function
get input and output baud rate
See cfgetispeed(3)
See mapmalloc(3X)
set input and output baud rate
See cfsetispeed(3)
See strftime(3C)
change the rendition of characters in a window
See plot(3)
clear a window
See ferror(3S)
See curs_outopts(3X)
set terminal output controls
See rpc_soc(3N)
See rpc_clnt_calls(3N)
See rpc_clnt_create(3N)
See rpc_clnt_calls(3N)
See rpc_clnt_create(3N)
See rpc_clnt_create(3N)
See rpc_clnt_calls(3N)
See rpc_clnt_create(3N)
See rpc_clnt_create(3N)
Intro (3) C Library Functions

clnt_sperrno(3N) See rpc_clnt_calls(3N)
clnt_sperror(3N) See rpc_clnt_calls(3N)
clnttcp_create(3N) See rpc_soc(3N)
clnt_tli_create(3N) See rpc_clnt_create(3N)
clnt_tp_create(3N) See rpc_clnt_create(3N)
clnt_tp_create_timed(3N)
clntudp_bufcreate(3N) See rpc_soc(3N)
clntudp_create(3N) See rpc_soc(3N)
clnt_vc_create(3N) See rpc_clnt_create(3N)
clock(3C) report CPU time used
clock_getres(3R) See clock_settime(3R)
clock_gettime(3R) See clock_settime(3R)
clock_settime(3R) high-resolution clock operations
closedir(3C)
closelog(3) See syslog(3)
closepl(3) See plot(3)
closevt(3) See plot(3)
clrtoeol(3X) See curs_clear(3X)
clrtoeol(3XC) clear to the end of a window
clrtoeol(3XC)
color_content(3X) See curs_color(3X)
color_content(3XC) See can_change_color(3XC)
COLOR_PAIR(3XC) See can_change_color(3XC)
color_set(3XC) See attr_get(3XC)
compile(3G) See regexpr(3G)
cond_broadcast(3T) See condition(3T)
cond_destroy(3T) See condition(3T)
cond_init(3T) See condition(3T)
condition(3T) condition variables
cond_signal(3T) See condition(3T)
cond_timedwait(3T) See condition(3T)
cond_wait(3T) See condition(3T)
confstr(3C) get configurable variables
connect(3N) initiate a connection on a socket
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>connect(3XN)</td>
<td>connect a socket</td>
</tr>
<tr>
<td>cont(3)</td>
<td>See plot(3)</td>
</tr>
<tr>
<td>copylist(3G)</td>
<td>copy a file into memory</td>
</tr>
<tr>
<td>copysign(3M)</td>
<td>return magnitude of first argument and sign of second argument</td>
</tr>
<tr>
<td>copywin(3X)</td>
<td>See curs_overlay(3X)</td>
</tr>
<tr>
<td>copywin(3XC)</td>
<td>overlay or overwrite any portion of window</td>
</tr>
<tr>
<td>cos(3M)</td>
<td>cosine function</td>
</tr>
<tr>
<td>cosh(3M)</td>
<td>hyperbolic cosine function</td>
</tr>
<tr>
<td>cplus_demangle(3)</td>
<td>See demangle(3)</td>
</tr>
<tr>
<td>crypt(3C)</td>
<td>string encoding function</td>
</tr>
<tr>
<td>cset(3C)</td>
<td>get information on EUC codesets</td>
</tr>
<tr>
<td>csetcol(3C)</td>
<td>See cset(3C)</td>
</tr>
<tr>
<td>csetlen(3C)</td>
<td>See cset(3C)</td>
</tr>
<tr>
<td>csetno(3C)</td>
<td>See cset(3C)</td>
</tr>
<tr>
<td>ctermid(3S)</td>
<td>generate path name for controlling terminal</td>
</tr>
<tr>
<td>ctermid_r(3S)</td>
<td>See ctermid(3S)</td>
</tr>
<tr>
<td>ctime(3C)</td>
<td>convert date and time to string</td>
</tr>
<tr>
<td>ctime_r(3C)</td>
<td>See ctime(3C)</td>
</tr>
<tr>
<td>ctype(3C)</td>
<td>character handling</td>
</tr>
<tr>
<td>current_field(3X)</td>
<td>See form_page(3X)</td>
</tr>
<tr>
<td>current_item(3X)</td>
<td>See menu_item_current(3X)</td>
</tr>
<tr>
<td>curs_addch(3X)</td>
<td>add a character (with attributes) to a curses window and advance cursor</td>
</tr>
<tr>
<td>curs_addchstr(3X)</td>
<td>add string of characters (and attributes) to a curses window</td>
</tr>
<tr>
<td>curs_addstr(3X)</td>
<td>add a string of characters to a curses window and advance cursor</td>
</tr>
<tr>
<td>curs_addwch(3X)</td>
<td>add a wchar_t character (with attributes) to a curses window and advance cursor</td>
</tr>
<tr>
<td>curs_addwchstr(3X)</td>
<td>add string of wchar_t characters (and attributes) to a curses window</td>
</tr>
<tr>
<td>curs_addwstr(3X)</td>
<td>add a string of wchar_t characters to a curses window and advance cursor</td>
</tr>
<tr>
<td>curs_alecompat(3X)</td>
<td>these functions are added to ALE curses</td>
</tr>
</tbody>
</table>

modified 4 Apr 1997
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>curs_attr</td>
<td>library for moving the cursor by character.</td>
</tr>
<tr>
<td>curs_bell</td>
<td>curses bell and screen flash routines</td>
</tr>
<tr>
<td>curs_bkgd</td>
<td>curses window background manipulation routines</td>
</tr>
<tr>
<td>curs.Border</td>
<td>create curses borders, horizontal and vertical lines</td>
</tr>
<tr>
<td>curs.Clear</td>
<td>clear all or part of a curses window</td>
</tr>
<tr>
<td>curs.Color</td>
<td>curses color manipulation routines</td>
</tr>
<tr>
<td>curs.Delch</td>
<td>delete character under cursor in a curses window</td>
</tr>
<tr>
<td>curs.Deleteln</td>
<td>delete and insert lines in a curses window</td>
</tr>
<tr>
<td>Curses</td>
<td>CRT screen handling and optimization package</td>
</tr>
<tr>
<td>Curses XC</td>
<td>introduction and overview of X/Open Curses</td>
</tr>
<tr>
<td>curs.Getch</td>
<td>get (or push back) characters from curses terminal keyboard</td>
</tr>
<tr>
<td>curs.Getstr</td>
<td>get character strings from curses terminal keyboard</td>
</tr>
<tr>
<td>curs.GetwcH</td>
<td>get (or push back) wchar_t characters from curses terminal keyboard</td>
</tr>
<tr>
<td>curs.Getwstr</td>
<td>get wchar_t character strings from curses terminal keyboard</td>
</tr>
<tr>
<td>curs.Getyx</td>
<td>get curses cursor and window coordinates</td>
</tr>
<tr>
<td>curs.Inch</td>
<td>get a character and its attributes from a curses window</td>
</tr>
<tr>
<td>curs.Inchstr</td>
<td>get a string of characters (and attributes) from a curses window</td>
</tr>
<tr>
<td>curs.Initscr</td>
<td>curses screen initialization and manipulation routines</td>
</tr>
<tr>
<td>curs.Inopts</td>
<td>curses terminal input option control routines</td>
</tr>
<tr>
<td>curs.Insch</td>
<td>insert a character before the character under the cursor in a curses window</td>
</tr>
<tr>
<td>curs.Insstr</td>
<td>insert string before character under the window</td>
</tr>
<tr>
<td>Function Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>curs_instr(3X)</td>
<td>get a string of characters from a curses window</td>
</tr>
<tr>
<td>curs_outwch(3X)</td>
<td>insert wchar_t character before the character under the cursor in a curses window</td>
</tr>
<tr>
<td>curs_inwstr(3X)</td>
<td>insert wchar_t string before character under the cursor in a curses window</td>
</tr>
<tr>
<td>curs_inwch(3X)</td>
<td>get a wchar_t character and its attributes from a curses window</td>
</tr>
<tr>
<td>curs_inwchstr(3X)</td>
<td>get a string of wchar_t characters (and attributes) from a curses window</td>
</tr>
<tr>
<td>curs_inwstr(3X)</td>
<td>get a string of wchar_t characters from a curses window</td>
</tr>
<tr>
<td>curs_kernel(3X)</td>
<td>low-level curses routines</td>
</tr>
<tr>
<td>curs_move(3X)</td>
<td>move curses window cursor</td>
</tr>
<tr>
<td>curs_outopts(3X)</td>
<td>curses terminal output option control routines</td>
</tr>
<tr>
<td>curs_overlay(3X)</td>
<td>overlap and manipulate overlapped curses windows</td>
</tr>
<tr>
<td>curs_pad(3X)</td>
<td>create and display curses pads</td>
</tr>
<tr>
<td>curs_printw(3X)</td>
<td>print formatted output in curses windows</td>
</tr>
<tr>
<td>curs_refresh(3X)</td>
<td>refresh curses windows and lines</td>
</tr>
<tr>
<td>curs_scanw(3X)</td>
<td>convert formatted input from a curses widow</td>
</tr>
<tr>
<td>curs_scr_dump(3X)</td>
<td>read (write) a curses screen from (to) a file</td>
</tr>
<tr>
<td>curs_scroll(3X)</td>
<td>scroll a curses window</td>
</tr>
<tr>
<td>curs_set(3X)</td>
<td>set visibility of cursor</td>
</tr>
<tr>
<td>curs_set(3XC)</td>
<td>set visibility of cursor</td>
</tr>
<tr>
<td>curs_slk(3X)</td>
<td>curses soft label routines</td>
</tr>
<tr>
<td>curs_termattrs(3X)</td>
<td>curses environment query routines</td>
</tr>
<tr>
<td>curs_termcap(3X)</td>
<td>curses interfaces (emulated) to the termcap library</td>
</tr>
<tr>
<td>curs_terminfo(3X)</td>
<td>curses interfaces to terminfo database</td>
</tr>
<tr>
<td>curs_touch(3X)</td>
<td>curses refresh control routines</td>
</tr>
<tr>
<td>curs_util(3X)</td>
<td>curses miscellaneous utility routines</td>
</tr>
<tr>
<td>curs_window(3X)</td>
<td>create curses windows</td>
</tr>
</tbody>
</table>
cuserid(3S)  get character login name of the user

data_ahead(3X)  See form_data(3X)
data_behind(3X)  See form_data(3X)
db_add_entry(3N)  See nis_db(3N)
db_checkpoint(3N)  See nis_db(3N)
db_create_table(3N)  See nis_db(3N)
db_destroy_table(3N)  See nis_db(3N)
db_first_entry(3N)  See nis_db(3N)
db_free_result(3N)  See nis_db(3N)
db_initialize(3N)  See nis_db(3N)
db_list_entries(3N)  See nis_db(3N)
dbm(3B)  data base subroutines
dbm_clearerr(3)  database functions
dbm_close(3)  See dbm_clearerr(3)

dbm_close(3B)  See dbm(3B)
dbm_delete(3)  See dbm_clearerr(3)
dbm_error(3)  See dbm_clearerr(3)
dbm_fetch(3)  See dbm_clearerr(3)
dbm_firstkey(3)  See dbm_clearerr(3)
dbminit(3B)  See dbm(3B)
dbm_nextkey(3)  See dbm_clearerr(3)
dbm_open(3)  See dbm_clearerr(3)
dbm_store(3)  See dbm_clearerr(3)
db_next_entry(3N)  See nis_db(3N)
db_remove_entry(3N)  See nis_db(3N)
db_reset_next_entry(3N)  See nis_db(3N)
db_standby(3N)  See nis_db(3N)
db_table_exists(3N)  See nis_db(3N)
db_unload_table(3N)  See nis_db(3N)
dcgettext(3C)  See gettext(3C)
decimal_to_double(3)  See decimal_to_floating(3)
decimal_to_extended(3)  See decimal_to_floating(3)
decimal_to_floating(3)  convert decimal record to floating-point value
decimal_to_quadruple(3)  See decimal_to_floating(3)
decimal_to_single(3)
See decimal_to_floating(3)
def_prog_mode(3X)
See curs_kernel(3X)
def_prog_mode(3XC)
save/restore terminal modes
def_shell_mode(3X)
See curs_kernel(3X)
def_shell_mode(3XC)
See def_prog_mode(3XC)
delay_output(3X)
See curs_util(3X)
delay_output(3XC)
delays output
delch(3X)
See curs_delch(3X)
delch(3XC)
remove a character
del_curterm(3X)
See curs_terminfo(3X)
del_curterm(3XC)
free space pointed to by terminal
delete(3B)
See dbm(3B)
deleteln(3X)
See curs_deleteln(3X)
deleteln(3XC)
remove a line
del_panel(3X)
See panel_new(3X)
delscreen(3X)
See curs_initscr(3X)
delscreen(3XC)
free space associated with the SCREEN data structure
delwin(3X)
See curs_window(3X)
delwin(3XC)
delete a window
demangle(3)
devid_compare(3)
devid_deviceid_to_nmlist(3)
devid_free(3)
devid_free_nmlist(3)
devid_get(3)
devid_get_minor_name(3)
devid_sizeof(3)
dgettext(3C)
dial(3N)
difftime(3C)
directio(3C)
delch(3X)
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dirname(3C)</td>
<td>report the parent directory name of a file path name</td>
</tr>
<tr>
<td>div(3C)</td>
<td>compute the quotient and remainder</td>
</tr>
<tr>
<td>dladdr(3X)</td>
<td>translate address to symbolic information</td>
</tr>
<tr>
<td>dlclose(3X)</td>
<td>close a shared object</td>
</tr>
<tr>
<td>dl dump(3X)</td>
<td>create a new file from a dynamic object component of the calling process</td>
</tr>
<tr>
<td>dl error(3X)</td>
<td>get diagnostic information</td>
</tr>
<tr>
<td>dl info(3X)</td>
<td>dynamic load information</td>
</tr>
<tr>
<td>dl open(3X)</td>
<td>gain access to an executable object file</td>
</tr>
<tr>
<td>dl sym(3X)</td>
<td>get the address of a symbol in a shared object</td>
</tr>
<tr>
<td>Dmi Add Component(3X)</td>
<td>Management Interface database administration functions</td>
</tr>
<tr>
<td>Dmi Add Group(3X)</td>
<td>See Dmi Add Component(3X)</td>
</tr>
<tr>
<td>Dmi Add Language(3X)</td>
<td>See Dmi Add Component(3X)</td>
</tr>
<tr>
<td>Dmi Add Row(3X)</td>
<td>Management Interface operation functions</td>
</tr>
<tr>
<td>Dmi Delete Component(3X)</td>
<td>See Dmi Add Component(3X)</td>
</tr>
<tr>
<td>Dmi Delete Group(3X)</td>
<td>See Dmi Add Component(3X)</td>
</tr>
<tr>
<td>Dmi Delete Language(3X)</td>
<td>See Dmi Add Component(3X)</td>
</tr>
<tr>
<td>Dmi Delete Row(3X)</td>
<td>See Dmi Add Component(3X)</td>
</tr>
<tr>
<td>Dmi Get Attribute(3X)</td>
<td>See Dmi Add Component(3X)</td>
</tr>
<tr>
<td>Dmi Get Config(3X)</td>
<td>See Dmi Add Component(3X)</td>
</tr>
<tr>
<td>Dmi Get Multiple(3X)</td>
<td>Management Interface listing functions</td>
</tr>
<tr>
<td>Dmi Get Version(3X)</td>
<td>See Dmi Get Config(3X)</td>
</tr>
<tr>
<td>Dmi List Attributes(3X)</td>
<td>Management Interface listing functions</td>
</tr>
<tr>
<td>Dmi List Class Names(3X)</td>
<td>See Dmi List Attributes(3X)</td>
</tr>
<tr>
<td>Dmi List Components(3X)</td>
<td>See Dmi List Attributes(3X)</td>
</tr>
<tr>
<td>Dmi List Components By Class(3X)</td>
<td>See Dmi List Attributes(3X)</td>
</tr>
<tr>
<td>Dmi List Groups(3X)</td>
<td>See Dmi List Attributes(3X)</td>
</tr>
<tr>
<td>Dmi List Languages(3X)</td>
<td>See Dmi List Attributes(3X)</td>
</tr>
<tr>
<td>Dmi Originate Event(3X)</td>
<td>See Dmi RegisterCi(3X)</td>
</tr>
<tr>
<td>Dmi Register(3X)</td>
<td>See Dmi Get Config(3X)</td>
</tr>
</tbody>
</table>
Service Provider functions for components
See DmiAddRow(3X)
See DmiGetConfig(3X)
See DmiAddRow(3X)
See DmiGetConfig(3X)
See DmiRegisterCi(3X)
See resolver(3N)
bind or unbind the current thread with the door server pool
invoke the function associated with a door descriptor
create a door descriptor
return credential information associated with the client
return information associated with a door descriptor
return from a door invocation
revoke access to a door descriptor
specify an alternative door server thread creation function
See door_bind(3X)
See floating_to_decimal(3)
See curs_refresh(3X)
refresh windows and lines
generate uniformly distributed pseudo-random numbers
duplicate an open file descriptor
See form_field_new(3X)
See curs_window(3X)
duplicate a window
See form_field_info(3X)
See curs_inopts(3X)
enable/disable terminal echo
See curs_addch(3X)
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>echochar(3XC)</td>
<td>add a single-byte character and refresh window</td>
</tr>
<tr>
<td>echowchar(3X)</td>
<td>See curs_addch(3X)</td>
</tr>
<tr>
<td>echo_wchar(3XC)</td>
<td>add a complex character and refresh window</td>
</tr>
<tr>
<td>econvert(3)</td>
<td>output conversion</td>
</tr>
<tr>
<td>ecvt(3)</td>
<td>See econvert(3)</td>
</tr>
<tr>
<td>ecvt(3C)</td>
<td>convert floating-point number to string</td>
</tr>
<tr>
<td>_edata(3C)</td>
<td>See end(3C)</td>
</tr>
<tr>
<td>edata(3C)</td>
<td>See end(3C)</td>
</tr>
<tr>
<td>elf(3E)</td>
<td>object file access library</td>
</tr>
<tr>
<td>elf32_fsize(3E)</td>
<td>return the size of an object file type</td>
</tr>
<tr>
<td>elf32_getehdr(3E)</td>
<td>retrieve class-dependent object file header</td>
</tr>
<tr>
<td>elf32_getphdr(3E)</td>
<td>retrieve class-dependent program header table</td>
</tr>
<tr>
<td>elf32_getshdr(3E)</td>
<td>retrieve class-dependent section header</td>
</tr>
<tr>
<td>elf32_newehdr(3E)</td>
<td>See elf32_getehdr(3E)</td>
</tr>
<tr>
<td>elf32_newphdr(3E)</td>
<td>See elf32_getphdr(3E)</td>
</tr>
<tr>
<td>elf32_xlatetof(3E)</td>
<td>class-dependent data translation</td>
</tr>
<tr>
<td>elf32_xlatetom(3E)</td>
<td>See elf32_xlatetof(3E)</td>
</tr>
<tr>
<td>elf_begin(3E)</td>
<td>process ELF object files</td>
</tr>
<tr>
<td>elf_cntl(3E)</td>
<td>control an elf file descriptor</td>
</tr>
<tr>
<td>elf_end(3E)</td>
<td>See elf_begin(3E)</td>
</tr>
<tr>
<td>elf_errmsg(3E)</td>
<td>error handling</td>
</tr>
<tr>
<td>elf_errno(3E)</td>
<td>See elf_errmsg(3E)</td>
</tr>
<tr>
<td>elf_fill(3E)</td>
<td>set fill byte</td>
</tr>
<tr>
<td>elf_flagdata(3E)</td>
<td>manipulate flags</td>
</tr>
<tr>
<td>elf_flagehdr(3E)</td>
<td>See elf_flagdata(3E)</td>
</tr>
<tr>
<td>elf_flagelf(3E)</td>
<td>See elf_flagdata(3E)</td>
</tr>
<tr>
<td>elf_flagphdr(3E)</td>
<td>See elf_flagdata(3E)</td>
</tr>
<tr>
<td>elf_flagscn(3E)</td>
<td>See elf_flagdata(3E)</td>
</tr>
<tr>
<td>elf_flagshdr(3E)</td>
<td>See elf_flagdata(3E)</td>
</tr>
<tr>
<td>elf_getarhdr(3E)</td>
<td>retrieve archive member header</td>
</tr>
<tr>
<td>elf_getarsym(3E)</td>
<td>retrieve archive symbol table</td>
</tr>
<tr>
<td>elf_getbase(3E)</td>
<td>get the base offset for an object file</td>
</tr>
<tr>
<td>elf_getdata(3E)</td>
<td>get section data</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------------------------------------------------------</td>
</tr>
<tr>
<td>elf_getident(3E)</td>
<td>retrieve file identification data</td>
</tr>
<tr>
<td>elf_getscn(3E)</td>
<td>get section information</td>
</tr>
<tr>
<td>elf_hash(3E)</td>
<td>compute hash value</td>
</tr>
<tr>
<td>elf_kind(3E)</td>
<td>determine file type</td>
</tr>
<tr>
<td>elf_memory(3E)</td>
<td>See elf_begin(3E)</td>
</tr>
<tr>
<td>elf_ndxscn(3E)</td>
<td>See elf_getscn(3E)</td>
</tr>
<tr>
<td>elf_newdata(3E)</td>
<td>See elf_getdata(3E)</td>
</tr>
<tr>
<td>elf_newscn(3E)</td>
<td>See elf_getscn(3E)</td>
</tr>
<tr>
<td>elf_next(3E)</td>
<td>See elf_begin(3E)</td>
</tr>
<tr>
<td>elf_nextscn(3E)</td>
<td>See elf_getscn(3E)</td>
</tr>
<tr>
<td>elf_rand(3E)</td>
<td>See elf_begin(3E)</td>
</tr>
<tr>
<td>elf_rawdata(3E)</td>
<td>See elf_getdata(3E)</td>
</tr>
<tr>
<td>elf_rawfile(3E)</td>
<td>retrieve uninterpreted file contents</td>
</tr>
<tr>
<td>elf_strptr(3E)</td>
<td>make a string pointer</td>
</tr>
<tr>
<td>elf_update(3E)</td>
<td>update an ELF descriptor</td>
</tr>
<tr>
<td>elf_version(3E)</td>
<td>coordinate ELF library and application versions</td>
</tr>
<tr>
<td>encrypt(3C)</td>
<td>encoding function</td>
</tr>
<tr>
<td>end(3C)</td>
<td>last locations in program</td>
</tr>
<tr>
<td>_end(3C)</td>
<td>See end(3C)</td>
</tr>
<tr>
<td>endac(3)</td>
<td>See getacinfo(3)</td>
</tr>
<tr>
<td>endauclass(3)</td>
<td>See getauclassent(3)</td>
</tr>
<tr>
<td>endauevent(3)</td>
<td>See getauevent(3)</td>
</tr>
<tr>
<td>endauuser(3)</td>
<td>See getauusrernam(3)</td>
</tr>
<tr>
<td>endgrent(3C)</td>
<td>See getgrentnam(3)</td>
</tr>
<tr>
<td>endhostent(3N)</td>
<td>See gethostbyname(3N)</td>
</tr>
<tr>
<td>endhostent(3XN)</td>
<td>network host database functions</td>
</tr>
<tr>
<td>endnetconfig(3N)</td>
<td>See getnetconfig(3N)</td>
</tr>
<tr>
<td>endnetent(3N)</td>
<td>See getnetbyname(3N)</td>
</tr>
<tr>
<td>endnetent(3XN)</td>
<td>network database functions</td>
</tr>
<tr>
<td>endnetgrent(3N)</td>
<td>See getnetgrent(3N)</td>
</tr>
<tr>
<td>endnetpath(3N)</td>
<td>See getnetpath(3N)</td>
</tr>
<tr>
<td>endprotoent(3N)</td>
<td>See getprotobynam(3N)</td>
</tr>
<tr>
<td>endprotoent(3XN)</td>
<td>network protocol database functions</td>
</tr>
<tr>
<td>endpwent(3C)</td>
<td>See getpwnam(3C)</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>endrpcent(3N)</td>
<td>See getrpcbyname(3N)</td>
</tr>
<tr>
<td>endservent(3N)</td>
<td>See getservbyname(3N)</td>
</tr>
<tr>
<td>endservent(3XN)</td>
<td>network services database functions</td>
</tr>
<tr>
<td>endspent(3C)</td>
<td>See getspnam(3C)</td>
</tr>
<tr>
<td>endusershell(3C)</td>
<td>See getusershell(3C)</td>
</tr>
<tr>
<td>endutent(3C)</td>
<td>See getutent(3C)</td>
</tr>
<tr>
<td>endutxent(3C)</td>
<td>See getutxent(3C)</td>
</tr>
<tr>
<td>endwin(3X)</td>
<td>See curs_initscr(3X)</td>
</tr>
<tr>
<td>endwin(3XC)</td>
<td>restore initial terminal environment</td>
</tr>
<tr>
<td>erand48(3C)</td>
<td>See drand48(3C)</td>
</tr>
<tr>
<td>erase(3)</td>
<td>See plot(3)</td>
</tr>
<tr>
<td>erase(3X)</td>
<td>See curs_clear(3X)</td>
</tr>
<tr>
<td>erase(3XC)</td>
<td>See clear(3XC)</td>
</tr>
<tr>
<td>eraserechar(3X)</td>
<td>See curs_termattrs(3X)</td>
</tr>
<tr>
<td>erasechar(3XC)</td>
<td>return current ERASE or KILL characters</td>
</tr>
<tr>
<td>erasewchar(3XC)</td>
<td>See erasechar(3XC)</td>
</tr>
<tr>
<td>erf(3M)</td>
<td>error and complementary error functions</td>
</tr>
<tr>
<td>erfc(3M)</td>
<td>See erf(3M)</td>
</tr>
<tr>
<td>errno(3C)</td>
<td>See perror(3C)</td>
</tr>
<tr>
<td>_etext(3C)</td>
<td>See end(3C)</td>
</tr>
<tr>
<td>etext(3C)</td>
<td>See end(3C)</td>
</tr>
<tr>
<td>ether_aton(3N)</td>
<td>See ethers(3N)</td>
</tr>
<tr>
<td>ether_hostton(3N)</td>
<td>See ethers(3N)</td>
</tr>
<tr>
<td>ether_line(3N)</td>
<td>See ethers(3N)</td>
</tr>
<tr>
<td>ether_ntoa(3N)</td>
<td>See ethers(3N)</td>
</tr>
<tr>
<td>ether_ntohost(3N)</td>
<td>See ethers(3N)</td>
</tr>
<tr>
<td>ethers(3N)</td>
<td>Ethernet address mapping operations</td>
</tr>
<tr>
<td>euccol(3C)</td>
<td>See euclen(3C)</td>
</tr>
<tr>
<td>euclen(3C)</td>
<td>get byte length and display width of EUC characters</td>
</tr>
<tr>
<td>eucsccol(3C)</td>
<td>See euclen(3C)</td>
</tr>
<tr>
<td>exit(3C)</td>
<td>terminate process</td>
</tr>
<tr>
<td>_exithandle(3C)</td>
<td>See exit(3C)</td>
</tr>
<tr>
<td>exp(3M)</td>
<td>exponential function</td>
</tr>
<tr>
<td>expm1(3M)</td>
<td>computes exponential functions</td>
</tr>
</tbody>
</table>
extended_to_decimal(3) See floating_to_decimal(3)
fabs(3M) absolute value function
fattach(3C) attach a STREAMS-based file descriptor to an object in the file system name space
fclose(3S) close a stream
fconvert(3) See econvert(3)
fcvt(3) See econvert(3)
fcvt(3C) See ecvt(3C)
fdataasync(3R) synchronize a file’s data
FD_CLR(3C) See select(3C)
fdetach(3C) detach a name from a STREAMS-based file descriptor
FD_ISSET(3C) See select(3C)
fdopen(3S) associate a stream with a file descriptor
FD_SET(3C) See select(3C)
FD_ZERO(3C) See select(3C)
feof(3S) See ferror(3S)
ferror(3S) stream status inquiries
fetch(3B) See dbm(3B)
fflush(3S) flush a stream
ffs(3C) find first set bit
fgetc(3S) See getc(3S)
fgetgrent(3C) See getgrent(3C)
fgetgrent_r(3C) See getgrent(3C)
fgetpos(3S) get current file position information
fgetpwent(3C) See getpwent(3C)
fgetpwent_r(3C) See getpwent(3C)
fgets(3S) See gets(3S)
fgetspent(3C) See getspent(3C)
fgetspent_r(3C) See getspent(3C)
fgetwc(3S) get a wide-character code from a stream
fgetws(3S) See getws(3S)
field_arg(3X) See form_field_validation(3X)
field_back(3X) See form_field_attributes(3X)
field_buffer(3X) See form_field_buffer(3X)
field_count(3X) See form_field(3X)
field_fore(3X)  See form_field_attributes(3X)
field_index(3X)  See form_page(3X)
field_info(3X)  See form_field_info(3X)
field_init(3X)  See form_hook(3X)
field_just(3X)  See form_field_just(3X)
field_opts(3X)  See form_field_opts(3X)
field_opts_off(3X)  See form_field_opts(3X)
field_opts_on(3X)  See form_field_opts(3X)
field_pad(3X)  See form_field_attributes(3X)
field_status(3X)  See form_field_buffer(3X)
field_term(3X)  See form_hook(3X)
field_type(3X)  See form_field_validation(3X)
field_userptr(3X)  See form_field_userptr(3X)
fileno(3S)  See ferror(3S)
file_to_decimal(3)  See string_to_decimal(3)
filter(3X)  See curs_util(3X)
filter(3XC)  disable use of certain terminal capabilities
finite(3C)  See isnan(3C)
firstkey(3B)  See dbm(3B)
flash(3X)  See curs_beep(3X)
flash(3XC)  See beep(3XC)
floating_to_decimal(3)  convert floating-point value to decimal record
flock(3B)  apply or remove an advisory lock on an open file
flockfile(3S)  acquire and release stream lock
floor(3M)  floor function
flushinp(3X)  See curs_util(3X)
flushinp(3XC)  discard type-ahead characters
fmod(3M)  floating-point remainder value function
fmtmsg(3C)  display a message on stderr or system console
fn_attr_bind(3N)  bind a reference to a name and associate attributes with named object
fn_attr_create_subcontext(3N)  create a subcontext in a context and associate attributes with newly created
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fn_attr_ext_search</td>
<td>search for names in the specified context(s) whose attributes satisfy the filter</td>
</tr>
<tr>
<td>fn_attr_get</td>
<td>return specified attribute associated with name</td>
</tr>
<tr>
<td>fn_attr_get_ids</td>
<td>get a list of the identifiers of all attributes associated with named object</td>
</tr>
<tr>
<td>fn_attr_get_values</td>
<td>return values of an attribute</td>
</tr>
<tr>
<td>fn_attribute_add</td>
<td>See FN_attribute_t</td>
</tr>
<tr>
<td>fn_attribute_assign</td>
<td>See FN_attribute_t</td>
</tr>
<tr>
<td>fn_attribute_copy</td>
<td>See FN_attribute_t</td>
</tr>
<tr>
<td>fn_attribute_create</td>
<td>See FN_attribute_t</td>
</tr>
<tr>
<td>fn_attribute_destroy</td>
<td>See FN_attribute_t</td>
</tr>
<tr>
<td>fn_attribute_first</td>
<td>See FN_attribute_t</td>
</tr>
<tr>
<td>fn_attribute_identifier</td>
<td>See FN_attribute_t</td>
</tr>
<tr>
<td>fn_attribute_next</td>
<td>See FN_attribute_t</td>
</tr>
<tr>
<td>fn_attribute_remove</td>
<td>See FN_attribute_t</td>
</tr>
<tr>
<td>fn_attribute_syntax</td>
<td>See FN_attribute_t</td>
</tr>
<tr>
<td>FN_attribute_t</td>
<td>an XFN attribute</td>
</tr>
<tr>
<td>fn_attribute_valuecount</td>
<td>See FN_attribute_t</td>
</tr>
<tr>
<td>fn_attr_modify</td>
<td>modify specified attribute associated with name</td>
</tr>
<tr>
<td>fn_attmodlist_add</td>
<td>See FN_attmodlist_t</td>
</tr>
<tr>
<td>fn_attmodlist_assign</td>
<td>See FN_attmodlist_t</td>
</tr>
<tr>
<td>fn_attmodlist_copy</td>
<td>See FN_attmodlist_t</td>
</tr>
<tr>
<td>fn_attmodlist_count</td>
<td>See FN_attmodlist_t</td>
</tr>
<tr>
<td>fn_attmodlist_create</td>
<td>See FN_attmodlist_t</td>
</tr>
<tr>
<td>fn_attmodlist_destroy</td>
<td>See FN_attmodlist_t</td>
</tr>
<tr>
<td>fn_attmodlist_first</td>
<td>See FN_attmodlist_t</td>
</tr>
<tr>
<td>fn_attmodlist_next</td>
<td>See FN_attmodlist_t</td>
</tr>
<tr>
<td>FN_attmodlist_t</td>
<td>a list of attribute modifications</td>
</tr>
<tr>
<td>fn_attre_multi_get</td>
<td>return multiple attributes associated with named object</td>
</tr>
<tr>
<td>fn_attre_multi_modify</td>
<td>modify multiple attributes associated with named object</td>
</tr>
<tr>
<td>fn_attr_search</td>
<td>search for the atomic name of objects with modified attributes</td>
</tr>
</tbody>
</table>
fn_attrset_add(3N) the specified attributes in a single context
fn_attrset_assign(3N) See FN_attrset_t(3N)
fn_attrset_copy(3N) See FN_attrset_t(3N)
fn_attrset_count(3N) See FN_attrset_t(3N)
fn_attrset_create(3N) See FN_attrset_t(3N)
fn_attrset_destroy(3N) See FN_attrset_t(3N)
fn_attrset_first(3N) See FN_attrset_t(3N)
fn_attrset_get(3N) See FN_attrset_t(3N)
fn_attrset_next(3N) See FN_attrset_t(3N)
fn_attrset_remove(3N) See FN_attrset_t(3N)
FN_attrset_t(3N)
FN_attrvalue_t(3N) a set of XFN attributes
fn_bindinglist_destroy(3N) See fn_ctx_list_bindings(3N)
fn_bindinglist_next(3N) See fn_ctx_list_bindings(3N)
FN_bindinglist_t(3N) See fn_ctx_list_bindings(3N)
fn_composite_name_append_comp(3N) See FN_composite_name_t(3N)
fn_composite_name_append_name(3N) See FN_composite_name_t(3N)
fn_composite_name_assign(3N) See FN_composite_name_t(3N)
fn_composite_name_copy(3N) See FN_composite_name_t(3N)
fn_composite_name_count(3N) See FN_composite_name_t(3N)
fn_composite_name_create(3N) See FN_composite_name_t(3N)
fn_composite_name_delete_comp(3N) See FN_composite_name_t(3N)
fn_composite_name_destroy(3N) See FN_composite_name_t(3N)
fn_composite_name_first(3N) See FN_composite_name_t(3N)
fn_composite_name_from_str(3N) See FN_composite_name_t(3N)
fn_composite_name_from_string(3N) See FN_composite_name_t(3N)
fn_composite_name_insert_comp(3N) See FN_composite_name_t(3N)
fn_composite_name_insert_name(3N) See FN_composite_name_t(3N)
fn_composite_name_is_empty(3N) See FN_composite_name_t(3N)
fn_composite_name_is_equal(3N) See FN_composite_name_t(3N)
fn_composite_name_is_prefix(3N) See FN_composite_name_t(3N)
fn_composite_name_is_suffix(3N) See FN_composite_name_t(3N)
fn_composite_name_last(3N) See FN_composite_name_t(3N)
fn_composite_name_next(3N) See FN_composite_name_t(3N)
C Library Functions

fn_composite_name_prefix(3N)
fn_composite_name_prepend_comp(3N)
fn_composite_name_prepend_name(3N)
fn_composite_name_prev(3N)
fn_composite_name_suffix(3N)
FN_composite_name_t(3N)

fn_composite_name_prepend_comp(3N) See FN_composite_name_t(3N)
fn_composite_name_prepend_name(3N) See FN_composite_name_t(3N)
fn_composite_name_prev(3N) See FN_composite_name_t(3N)
fn_composite_name_suffix(3N) See FN_composite_name_t(3N)

FN_composite_name_t(3N) a sequence of component names spanning multiple naming systems

fn_compound_name_append_comp(3N)
fn_compound_name_assign(3N)
fn_compound_name_copy(3N)
fn_compound_name_count(3N)
fn_compound_name_delete_all(3N)
fn_compound_name_delete_comp(3N)
fn_compound_name_destroy(3N)
fn_compound_name_first(3N)
fn_compound_name_from_syntax_attrs(3N)
fn_compound_name_get_syntax_attrs(3N)
fn_compound_name_insert_comp(3N)
fn_compound_name_is_empty(3N)
fn_compound_name_is_equal(3N)
fn_compound_name_is_prefix(3N)
fn_compound_name_is_suffix(3N)
fn_compound_name_last(3N)
fn_compound_name_next(3N)
fn_compound_name_prefix(3N)
fn_compound_name_prepend_comp(3N)
fn_compound_name_prev(3N)
fn_compound_name_suffix(3N)
FN_compound_name_t(3N)

fn_ctx_bind(3N)
fn_ctx_create_subcontext(3N)
fn_ctx_destroy_subcontext(3N)
fn_ctx_equivalent_name(3N)
fn_ctx_get_ref(3N)

See FN_compound_name_t(3N)
See FN_compound_name_t(3N)
See FN_compound_name_t(3N)
See FN_compound_name_t(3N)
See FN_compound_name_t(3N)
See FN_compound_name_t(3N)
See FN_compound_name_t(3N)
See FN_compound_name_t(3N)
See FN_compound_name_t(3N)
See FN_compound_name_t(3N)
See FN_compound_name_t(3N)
See FN_compound_name_t(3N)
See FN_compound_name_t(3N)
See FN_compound_name_t(3N)
See FN_compound_name_t(3N)
See FN_compound_name_t(3N)
See FN_compound_name_t(3N)
See FN_compound_name_t(3N)
See FN_compound_name_t(3N)
See FN_compound_name_t(3N)

an XFN compound name
bind a reference to a name
create a subcontext in a context
destroy the named context and remove its binding from the parent context
construct an equivalent name in same context
return a context’s reference

modified 4 Apr 1997

SunOS 5.6

3-31
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fn_ctx_get_syntax_attrs</td>
<td>return syntax attributes associated with named context</td>
</tr>
<tr>
<td>fn_ctx_handle_destroy</td>
<td>release storage associated with context handle</td>
</tr>
<tr>
<td>fn_ctx_handle_from_initial</td>
<td>return a handle to the Initial Context</td>
</tr>
<tr>
<td>fn_ctx_handle_from_ref</td>
<td>construct a handle to a context object using the given reference</td>
</tr>
<tr>
<td>fn_ctx_list_bindings</td>
<td>list the atomic names and references bound in a context</td>
</tr>
<tr>
<td>fn_ctx_list_names</td>
<td>list the atomic names bound in a context</td>
</tr>
<tr>
<td>fn_ctx_lookup</td>
<td>look up name in context</td>
</tr>
<tr>
<td>fn_ctx_lookup_link</td>
<td>look up the link reference bound to a name</td>
</tr>
<tr>
<td>fn_ctx_rename</td>
<td>rename the name of a binding</td>
</tr>
<tr>
<td>FN_ctx_t</td>
<td>an XFN context</td>
</tr>
<tr>
<td>fn_ctx_unbind</td>
<td>unbind a name from a context</td>
</tr>
<tr>
<td>fn_ext_searchlist_destroy</td>
<td>See fn_attr_ext_search</td>
</tr>
<tr>
<td>fn_ext_searchlist_next</td>
<td>See fn_attr_ext_search</td>
</tr>
<tr>
<td>FN_ext_searchlist_t</td>
<td>See fn_attr_ext_search</td>
</tr>
<tr>
<td>FN_identifier_t</td>
<td>an XFN identifier</td>
</tr>
<tr>
<td>fnmatch</td>
<td>match filename or path name</td>
</tr>
<tr>
<td>fn_multigetlist_destroy</td>
<td>See fn_attr_multi_get</td>
</tr>
<tr>
<td>fn_multigetlist_next</td>
<td>See fn_attr_multi_get</td>
</tr>
<tr>
<td>FN_multigetlist_t</td>
<td>See fn_attr_multi_get</td>
</tr>
<tr>
<td>fn_namelist_destroy</td>
<td>See fn_ctx_list_names</td>
</tr>
<tr>
<td>fn_namelist_next</td>
<td>See fn_ctx_list_names</td>
</tr>
<tr>
<td>FN_namelist_t</td>
<td>See fn_ctx_list_names</td>
</tr>
<tr>
<td>fn_ref_addr_assign</td>
<td>See FN_ref_addr_t</td>
</tr>
<tr>
<td>fn_ref_addr_copy</td>
<td>See FN_ref_addr_t</td>
</tr>
<tr>
<td>fn_ref_addrcount</td>
<td>See FN_ref_addr_t</td>
</tr>
<tr>
<td>fn_ref_addr_create</td>
<td>See FN_ref_addr_t</td>
</tr>
<tr>
<td>fn_ref_addr_data</td>
<td>See FN_ref_addr_t</td>
</tr>
<tr>
<td>fn_ref_addr_description</td>
<td>See FN_ref_addr_t</td>
</tr>
<tr>
<td>fn_ref_addr_destroy</td>
<td>See FN_ref_addr_t</td>
</tr>
<tr>
<td>fn_ref_addr_length</td>
<td>See FN_ref_addr_t</td>
</tr>
<tr>
<td>FN_ref_addr_t</td>
<td>an address in an XFN reference</td>
</tr>
</tbody>
</table>

3-32 SunOS 5.6 modified 4 Apr 1997
fn_ref_addr_type(3N) See FN_ref_addr_t(3N)
fn_ref_append_addr(3N) See FN_ref_t(3N)
fn_ref_assign(3N) See FN_ref_t(3N)
fn_ref_copy(3N) See FN_ref_t(3N)
fn_ref_create(3N) See FN_ref_t(3N)
fn_ref_create_link(3N) See FN_ref_t(3N)
fn_ref_delete_addr(3N) See FN_ref_t(3N)
fn_ref_delete_all(3N) See FN_ref_t(3N)
fn_ref_description(3N) See FN_ref_t(3N)
fn_ref_destroy(3N) See FN_ref_t(3N)
fn_ref_first(3N) See FN_ref_t(3N)
fn_ref_insert_addr(3N) See FN_ref_t(3N)
fn_ref_is_link(3N) See FN_ref_t(3N)
fn_ref_link_name(3N) See FN_ref_t(3N)
fn_ref_next(3N) See FN_ref_t(3N)
fn_ref_prepend_addr(3N) See FN_ref_t(3N)
FN_ref_t(3N) an XFN reference
fn_ref_type(3N) See FN_ref_t(3N)
fn_search_control_assign(3N) See FN_search_control_t(3N)
fn_search_control_copy(3N) See FN_search_control_t(3N)
fn_search_control_create(3N) See FN_search_control_t(3N)
fn_search_control_destroy(3N) See FN_search_control_t(3N)
fn_search_control_follow_links(3N) See FN_search_control_t(3N)
fn_search_control_max_names(3N) See FN_search_control_t(3N)
fn_search_control_return_attr_ids(3N) See FN_search_control_t(3N)
fn_search_control_return_ref(3N) See FN_search_control_t(3N)
fn_search_control_scope(3N) See FN_search_control_t(3N)
FN_search_control_t(3N) options for attribute search
fn_search_filter_arguments(3N) See FN_search_filter_t(3N)
fn_search_filter_assign(3N) See FN_search_filter_t(3N)
fn_search_filter_copy(3N) See FN_search_filter_t(3N)
fn_search_filter_create(3N) See FN_search_filter_t(3N)
fn_search_filter_destroy(3N) See FN_search_filter_t(3N)
fn_search_filter_expression(3N) See FN_search_filter_t(3N)
FN_search_filter_t(3N) filter expression for attribute search
fn_searchlist_destroy(3N)  See fn_attr_search(3N)
fn_searchlist_next(3N)  See fn_attr_search(3N)
FN_searchlist_t(3N)  See fn_attr_search(3N)
fn_status_advance_by_name(3N)  See FN_status_t(3N)
fn_status_append_remaining_name(3N)  See FN_status_t(3N)
fn_status_append_resolved_name(3N)  See FN_status_t(3N)
fn_status_assign(3N)  See FN_status_t(3N)
fn_status_code(3N)  See FN_status_t(3N)
fn_status_copy(3N)  See FN_status_t(3N)
fn_status_create(3N)  See FN_status_t(3N)
fn_status_description(3N)  See FN_status_t(3N)
fn_status_destroy(3N)  See FN_status_t(3N)
fn_status_diagnostic_message(3N)  See FN_status_t(3N)
fn_status_is_success(3N)  See FN_status_t(3N)
fn_status_link_code(3N)  See FN_status_t(3N)
fn_status_link_diagnostic_message(3N)  See FN_status_t(3N)
fn_status_link_remaining_name(3N)  See FN_status_t(3N)
fn_status_link_resolved_name(3N)  See FN_status_t(3N)
fn_status_link_resolved_ref(3N)  See FN_status_t(3N)
fn_status_remaining_name(3N)  See FN_status_t(3N)
fn_status_resolved_name(3N)  See FN_status_t(3N)
fn_status_resolved_ref(3N)  See FN_status_t(3N)
fn_status_set(3N)  See FN_status_t(3N)
fn_status_set_code(3N)  See FN_status_t(3N)
fn_status_set_diagnostic_message(3N)  See FN_status_t(3N)
fn_status_set_link_code(3N)  See FN_status_t(3N)
fn_status_set_link_diagnostic_message(3N)  See FN_status_t(3N)
fn_status_set_link_remaining_name(3N)  See FN_status_t(3N)
fn_status_set_link_resolved_name(3N)  See FN_status_t(3N)
fn_status_set_link_resolved_ref(3N)  See FN_status_t(3N)
fn_status_set_remaining_name(3N)  See FN_status_t(3N)
fn_status_set_resolved_name(3N)  See FN_status_t(3N)
fn_status_set_resolved_ref(3N)  See FN_status_t(3N)
fn_status_set_success(3N)  See FN_status_t(3N)
FN_status_t(3N)  an XFN status object
fn_string_assign(3N) See FN_string_t(3N)
fn_string_bytecount(3N) See FN_string_t(3N)
fn_string_charcount(3N) See FN_string_t(3N)
fn_string_code_set(3N) See FN_string_t(3N)
fn_string_compare(3N) See FN_string_t(3N)
fn_string_compare_substring(3N) See FN_string_t(3N)
fn_string_contents(3N) See FN_string_t(3N)
fn_string_copy(3N) See FN_string_t(3N)
fn_string_create(3N) See FN_string_t(3N)
fn_string_destroy(3N) See FN_string_t(3N)
fn_string_from_composite_name(3N) See FN_composite_name_t(3N)
fn_string_from_compound_name(3N) See FN_compound_name_t(3N)
fn_string_from_contents(3N) See FN_string_t(3N)
fn_string_from_str(3N) See FN_string_t(3N)
fn_string_from_strings(3N) See FN_string_t(3N)
fn_string_from_str_n(3N) See FN_string_t(3N)
fn_string_from_substring(3N) See FN_string_t(3N)
fn_string_is_empty(3N) See FN_string_t(3N)
fn_string_next_substring(3N) See FN_string_t(3N)
fn_string_prev_substring(3N) See FN_string_t(3N)
fn_string_str(3N) See FN_string_t(3N)
FN_string_t(3N) a character string
fn_value_list_destroy(3N) See fn_attr_get_values(3N)
fn_value_list_next(3N) See fn_attr_get_values(3N)
FN_value_list_t(3N) See fn_attr_get_values(3N)
fopen(3B) open a stream
fopen(3S) open a stream
form_cursor(3X) position forms window cursor
form_data(3X) tell if forms field has off-screen data ahead or behind
form_driver(3X) command processor for the forms subsystem
form_field(3X) connect fields to forms
form_field_attributes(3X) format the general display attributes of forms
form_field_buffer(3X) set and get forms field attributes

modified 4 Apr 1997 SunOS 5.6 3-35
form_field_info(3X)
form_field_just(3X)
form_field_new(3X)
form_field_opts(3X)
form_fields(3X)
form_fieldtype(3X)
form_field_userptr(3X)
form_field_validation(3X)
form_hook(3X)
form_init(3X)
form_new(3X)
form_new_page(3X)
form_opts(3X)
form_opts_off(3X)
form_opts_on(3X)
form_page(3X)
form_post(3X)
forms(3X)
form_sub(3X)
form_term(3X)
form_userptr(3X)
form_win(3X)
fpclass(3C)
fpgetmask(3C)
fpgetround(3C)
fpgetsticky(3C)
fprintf(3B)
fprintf(3S)
fpsetmask(3C)
fpsetround(3C)
fpsetsticky(3C)
fputc(3S)
fputs(3S)

get forms field characteristics
format the general appearance of forms
create and destroy forms fields
forms field option routines
See form_field(3X)
forms fieldtype routines
associate application data with forms
forms field data type validation
assign application-specific routines for invocation by forms
See form_hook(3X)
create and destroy forms
forms pagination
forms option routines
See form_opts(3X)
See form_opts(3X)
set forms current page and field
write or erase forms from associated subwindows
character based forms package
See form_win(3X)
See form_hook(3X)
associate application data with forms
forms window and subwindow association routines
See isnan(3C)
See fpgetround(3C)
IEEE floating-point environment control
See fpgetround(3C)
See printf(3B)
See printf(3S)
See fpgetround(3C)
See fpgetround(3C)
See fpgetround(3C)
See putc(3S)
See puts(3S)
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fputwc(3S)</td>
<td>put wide-character code on a stream</td>
</tr>
<tr>
<td>fputws(3S)</td>
<td>put wide character string on a stream</td>
</tr>
<tr>
<td>fread(3S)</td>
<td>buffered binary input/output</td>
</tr>
<tr>
<td>free(3C)</td>
<td>See malloc(3C)</td>
</tr>
<tr>
<td>free(3X)</td>
<td>See bsdmalloc(3X)</td>
</tr>
<tr>
<td>free(3X)</td>
<td>See malloc(3X)</td>
</tr>
<tr>
<td>free(3X)</td>
<td>See mapmalloc(3X)</td>
</tr>
<tr>
<td>free(3X)</td>
<td>See watchmalloc(3X)</td>
</tr>
<tr>
<td>free_field(3X)</td>
<td>See form_field_new(3X)</td>
</tr>
<tr>
<td>free_fieldtype(3X)</td>
<td>See form_fieldtype(3X)</td>
</tr>
<tr>
<td>free_form(3X)</td>
<td>See form_new(3X)</td>
</tr>
<tr>
<td>free_item(3X)</td>
<td>See menu_item_new(3X)</td>
</tr>
<tr>
<td>free_menu(3X)</td>
<td>See menu_new(3X)</td>
</tr>
<tr>
<td>freenetconfig(3N)</td>
<td>See getnetconfig(3N)</td>
</tr>
<tr>
<td>freopen(3B)</td>
<td>See fopen(3B)</td>
</tr>
<tr>
<td>freopen(3S)</td>
<td>open a stream</td>
</tr>
<tr>
<td>frexp(3C)</td>
<td>extract mantissa and exponent from double precision number</td>
</tr>
<tr>
<td>fscanf(3S)</td>
<td>See scanf(3S)</td>
</tr>
<tr>
<td>fseek(3S)</td>
<td>reposition a file-position indicator in a stream</td>
</tr>
<tr>
<td>fseeko(3S)</td>
<td>See fseek(3S)</td>
</tr>
<tr>
<td>fsetpos(3S)</td>
<td>reposition a file pointer in a stream</td>
</tr>
<tr>
<td>fsync(3C)</td>
<td>synchronize a file's in-memory state with that on the physical medium</td>
</tr>
<tr>
<td>ftell(3S)</td>
<td>return a file offset in a stream</td>
</tr>
<tr>
<td>ftello(3S)</td>
<td>See ftell(3S)</td>
</tr>
<tr>
<td>ftm(3C)</td>
<td>get date and time</td>
</tr>
<tr>
<td>ftok(3C)</td>
<td>generate an IPC key</td>
</tr>
<tr>
<td>ftruncate(3C)</td>
<td>See truncate(3C)</td>
</tr>
<tr>
<td>ftrylockfile(3S)</td>
<td>See flockfile(3S)</td>
</tr>
<tr>
<td>ftw(3C)</td>
<td>walk a file tree</td>
</tr>
<tr>
<td>func_to_decimal(3)</td>
<td>See string_to_decimal(3)</td>
</tr>
<tr>
<td>funlockfile(3S)</td>
<td>See flockfile(3S)</td>
</tr>
<tr>
<td>fwrite(3S)</td>
<td>See fread(3S)</td>
</tr>
<tr>
<td>gamma(3M)</td>
<td>See lgamma(3M)</td>
</tr>
</tbody>
</table>

modified 4 Apr 1997
gamma_r(3M)  See lgamma(3M)
gconvert(3)  See econvert(3)
gcvtn(3)  See econvert(3)
gcvtn(3C)  See ecvt(3C)
getacdir(3)  See getacinfo(3)
getacfg(3)  See getacinfo(3)
getacinfo(3)  get audit control file information
getacmin(3)  See getacinfo(3)
getacna(3)  See getacinfo(3)
getauclassevent(3)  get audit_class entry
getauclassevent_r(3)  See getauclassevent(3)
getauclasseventnam(3)  See getauclassevent(3)
getauclasseventnam_r(3)  See getauclassevent(3)
getauditflags(3)  convert audit flag specifications
getauditflagsbin(3)  See getauditflags(3)
getauditflagschar(3)  See getauditflags(3)
getauclassevent(3)  get audit_event entry
getauclassevent_r(3)  See getauclassevent(3)
getauclasseventnam(3)  See getauclassevent(3)
getauclasseventnam_r(3)  See getauclassevent(3)
getauclasseventnum(3)  See getauclassevent(3)
getauclasseventnum_r(3)  See getauclassevent(3)
getauuserent(3)  See getauusernam(3)
getauusernam(3)  get audit_user entry
getbegyx(3X)  See curs_getyx(3X)
getbegyx(3XC)  get cursor or window coordinates
getbkgrnd(3XC)  See bkgrnd(3XC)
getc(3S)  get character or word from a stream
getcchar(3XC)  get a wide character string (with rendition) from a cchar_t
getch(3X)  See curs_getch(3X)
getch(3XC)  get a single-byte character from terminal
getchar(3S)  See getc(3S)
getchar_unlocked(3S)  See getc(3S)
getc_unlocked(3S)  See getc(3S)
getcwd(3C)  get pathname of current working directory
getdate(3C)  convert user format date and time
getdtablesized(3C)  get the file descriptor table size
getenv(3C)  return value for environment name
getexecname(3C)  return pathname of executable
getfauditflags(3)  generates the process audit state
getgrent(3C)  See getgrnam(3C)
getgrent_r(3C)  See getgrnam(3C)
getgrgid(3C)  See getgrnam(3C)
getgrgid_r(3C)  See getgrnam(3C)
getgrnam(3C)  get group entry
getgrnam_r(3C)  See getgrnam(3C)
gethostbyaddr(3N)  See gethostbyname(3N)
gethostbyaddr(3XN)  See endhostent(3XN)
gethostbyaddr_r(3N)  See gethostbyname(3N)
gethostbyname(3N)  get network host entry
gethostbyname(3XN)  See endhostent(3XN)
gethostbyname_r(3N)  See gethostbyname(3N)
gethostent(3N)  See gethostbyname(3N)
gethostent(3XN)  See endhostent(3XN)
gethostent_r(3N)  See gethostbyname(3N)
gethostid(3C)  get unique identifier of current host
gethostname(3C)  get or set name of current host
gethostname(3XN)  get name of current host
gethrtme(3C)  get high resolution time
gethrtime(3C)  See gethrtime(3C)
getlogin(3C)  get login name
getlogin_r(3C)  See getlogin(3C)
getmaxyx(3X)  See curs_getyx(3X)
getmaxyx(3XC)  See getbegyx(3XC)
getmntany(3C)  See getmntent(3C)
getmntent(3C)  get mnttab file information
get_myaddress(3N)  See rpc_soc(3N)
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getnetbyaddr(3N)</td>
<td>See getnetbyname(3N)</td>
</tr>
<tr>
<td>getnetbyaddr(3XN)</td>
<td>See endnetent(3XN)</td>
</tr>
<tr>
<td>getnetbyaddr_r(3N)</td>
<td>See getnetbyname(3N)</td>
</tr>
<tr>
<td>getnetbyname(3N)</td>
<td>get network entry</td>
</tr>
<tr>
<td>getnetbyname(3XN)</td>
<td>See endnetent(3XN)</td>
</tr>
<tr>
<td>getnetbyname_r(3N)</td>
<td>See getnetbyname(3N)</td>
</tr>
<tr>
<td>getnetconfig(3N)</td>
<td>get network configuration database entry</td>
</tr>
<tr>
<td>getnetconfigent(3N)</td>
<td>See getnetconfig(3N)</td>
</tr>
<tr>
<td>getnetent(3N)</td>
<td>See getnetbyname(3N)</td>
</tr>
<tr>
<td>getnetent(3XN)</td>
<td>See endnetent(3XN)</td>
</tr>
<tr>
<td>getnetent_r(3N)</td>
<td>See getnetbyname(3N)</td>
</tr>
<tr>
<td>getnetgrent(3N)</td>
<td>get network group entry</td>
</tr>
<tr>
<td>getnetgrent_r(3N)</td>
<td>See getnetgrent(3N)</td>
</tr>
<tr>
<td>getnetname(3N)</td>
<td>See secure_rpc(3N)</td>
</tr>
<tr>
<td>getnetpath(3N)</td>
<td>get /etc/netconfig entry corresponding to NETPATH component</td>
</tr>
<tr>
<td>getnstr(3XC)</td>
<td>get a multibyte character string from terminal</td>
</tr>
<tr>
<td>getn_wstr(3X)</td>
<td>See curs_getwstr(3X)</td>
</tr>
<tr>
<td>getnwstr(3X)</td>
<td>get a wide character string from terminal</td>
</tr>
<tr>
<td>getopt(3C)</td>
<td>get option letter from argument vector</td>
</tr>
<tr>
<td>getpagesize(3C)</td>
<td>get system page size</td>
</tr>
<tr>
<td>getparyx(3X)</td>
<td>See curs_getyx(3X)</td>
</tr>
<tr>
<td>getparyx(3XC)</td>
<td>See getbegyx(3XC)</td>
</tr>
<tr>
<td>getpass(3C)</td>
<td>read a string of characters without echo</td>
</tr>
<tr>
<td>getpassphrase(3C)</td>
<td>See getpass(3C)</td>
</tr>
<tr>
<td>getpeername(3N)</td>
<td>get name of connected peer</td>
</tr>
<tr>
<td>getpeername(3XN)</td>
<td>get the name of the peer socket</td>
</tr>
<tr>
<td>getpriority(3C)</td>
<td>get or set process scheduling priority</td>
</tr>
<tr>
<td>getprotobyname(3N)</td>
<td>get protocol entry</td>
</tr>
<tr>
<td>getprotobyname(3XN)</td>
<td>See endprotoent(3XN)</td>
</tr>
<tr>
<td>getprotobyname_r(3N)</td>
<td>See getprotobyname(3N)</td>
</tr>
<tr>
<td>getprotobynumber(3N)</td>
<td>See getprotobyname(3N)</td>
</tr>
<tr>
<td>getprotobynumber_r(3N)</td>
<td>See endprotoent(3XN)</td>
</tr>
<tr>
<td>getprotoent(3N)</td>
<td>See getprotobyname(3N)</td>
</tr>
</tbody>
</table>
getprotoent(3XN)
getprotoent_r(3N)
getpublickey(3N)
getpw(3C)
getpwent(3C)
getpwent_r(3C)
getpwnam(3C)
getpwnam_r(3C)
getpwuid(3C)
getpwuid_r(3C)
getrpcbyname(3N)
getrpcbyname_r(3N)
getrpcbyname(3N)
getrpcbyname_r(3N)
getrpcent(3N)
getrpcent_r(3N)
getrpcport(3N)
getrusage(3C)
gets(3S)
getsecretkey(3N)
getservbyname(3N)
getservbyname(3XN)
getservbyname_r(3N)
getservbyport(3N)
getservbyport(3XN)
getservbyport_r(3N)
getservent(3N)
getservent(3XN)
getservent_r(3N)
getsockname(3N)
getsockname(3XN)
getsockopt(3N)
getsockopt(3XN)
getsockopt_r(3N)

See endprotoent(3XN)
See getprotobyname(3N)
retrieve public or secret key
get passwd entry from UID
See getpwnam(3C)
See getpwnam(3C)
get password entry
See getpwnam(3C)
See getpwnam(3C)
get RPC entry
See getrpcbyname(3N)
See getrpcbyname(3N)
See getrpcbyname(3N)
See getrpcbyname(3N)
See getrpcbyname(3N)
See getrpcbyname(3N)
See getrpcbyname(3N)
See getrpcbyname(3N)
See getrpcbyname(3N)
See getrpcbyname(3N)
See getrpcbyname(3N)
See getrpcbyname(3N)
See getrpcbyname(3N)
See getrpcbyname(3N)
get information about resource utilization
get a string from a stream
See getpublickey(3N)
get service entry
See endservent(3XN)
See getservbyname(3N)
See endservent(3XN)
See getservbyname(3N)
See endservent(3XN)
See getservbyname(3N)
See endservent(3XN)
See getservbyname(3N)
See endservent(3XN)
See getservbyname(3N)
get socket name
get the socket name
get and set options on sockets
get the socket options
See getspnam(3C)
See getspnam(3C)
getspnam(3C)  get password entry
getspnam_r(3C)  See getspnam(3C)
getstr(3X)  See curs_getstr(3X)
getstr(3XC)  See getstr(3XC)
getsubopt(3C)  parse suboptions from a string
getsyx(3X)  See curs_kernel(3X)
gettext(3C)  message handling functions
gettimeofday(3B)  get or set the date and time
gettimeofday(3C)  get or set the date and time
gettext(3C)  retrieve a text string
getusershell(3C)  get legal user shells
getutent(3C)  access utmp file entry
getutid(3C)  See getutent(3C)
getutline(3C)  See getutent(3C)
getutmp(3C)  See getutxent(3C)
getutmpx(3C)  See getutxent(3C)
getutxent(3C)  access utmpx file entry
getutxid(3C)  See getutxent(3C)
getutxline(3C)  See getutxent(3C)
getvfsany(3C)  See getvfsent(3C)
getvfsent(3C)  get vfstab file entry
getvfsfile(3C)  See getvfsent(3C)
getvfsspec(3C)  See getvfsent(3C)
getw(3S)  See getc(3S)
getwc(3S)  get wide character from a stream
getwch(3X)  See curs_getwch(3X)
get_wch(3XC)  get a wide character from terminal
getwchar(3S)  get wide character from stdin stream
getwd(3C)  get current working directory pathname
getwidth(3C)  get codeset information
getwin(3X)  See curs_util(3X)
getwin(3XC)  read a window from, and write a window to, a file
getws(3S)  convert a string of EUC characters from the stream to Process Code
getwstr(3X)  See curs_getwstr(3X)
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_wstr(3XC)</td>
<td>See getn_wstr(3XC)</td>
</tr>
<tr>
<td>getyx(3X)</td>
<td>See curs_getyx(3X)</td>
</tr>
<tr>
<td>getyx(3XC)</td>
<td>See getbegyx(3XC)</td>
</tr>
<tr>
<td>glob(3C)</td>
<td>generate path names matching a pattern</td>
</tr>
<tr>
<td>global_variables(3XC)</td>
<td>variables used for X/Open Curses</td>
</tr>
<tr>
<td>globfree(3C)</td>
<td>See glob(3C)</td>
</tr>
<tr>
<td>gmatch(3G)</td>
<td>shell global pattern matching</td>
</tr>
<tr>
<td>gmtime(3C)</td>
<td>See ctime(3C)</td>
</tr>
<tr>
<td>gmtime_r(3C)</td>
<td>See ctime(3C)</td>
</tr>
<tr>
<td>grantpt(3C)</td>
<td>grant access to the slave pseudo-terminal device</td>
</tr>
<tr>
<td>gsignal(3C)</td>
<td>See ssignal(3C)</td>
</tr>
<tr>
<td>halfdelay(3X)</td>
<td>See curs_inopts(3X)</td>
</tr>
<tr>
<td>halfdelay(3XC)</td>
<td>enable/disable half-delay mode</td>
</tr>
<tr>
<td>has_colors(3X)</td>
<td>See curs_color(3X)</td>
</tr>
<tr>
<td>has_colors(3XC)</td>
<td>See can_change_color(3XC)</td>
</tr>
<tr>
<td>has_ic(3X)</td>
<td>See curs_termattr(3X)</td>
</tr>
<tr>
<td>has_ic(3XC)</td>
<td>determine insert/delete character/line capability</td>
</tr>
<tr>
<td>has_il(3X)</td>
<td>See curs_termattr(3X)</td>
</tr>
<tr>
<td>has_il(3XC)</td>
<td>See has_ic(3XC)</td>
</tr>
<tr>
<td>hasmntopt(3C)</td>
<td>See getmntent(3C)</td>
</tr>
<tr>
<td>havedisk(3N)</td>
<td>See rstat(3N)</td>
</tr>
<tr>
<td>hcreate(3C)</td>
<td>See hsearch(3C)</td>
</tr>
<tr>
<td>hdestroy(3C)</td>
<td>See hsearch(3C)</td>
</tr>
<tr>
<td>hide_panel(3C)</td>
<td>See panel_show(3X)</td>
</tr>
<tr>
<td>hline(3XC)</td>
<td>use single-byte characters (and renditions) to draw lines</td>
</tr>
<tr>
<td>hline_set(3XC)</td>
<td>use complex characters (and renditions) to draw lines</td>
</tr>
<tr>
<td>host2netname(3N)</td>
<td>See secure_rpc(3N)</td>
</tr>
<tr>
<td>hsearch(3C)</td>
<td>manage hash search tables</td>
</tr>
<tr>
<td>htonl(3N)</td>
<td>See byteorder(3N)</td>
</tr>
<tr>
<td>htonl(3XN)</td>
<td>convert values between host and network byte order</td>
</tr>
<tr>
<td>htons(3N)</td>
<td>See byteorder(3N)</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>---------------------------------------------------------------</td>
</tr>
<tr>
<td>htons(3XN)</td>
<td>See htonl(3XN) Euclidean distance function</td>
</tr>
<tr>
<td>hypot(3M)</td>
<td>Euclidean distance function</td>
</tr>
<tr>
<td>iconv(3)</td>
<td>code conversion function</td>
</tr>
<tr>
<td>iconv_close(3)</td>
<td>code conversion deallocation function</td>
</tr>
<tr>
<td>iconv_open(3)</td>
<td>code conversion allocation function</td>
</tr>
<tr>
<td>idcok(3X)</td>
<td>See curs_outopts(3X) enable/disable hardware insert-character and delete-character features</td>
</tr>
<tr>
<td>idcok(3XC)</td>
<td>enable/disable hardware insert-character and delete-character features</td>
</tr>
<tr>
<td>idlok(3X)</td>
<td>See curs_outopts(3X)</td>
</tr>
<tr>
<td>idlok(3XC)</td>
<td>See clearok(3XC)</td>
</tr>
<tr>
<td>ilogb(3M)</td>
<td>returns an unbiased exponent</td>
</tr>
<tr>
<td>immedok(3X)</td>
<td>See curs_outopts(3X) call refresh on changes to window</td>
</tr>
<tr>
<td>immedok(3XC)</td>
<td>call refresh on changes to window</td>
</tr>
<tr>
<td>inch(3X)</td>
<td>See curs_inch(3X) return a single-byte character (with rendition)</td>
</tr>
<tr>
<td>inch(3XC)</td>
<td>return a single-byte character (with rendition)</td>
</tr>
<tr>
<td>inchnstr(3X)</td>
<td>See curs_inchstr(3X) retrieve a single-byte character string (with rendition)</td>
</tr>
<tr>
<td>inchnstr(3XC)</td>
<td>retrieve a single-byte character string (with rendition)</td>
</tr>
<tr>
<td>inchstr(3X)</td>
<td>See curs_inchstr(3X)</td>
</tr>
<tr>
<td>inchstr(3XC)</td>
<td>See inchnstr(3XC)</td>
</tr>
<tr>
<td>index(3C)</td>
<td>string operations</td>
</tr>
<tr>
<td>inet(3N)</td>
<td>Internet address manipulation</td>
</tr>
<tr>
<td>inet_addr(3N)</td>
<td>See inet(3N) Internet address manipulation</td>
</tr>
<tr>
<td>inet_addr(3XN)</td>
<td>Internet address manipulation</td>
</tr>
<tr>
<td>inet_lnaof(3N)</td>
<td>See inet(3N)</td>
</tr>
<tr>
<td>inet_lnaof(3XN)</td>
<td>See inet_addr(3XN)</td>
</tr>
<tr>
<td>inet_makeaddr(3N)</td>
<td>See inet(3N)</td>
</tr>
<tr>
<td>inet_makeaddr(3XN)</td>
<td>See inet_addr(3XN)</td>
</tr>
<tr>
<td>inet_netof(3N)</td>
<td>See inet(3N)</td>
</tr>
<tr>
<td>inet_netof(3XN)</td>
<td>See inet_addr(3XN)</td>
</tr>
<tr>
<td>inet_network(3N)</td>
<td>See inet(3N)</td>
</tr>
<tr>
<td>inet_network(3XN)</td>
<td>See inet_addr(3XN)</td>
</tr>
<tr>
<td>inet_ntoa(3N)</td>
<td>See inet(3N)</td>
</tr>
<tr>
<td>inet_ntoa(3XN)</td>
<td>See inet_addr(3XN)</td>
</tr>
<tr>
<td>init_color(3X)</td>
<td>See curs_color(3X)</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>init_color(3XC)</td>
<td>See can_change_color(3XC)</td>
</tr>
<tr>
<td>initgroups(3C)</td>
<td>initialize the supplementary group access list</td>
</tr>
<tr>
<td>init_pair(3X)</td>
<td>See curs_color(3X)</td>
</tr>
<tr>
<td>init_pair(3XC)</td>
<td>See can_change_color(3XC)</td>
</tr>
<tr>
<td>initscr(3X)</td>
<td>See curs_initscr(3X)</td>
</tr>
<tr>
<td>initscr(3XC)</td>
<td>screen initialization functions</td>
</tr>
<tr>
<td>initstate(3C)</td>
<td>See random(3C)</td>
</tr>
<tr>
<td>innetgr(3N)</td>
<td>See getnetgrent(3N)</td>
</tr>
<tr>
<td>innstr(3X)</td>
<td>See curs_instr(3X)</td>
</tr>
<tr>
<td>innstr(3XC)</td>
<td>retrieve a multibyte character string (without rendition)</td>
</tr>
<tr>
<td>innwstr(3X)</td>
<td>See curs_inwstr(3X)</td>
</tr>
<tr>
<td>innwstr(3XC)</td>
<td>retrieve a wide character string (without rendition)</td>
</tr>
<tr>
<td>insch(3X)</td>
<td>See curs_insch(3X)</td>
</tr>
<tr>
<td>insch(3XC)</td>
<td>insert a character</td>
</tr>
<tr>
<td>insdelln(3X)</td>
<td>See curs_deleteln(3X)</td>
</tr>
<tr>
<td>insdelln(3XC)</td>
<td>insert/delete lines to/from the window</td>
</tr>
<tr>
<td>insertln(3X)</td>
<td>See curs_deleteln(3X)</td>
</tr>
<tr>
<td>insertln(3XC)</td>
<td>insert a line in a window</td>
</tr>
<tr>
<td>insnstr(3X)</td>
<td>See curs_instr(3X)</td>
</tr>
<tr>
<td>insnstr(3XC)</td>
<td>insert a multibyte character string</td>
</tr>
<tr>
<td>insnwstr(3X)</td>
<td>See curs_inwstr(3X)</td>
</tr>
<tr>
<td>ins_nwstr(3XC)</td>
<td>insert a wide character string</td>
</tr>
<tr>
<td>insque(3C)</td>
<td>insert/remove element from a queue</td>
</tr>
<tr>
<td>insstr(3X)</td>
<td>See curs_instr(3X)</td>
</tr>
<tr>
<td>insstr(3XC)</td>
<td>See insnstr(3XC)</td>
</tr>
<tr>
<td>instr(3X)</td>
<td>See curs_instr(3X)</td>
</tr>
<tr>
<td>instr(3XC)</td>
<td>Seeinsnstr(3XC)</td>
</tr>
<tr>
<td>inswch(3X)</td>
<td>See curs_inswch(3X)</td>
</tr>
<tr>
<td>ins_wch(3XC)</td>
<td>insert a complex character</td>
</tr>
<tr>
<td>inswstr(3X)</td>
<td>See curs_inwstr(3X)</td>
</tr>
<tr>
<td>ins_wstr(3XC)</td>
<td>See ins_nwstr(3XC)</td>
</tr>
<tr>
<td>intrflush(3X)</td>
<td>See curs_inopts(3X)</td>
</tr>
<tr>
<td>intrflush(3XC)</td>
<td>flush output in tty on interrupt</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>inwch(3X)</td>
<td>See curs_inwch(3X)</td>
</tr>
<tr>
<td>in_wch(3XC)</td>
<td>retrieve a complex character (with rendition)</td>
</tr>
<tr>
<td>inwchnstr(3X)</td>
<td>See curs_inwchnstr(3X)</td>
</tr>
<tr>
<td>in_wchnstr(3XC)</td>
<td>retrieve complex character string (with rendition)</td>
</tr>
<tr>
<td>inwchstr(3X)</td>
<td>See curs_inwchstr(3X)</td>
</tr>
<tr>
<td>in_wchstr(3XC)</td>
<td>See in_wchnstr(3XC)</td>
</tr>
<tr>
<td>inwstr(3X)</td>
<td>See curs_inwstr(3X)</td>
</tr>
<tr>
<td>inwstr(3XC)</td>
<td>See innwstr(3XC)</td>
</tr>
<tr>
<td>isalnum(3C)</td>
<td>See ctype(3C)</td>
</tr>
<tr>
<td>isalpha(3C)</td>
<td>See ctype(3C)</td>
</tr>
<tr>
<td>isascii(3C)</td>
<td>See ctype(3C)</td>
</tr>
<tr>
<td>isastream(3C)</td>
<td>test a file descriptor</td>
</tr>
<tr>
<td>isatty(3C)</td>
<td>test for a terminal device</td>
</tr>
<tr>
<td>iscntrl(3C)</td>
<td>See ctype(3C)</td>
</tr>
<tr>
<td>isdigit(3C)</td>
<td>See ctype(3C)</td>
</tr>
<tr>
<td>isencrypt(3G)</td>
<td>determine whether a buffer of characters is encrypted</td>
</tr>
<tr>
<td>isendwin(3X)</td>
<td>See curs_initscr(3X)</td>
</tr>
<tr>
<td>isendwin(3XC)</td>
<td>See endwin(3XC)</td>
</tr>
<tr>
<td>isenglish(3C)</td>
<td>See iswalpha(3C)</td>
</tr>
<tr>
<td>isgraph(3C)</td>
<td>See ctype(3C)</td>
</tr>
<tr>
<td>isideogram(3C)</td>
<td>See iswalpha(3C)</td>
</tr>
<tr>
<td>is_linetouched(3X)</td>
<td>control window refresh</td>
</tr>
<tr>
<td>is_linetouched(3XC)</td>
<td>See curs_touch(3X)</td>
</tr>
<tr>
<td>islower(3C)</td>
<td>See ctype(3C)</td>
</tr>
<tr>
<td>isnan(3C)</td>
<td>determine type of floating-point number</td>
</tr>
<tr>
<td>isnan(3M)</td>
<td>test for NaN</td>
</tr>
<tr>
<td>isnand(3C)</td>
<td>See isnan(3C)</td>
</tr>
<tr>
<td>isnanf(3C)</td>
<td>See isnanf(3C)</td>
</tr>
<tr>
<td>isnumber(3C)</td>
<td>See iswalpha(3C)</td>
</tr>
<tr>
<td>ispunct(3C)</td>
<td>See ctype(3C)</td>
</tr>
<tr>
<td>isspace(3C)</td>
<td>See ctype(3C)</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>isspecial(3C)</td>
<td>See iswalpha(3C)</td>
</tr>
<tr>
<td>isupper(3C)</td>
<td>See ctype(3C)</td>
</tr>
<tr>
<td>iswalnum(3C)</td>
<td>See iswalpha(3C)</td>
</tr>
<tr>
<td>iswalpha(3C)</td>
<td>wide-character code classification functions</td>
</tr>
<tr>
<td>iswascii(3C)</td>
<td>See iswalpha(3C)</td>
</tr>
<tr>
<td>iswcntrl(3C)</td>
<td>See iswalpha(3C)</td>
</tr>
<tr>
<td>iswctype(3C)</td>
<td>test character for specified class</td>
</tr>
<tr>
<td>iswdigit(3C)</td>
<td>See iswalpha(3C)</td>
</tr>
<tr>
<td>iswgraph(3C)</td>
<td>See iswalpha(3C)</td>
</tr>
<tr>
<td>is_wintouched(3X)</td>
<td>See curs_touch(3X)</td>
</tr>
<tr>
<td>is_wintouched(3XC)</td>
<td>See is_linetouched(3XC)</td>
</tr>
<tr>
<td>iswlower(3C)</td>
<td>See iswalpha(3C)</td>
</tr>
<tr>
<td>iswprint(3C)</td>
<td>See iswalpha(3C)</td>
</tr>
<tr>
<td>iswpunct(3C)</td>
<td>See iswalpha(3C)</td>
</tr>
<tr>
<td>iswspace(3C)</td>
<td>See iswalpha(3C)</td>
</tr>
<tr>
<td>iswupper(3C)</td>
<td>See iswalpha(3C)</td>
</tr>
<tr>
<td>iswxdigit(3C)</td>
<td>See iswalpha(3C)</td>
</tr>
<tr>
<td>isxdigit(3C)</td>
<td>See ctype(3C)</td>
</tr>
<tr>
<td>item_count(3X)</td>
<td>See menu_items(3X)</td>
</tr>
<tr>
<td>item_description(3X)</td>
<td>See menu_item_name(3X)</td>
</tr>
<tr>
<td>item_index(3X)</td>
<td>See menu_item_current(3X)</td>
</tr>
<tr>
<td>item_init(3X)</td>
<td>See menu_hook(3X)</td>
</tr>
<tr>
<td>item_name(3X)</td>
<td>See menu_item_name(3X)</td>
</tr>
<tr>
<td>item_opts(3X)</td>
<td>See menu_item_opts(3X)</td>
</tr>
<tr>
<td>item_opts_off(3X)</td>
<td>See menu_item_opts(3X)</td>
</tr>
<tr>
<td>item_opts_on(3X)</td>
<td>See menu_item_opts(3X)</td>
</tr>
<tr>
<td>item_term(3X)</td>
<td>See menu_hook(3X)</td>
</tr>
<tr>
<td>item_userptr(3X)</td>
<td>See menu_item_userptr(3X)</td>
</tr>
<tr>
<td>item_value(3X)</td>
<td>See menu_item_value(3X)</td>
</tr>
<tr>
<td>item_visible(3X)</td>
<td>See menu_item_visible(3X)</td>
</tr>
<tr>
<td>j0(3M)</td>
<td>Bessel functions of the first kind</td>
</tr>
<tr>
<td>j1(3M)</td>
<td>See j0(3M)</td>
</tr>
<tr>
<td>jn(3M)</td>
<td>See j0(3M)</td>
</tr>
<tr>
<td>jrand48(3C)</td>
<td>See drand48(3C)</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>kerberos(3N)</td>
<td>Kerberos authentication library</td>
</tr>
<tr>
<td>kerberos_rpc(3N)</td>
<td>library routines for remote procedure calls using Kerberos authentication</td>
</tr>
<tr>
<td>key_decryptsession(3N)</td>
<td>See secure_rpc(3N)</td>
</tr>
<tr>
<td>key_encryptsession(3N)</td>
<td>See secure_rpc(3N)</td>
</tr>
<tr>
<td>key_gendes(3N)</td>
<td>See secure_rpc(3N)</td>
</tr>
<tr>
<td>key_decryptsession(3N)</td>
<td>return character string used as key name</td>
</tr>
<tr>
<td>keyname(3X)</td>
<td>See secure_rpc(3N)</td>
</tr>
<tr>
<td>keyname(3XC)</td>
<td>See secure_rpc(3N)</td>
</tr>
<tr>
<td>key_name(3XC)</td>
<td>enable/disable keypad handling</td>
</tr>
<tr>
<td>keypad(3X)</td>
<td>See secure_rpc(3N)</td>
</tr>
<tr>
<td>keypad(3XC)</td>
<td>See secure_rpc(3N)</td>
</tr>
<tr>
<td>key_secretkey_is_set(3N)</td>
<td>See secure_rpc(3N)</td>
</tr>
<tr>
<td>key_setsecret(3N)</td>
<td>See secure_rpc(3N)</td>
</tr>
<tr>
<td>killchar(3X)</td>
<td>send signal to a process group</td>
</tr>
<tr>
<td>killchar(3XC)</td>
<td>See secure_rpc(3N)</td>
</tr>
<tr>
<td>.killpg(3C)</td>
<td>enable/disable keypad handling</td>
</tr>
<tr>
<td>killwchar(3XC)</td>
<td>See secure_rpc(3N)</td>
</tr>
<tr>
<td>krb_get_admhst(3N)</td>
<td>See krb_realmofhost(3N)</td>
</tr>
<tr>
<td>krb_get_cred(3N)</td>
<td>See kerberos(3N)</td>
</tr>
<tr>
<td>krb_get_krbhst(3N)</td>
<td>See krb_realmofhost(3N)</td>
</tr>
<tr>
<td>krb_get_lrealm(3N)</td>
<td>See krb_realmofhost(3N)</td>
</tr>
<tr>
<td>krb_get_phost(3N)</td>
<td>See krb_realmofhost(3N)</td>
</tr>
<tr>
<td>krb_kntoln(3N)</td>
<td>See kerberos(3N)</td>
</tr>
<tr>
<td>krb_mk_err(3N)</td>
<td>See kerberos(3N)</td>
</tr>
<tr>
<td>krb_mk_req(3N)</td>
<td>See kerberos(3N)</td>
</tr>
<tr>
<td>krb_mk_safe(3N)</td>
<td>See kerberos(3N)</td>
</tr>
<tr>
<td>krb_net_read(3N)</td>
<td>See krb_sendauth(3N)</td>
</tr>
<tr>
<td>krb_net_write(3N)</td>
<td>See krb_sendauth(3N)</td>
</tr>
<tr>
<td>krb_rd_err(3N)</td>
<td>See kerberos(3N)</td>
</tr>
<tr>
<td>krb_rd_req(3N)</td>
<td>See kerberos(3N)</td>
</tr>
<tr>
<td>krb_rd_safe(3N)</td>
<td>See kerberos(3N)</td>
</tr>
<tr>
<td>krb_realmofhost(3N)</td>
<td>additional Kerberos utility routines</td>
</tr>
<tr>
<td>krb_recvaith(3N)</td>
<td>See krb_sendauth(3N)</td>
</tr>
<tr>
<td>krb_sendauth(3N)</td>
<td>Kerberos routines for sending authentication via network stream sockets</td>
</tr>
<tr>
<td>krb_set_key(3N)</td>
<td>See kerberos(3N)</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td>krb_set_tkt_string(3N)</td>
<td>set Kerberos ticket cache file name</td>
</tr>
<tr>
<td>kstat(3K)</td>
<td>kernel statistics facility</td>
</tr>
<tr>
<td>kstat_chain_update(3K)</td>
<td>update the kstat header chain</td>
</tr>
<tr>
<td>kstat_close(3K)</td>
<td>See kstat_open(3K)</td>
</tr>
<tr>
<td>kstat_data_lookup(3K)</td>
<td>See kstat_lookup(3K)</td>
</tr>
<tr>
<td>kstat_lookup(3K)</td>
<td>find a kstat by name</td>
</tr>
<tr>
<td>kstat_open(3K)</td>
<td>initialize kernel statistics facility</td>
</tr>
<tr>
<td>kstat_read(3K)</td>
<td>read or write kstat data</td>
</tr>
<tr>
<td>kstat_write(3K)</td>
<td>See kstat_read(3K)</td>
</tr>
<tr>
<td>kvm_close(3K)</td>
<td>See kvm_open(3K)</td>
</tr>
<tr>
<td>kvm_getcmd(3K)</td>
<td>See kvm_getu(3K)</td>
</tr>
<tr>
<td>kvm_getproc(3K)</td>
<td>get the u-area or invocation arguments for a process</td>
</tr>
<tr>
<td>kvm_getu(3K)</td>
<td>See kvm_read(3K)</td>
</tr>
<tr>
<td>kvm_kread(3K)</td>
<td>See kvm_read(3K)</td>
</tr>
<tr>
<td>kvm_kwrite(3K)</td>
<td>read system process structures</td>
</tr>
<tr>
<td>kvm_nextproc(3K)</td>
<td>get entries from kernel symbol table</td>
</tr>
<tr>
<td>kvm_nlist(3K)</td>
<td>specify a kernel to examine</td>
</tr>
<tr>
<td>kvm_open(3K)</td>
<td>copy data to or from a kernel image or running system</td>
</tr>
<tr>
<td>kvm_read(3K)</td>
<td>See kvm_nextproc(3K)</td>
</tr>
<tr>
<td>kvm_setproc(3K)</td>
<td>See kvm_read(3K)</td>
</tr>
<tr>
<td>kvm_uread(3K)</td>
<td>See kvm_read(3K)</td>
</tr>
<tr>
<td>kvm_uwrite(3K)</td>
<td>See kvm_read(3K)</td>
</tr>
<tr>
<td>kvm_write(3K)</td>
<td>See kvm_read(3K)</td>
</tr>
<tr>
<td>l64a(3C)</td>
<td>See a64l(3C)</td>
</tr>
<tr>
<td>label(3)</td>
<td>See plot(3)</td>
</tr>
<tr>
<td>labs(3C)</td>
<td>See abs(3C)</td>
</tr>
<tr>
<td>lckpwdf(3C)</td>
<td>manipulate shadow password database lock file</td>
</tr>
<tr>
<td>lcong48(3C)</td>
<td>See drand48(3C)</td>
</tr>
<tr>
<td>ldexp(3C)</td>
<td>load exponent of a floating point number</td>
</tr>
<tr>
<td>ldiv(3C)</td>
<td>See div(3C)</td>
</tr>
<tr>
<td>leaveok(3X)</td>
<td>See curs_outopts(3X)</td>
</tr>
<tr>
<td>leaveok(3XC)</td>
<td>See clearok(3XC)</td>
</tr>
<tr>
<td>lfind(3C)</td>
<td>See lsearch(3C)</td>
</tr>
</tbody>
</table>
lfmt(3C) display error message in standard format and pass to logging and monitoring services
lgamma(3M) log gamma function
lgamma_r(3M) See lgamma(3M)
libpthread(3T) See threads(3T)
libthread(3T) See threads(3T)
libthread_db(3T) library of interfaces for monitoring and manipulating threads-related aspects of multithreaded programs
libtnfctl(3X) library for TNF probe control in a process or the kernel
line(3) See plot(3)
link_field(3X) See form_field_new(3X)
link_fieldtype(3X) See form_fieldtype(3X)
linmod(3) See plot(3)
lio_listio(3R) list directed I/O
listen(3N) listen for connections on a socket
listen(3XN) listen for socket connections and limit the queue of incoming connections
llabs(3C) See abs(3C)
lldiv(3C) See div(3C)
lltostr(3C) See strtol(3C)
localeconv(3C) get numeric formatting information
localtime(3C) See ctime(3C)
localtime_r(3C) See ctime(3C)
lockf(3C) record locking on files
log(3M) natural logarithm function
log10(3M) base 10 logarithm function
log1p(3M) compute natural logarithm
logb(3M) radix-independent exponent
_longjmp(3B) See setjmp(3B)
longjmp(3B) See setjmp(3B)
_longjmp(3C) non-local goto
longjmp(3C) See setjmp(3C)
longname(3X) See curs_termattr(3X)
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>longname(3XC)</td>
<td>return full terminal type name</td>
</tr>
<tr>
<td>lrand48(3C)</td>
<td>See drand48(3C)</td>
</tr>
<tr>
<td>lsearch(3C)</td>
<td>linear search and update</td>
</tr>
<tr>
<td>madvise(3)</td>
<td>provide advice to VM system</td>
</tr>
<tr>
<td>maillock(3X)</td>
<td>functions to manage lockfile(s) for user’s mailbox</td>
</tr>
<tr>
<td>mailunlock(3X)</td>
<td>See maillock(3X)</td>
</tr>
<tr>
<td>major(3C)</td>
<td>See makedev(3C)</td>
</tr>
<tr>
<td>makecontext(3C)</td>
<td>manipulate user contexts</td>
</tr>
<tr>
<td>makedev(3C)</td>
<td>manage a device number</td>
</tr>
<tr>
<td>mallinfo(3X)</td>
<td>See malloc(3X)</td>
</tr>
<tr>
<td>malloc(3C)</td>
<td>memory allocator</td>
</tr>
<tr>
<td>malloc(3X)</td>
<td>memory allocator</td>
</tr>
<tr>
<td>malloc(3X)</td>
<td>See bsdmalloc(3X)</td>
</tr>
<tr>
<td>malloc(3X)</td>
<td>See watchmalloc(3X)</td>
</tr>
<tr>
<td>mallalloc(3X)</td>
<td>See malloc(3X)</td>
</tr>
<tr>
<td>malloc(3X)</td>
<td>memory allocator</td>
</tr>
<tr>
<td>malloc(3X)</td>
<td>math library exception-handling function</td>
</tr>
<tr>
<td>mbslen(3C)</td>
<td>get number of bytes in a character</td>
</tr>
<tr>
<td>mbstowcs(3C)</td>
<td>convert a character string to a wide-character string</td>
</tr>
<tr>
<td>mbtowc(3C)</td>
<td>convert a character to a wide-character code</td>
</tr>
<tr>
<td>mctl(3B)</td>
<td>memory management control</td>
</tr>
<tr>
<td>media_findname(3X)</td>
<td>convert a supplied name into an absolute pathname that can be used to access removable media</td>
</tr>
<tr>
<td>media_getattr(3X)</td>
<td>get and set media attributes</td>
</tr>
<tr>
<td>media_setattr(3X)</td>
<td>See media_getattr(3X)</td>
</tr>
<tr>
<td>memalign(3C)</td>
<td>See malloc(3C)</td>
</tr>
<tr>
<td>memalign(3X)</td>
<td>See watchmalloc(3X)</td>
</tr>
<tr>
<td>memccpy(3C)</td>
<td>See memory(3C)</td>
</tr>
<tr>
<td>memchr(3C)</td>
<td>See memory(3C)</td>
</tr>
<tr>
<td>memcmp(3C)</td>
<td>See memory(3C)</td>
</tr>
<tr>
<td>memcpy(3C)</td>
<td>See memory(3C)</td>
</tr>
<tr>
<td>memmove(3C)</td>
<td>See memory(3C)</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>memory(3C)</code></td>
<td>memory operations</td>
</tr>
<tr>
<td><code>memset(3C)</code></td>
<td>See <code>memory(3C)</code></td>
</tr>
<tr>
<td><code>menu_attributes(3X)</code></td>
<td>control menus display attributes</td>
</tr>
<tr>
<td><code>menu_back(3X)</code></td>
<td>See <code>menu_attributes(3X)</code></td>
</tr>
<tr>
<td><code>menu_cursor(3X)</code></td>
<td>correctly position a menus cursor</td>
</tr>
<tr>
<td><code>menu_driver(3X)</code></td>
<td>command processor for the menus subsystem</td>
</tr>
<tr>
<td><code>menu_fore(3X)</code></td>
<td>See <code>menu_attributes(3X)</code></td>
</tr>
<tr>
<td><code>menu_format(3X)</code></td>
<td>set and get maximum numbers of rows and columns in menus</td>
</tr>
<tr>
<td><code>menu_grey(3X)</code></td>
<td>See <code>menu_attributes(3X)</code></td>
</tr>
<tr>
<td><code>menu_hook(3X)</code></td>
<td>assign application-specific routines for automatic invocation by menus</td>
</tr>
<tr>
<td><code>menu_init(3X)</code></td>
<td>See <code>menu_hook(3X)</code></td>
</tr>
<tr>
<td><code>menu_item_current(3X)</code></td>
<td>set and get current menus items</td>
</tr>
<tr>
<td><code>menu_item_name(3X)</code></td>
<td>get menus item name and description</td>
</tr>
<tr>
<td><code>menu_item_new(3X)</code></td>
<td>create and destroy menus items</td>
</tr>
<tr>
<td><code>menu_item_opts(3X)</code></td>
<td>menus item option routines</td>
</tr>
<tr>
<td><code>menu_items(3X)</code></td>
<td>connect and disconnect items to and from menus</td>
</tr>
<tr>
<td><code>menu_item_userptr(3X)</code></td>
<td>associate application data with menus items</td>
</tr>
<tr>
<td><code>menu_item_value(3X)</code></td>
<td>set and get menus item values</td>
</tr>
<tr>
<td><code>menu_item_visible(3X)</code></td>
<td>tell if menus item is visible</td>
</tr>
<tr>
<td><code>menu_mark(3X)</code></td>
<td>menus mark string routines</td>
</tr>
<tr>
<td><code>menu_new(3X)</code></td>
<td>create and destroy menus</td>
</tr>
<tr>
<td><code>menu_opts(3X)</code></td>
<td>menus option routines</td>
</tr>
<tr>
<td><code>menu_opts_off(3X)</code></td>
<td>See <code>menu_opts(3X)</code></td>
</tr>
<tr>
<td><code>menu_opts_on(3X)</code></td>
<td>See <code>menu_opts(3X)</code></td>
</tr>
<tr>
<td><code>menu_pad(3X)</code></td>
<td>See <code>menu_attributes(3X)</code></td>
</tr>
<tr>
<td><code>menu_pattern(3X)</code></td>
<td>set and get menus pattern match buffer</td>
</tr>
<tr>
<td><code>menu_post(3X)</code></td>
<td>write or erase menus from associated subwindows</td>
</tr>
<tr>
<td><code>menus(3X)</code></td>
<td>character based menus package</td>
</tr>
<tr>
<td><code>menu_sub(3X)</code></td>
<td>See <code>menu_win(3X)</code></td>
</tr>
<tr>
<td><code>menu_term(3X)</code></td>
<td>See <code>menu_hook(3X)</code></td>
</tr>
</tbody>
</table>
menu_userptr(3X)  
  associate application data with menus

menu_win(3X)  
  menus window and subwindow association routines

meta(3X)  
  See curs_inopts(3X)

meta(3XC)  
  enable/disable meta keys

minor(3C)  
  See makedev(3C)

mkdirp(3G)  
  create, remove directories in a path

mkfifo(3C)  
  create a new FIFO

mkstemp(3C)  
  make a unique file name

mktemp(3C)  
  make a unique file name

mktime(3C)  
  converts a tm structure to a calendar time

mlock(3C)  
  lock (or unlock) pages in memory

mlockall(3C)  
  lock or unlock address space

modf(3C)  
  decompose floating-point number

modff(3C)  
  See modf(3C)

monitor(3C)  
  prepare process execution profile

  See plot(3)

move(3)  
  See curs_move(3X)

move(3X)  
  move cursor in window

move(3XC)  
  See form_field(3X)

move_field(3X)  
  See curs_alecompat(3X)

move_nextch(3X)  
  See curs_alecompat(3X)

move_panel(3X)  
  See panel_move(3X)

move_prevch(3X)  
  See curs_alecompat(3X)

mp(3M)  
  multiple precision integer arithmetic

mp_gcd(3M)  
  See mp(3M)

mp_itom(3M)  
  See mp(3M)

mp_madd(3M)  
  See mp(3M)

mp_mcmp(3M)  
  See mp(3M)

mp_mdiv(3M)  
  See mp(3M)

mp_mfree(3M)  
  See mp(3M)

mp_min(3M)  
  See mp(3M)

mp_mout(3M)  
  See mp(3M)

mp_msub(3M)  
  See mp(3M)

mp_mtox(3M)  
  See mp(3M)

mp_mult(3M)  
  See mp(3M)
mp_pow(3M) See mp(3M)
mp_rpow(3M) See mp(3M)
mp_xtom(3M) See mp(3M)
mq_close(3R) close a message queue
mq_getattr(3R) See mq_setattr(3R)
mq_notify(3R) notify process (or thread) that a message is available on a queue
mq_open(3R) open a message queue
mq_receive(3R) receive a message from a message queue
mq_send(3R) send a message to a message queue
mq_setattr(3R) set/get message queue attributes
mq_unlink(3R) remove a message queue
mrnd48(3C) See drand48(3C)
msync(3C) synchronize memory with physical storage
munlock(3C) See mlock(3C)
munlockall(3C) See mlockall(3C)
mutex(3T) mutual exclusion locks
mutex_destroy(3T) See mutex(3T)
mutex_init(3T) See mutex(3T)
mutex_lock(3T) See mutex(3T)
mutex_trylock(3T) See mutex(3T)
mutex_unlock(3T) See mutex(3T)
mvaddch(3X) See curs_addch(3X)
mvaddch(3XC) See addch(3XC)
mvaddchnstr(3X) See curs_addchstr(3X)
mvaddchnstr(3XC) See addchstr(3XC)
mvaddchstr(3X) See curs_addchstr(3X)
mvaddchstr(3XC) See addchstr(3XC)
mvaddnstr(3X) See curs_addstr(3X)
mvaddnstr(3XC) See addnstr(3XC)
mvaddnwstr(3X) See curs_addwstr(3X)
mvaddnwstr(3XC) See addnwstr(3XC)
mvaddstr(3X) See curs_addstr(3X)
mvaddstr(3XC) See addnstr(3XC)
mvaddwch(3X) See curs_addwch(3X)
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mvadd_wch(3XC)</td>
<td>See add_wch(3XC)</td>
</tr>
<tr>
<td>mvaddwchnstr(3X)</td>
<td>See curs_addwchstr(3X)</td>
</tr>
<tr>
<td>mvadd_wchnstr(3XC)</td>
<td>See add_wchnstr(3XC)</td>
</tr>
<tr>
<td>mvaddwchstr(3X)</td>
<td>See curs_addwchstr(3X)</td>
</tr>
<tr>
<td>mvadd_wchstr(3XC)</td>
<td>See add_wchstr(3XC)</td>
</tr>
<tr>
<td>mvaddwstr(3X)</td>
<td>See curs_addwstr(3X)</td>
</tr>
<tr>
<td>mvaddwstr(3XC)</td>
<td>See addwstr(3XC)</td>
</tr>
<tr>
<td>mvchgat(3XC)</td>
<td>See chgat(3XC)</td>
</tr>
<tr>
<td>mvcur(3X)</td>
<td>See curs_terminfo(3X)</td>
</tr>
<tr>
<td>mvcur(3XC)</td>
<td>move the cursor</td>
</tr>
<tr>
<td>mvdelch(3X)</td>
<td>See curs_delch(3X)</td>
</tr>
<tr>
<td>mvdelch(3XC)</td>
<td>See delch(3XC)</td>
</tr>
<tr>
<td>mvderwin(3X)</td>
<td>See curs_window(3X)</td>
</tr>
<tr>
<td>mvderwin(3XC)</td>
<td>map area of parent window to subwindow</td>
</tr>
<tr>
<td>mvgetch(3X)</td>
<td>See curs_getch(3X)</td>
</tr>
<tr>
<td>mvgetch(3XC)</td>
<td>See getch(3XC)</td>
</tr>
<tr>
<td>mvgetnstr(3XC)</td>
<td>See getnstr(3XC)</td>
</tr>
<tr>
<td>mvgetnstr(3XC)</td>
<td>See curs_getnstr(3XC)</td>
</tr>
<tr>
<td>mvgetn_wstr(3XC)</td>
<td>See getn_wstr(3XC)</td>
</tr>
<tr>
<td>mvgetnstr(3X)</td>
<td>See curs_getnstr(3XC)</td>
</tr>
<tr>
<td>mvgetnstr(3XC)</td>
<td>See getnstr(3XC)</td>
</tr>
<tr>
<td>mvgetstr(3XC)</td>
<td>See curs_getstr(3XC)</td>
</tr>
<tr>
<td>mvgetstr(3X)</td>
<td>See getnstr(3XC)</td>
</tr>
<tr>
<td>mvgetwch(3XC)</td>
<td>See curs_getwch(3X)</td>
</tr>
<tr>
<td>mvgetwch(3X)</td>
<td>See get_wch(3XC)</td>
</tr>
<tr>
<td>mvgetwstr(3X)</td>
<td>See curs_getwstr(3X)</td>
</tr>
<tr>
<td>mvgetwstr(3XC)</td>
<td>See getn_wstr(3XC)</td>
</tr>
<tr>
<td>mvhline(3XC)</td>
<td>See hline(3XC)</td>
</tr>
<tr>
<td>mvhline_set(3XC)</td>
<td>See hline_set(3XC)</td>
</tr>
<tr>
<td>mvinch(3X)</td>
<td>See curs_inch(3X)</td>
</tr>
<tr>
<td>mvinch(3XC)</td>
<td>See inch(3XC)</td>
</tr>
<tr>
<td>mvinchnstr(3X)</td>
<td>See curs_inchnstr(3X)</td>
</tr>
<tr>
<td>mvinchnstr(3XC)</td>
<td>See inchnstr(3XC)</td>
</tr>
<tr>
<td>mvinchstr(3X)</td>
<td>See curs_inchstr(3X)</td>
</tr>
<tr>
<td>mvinchstr(3XC)</td>
<td>See inchnstr(3XC)</td>
</tr>
<tr>
<td>mvinnstr(3X)</td>
<td>See curs_instr(3X)</td>
</tr>
<tr>
<td>mvinnstr(3XC)</td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>------------------------------------------------</td>
</tr>
<tr>
<td>mvinnstr(3XC)</td>
<td>See innstr(3XC)</td>
</tr>
<tr>
<td>mvinnwstr(3X)</td>
<td>See curs_inwstr(3X)</td>
</tr>
<tr>
<td>mvinnwstr(3XC)</td>
<td>See innwstr(3XC)</td>
</tr>
<tr>
<td>mvisnch(3X)</td>
<td>See curs_insch(3X)</td>
</tr>
<tr>
<td>mvisnch(3XC)</td>
<td>See insch(3XC)</td>
</tr>
<tr>
<td>mvisnstr(3X)</td>
<td>See curs_insstr(3X)</td>
</tr>
<tr>
<td>mvisnstr(3XC)</td>
<td>See insnstr(3XC)</td>
</tr>
<tr>
<td>mvisnwstr(3X)</td>
<td>See curs_inswwstr(3X)</td>
</tr>
<tr>
<td>mvis_nwstr(3XC)</td>
<td>See ins_nwstr(3XC)</td>
</tr>
<tr>
<td>mvisnsstr(3X)</td>
<td>See curs_insstr(3X)</td>
</tr>
<tr>
<td>mvisnsstr(3XC)</td>
<td>See insnstr(3XC)</td>
</tr>
<tr>
<td>mvisnwstr(3X)</td>
<td>See curs_inswwstr(3X)</td>
</tr>
<tr>
<td>mvisnstr(3XC)</td>
<td>See curs_instr(3X)</td>
</tr>
<tr>
<td>mvinstr(3XC)</td>
<td>See innstr(3XC)</td>
</tr>
<tr>
<td>mvinstr(3XC)</td>
<td>See curs_instr(3X)</td>
</tr>
<tr>
<td>mvisnwstr(3X)</td>
<td>See curs_inswwstr(3X)</td>
</tr>
<tr>
<td>mvisnsstr(3XC)</td>
<td>See curs_insstr(3X)</td>
</tr>
<tr>
<td>mvisnsstr(3XC)</td>
<td>See inwwstr(3XC)</td>
</tr>
<tr>
<td>mvprintw(3X)</td>
<td>See curs_printw(3X)</td>
</tr>
<tr>
<td>mvprintw(3XC)</td>
<td>write formatted output to window</td>
</tr>
<tr>
<td>mvscanw(3X)</td>
<td>See curs_scanw(3X)</td>
</tr>
<tr>
<td>mvscanw(3XC)</td>
<td>read formatted input from window</td>
</tr>
<tr>
<td>mvvline(3XC)</td>
<td>See hline(3XC)</td>
</tr>
<tr>
<td>mvvline_set(3XC)</td>
<td>See hline_set(3XC)</td>
</tr>
<tr>
<td>mvwaddch(3X)</td>
<td>See curs_addch(3X)</td>
</tr>
<tr>
<td>mvwaddch(3XC)</td>
<td>See addch(3X)</td>
</tr>
<tr>
<td>mvwaddchnstr(3X)</td>
<td>See curs_addchstr(3X)</td>
</tr>
<tr>
<td>mvwaddchnstr(3XC)</td>
<td>See addchstr(3X)</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td>mvwaddchstr(3X)</td>
<td>See curs_addchstr(3X)</td>
</tr>
<tr>
<td>mvwaddchstr(3XC)</td>
<td>See addchstr(3XC)</td>
</tr>
<tr>
<td>mvwaddnstr(3X)</td>
<td>See curs_addstr(3X)</td>
</tr>
<tr>
<td>mvwaddnstr(3XC)</td>
<td>See addnstr(3XC)</td>
</tr>
<tr>
<td>mvwaddnwstr(3X)</td>
<td>See addnwstr(3X)</td>
</tr>
<tr>
<td>mvwaddnwstr(3XC)</td>
<td>See addnwstr(3XC)</td>
</tr>
<tr>
<td>mvwaddstr(3X)</td>
<td>See curs_addstr(3X)</td>
</tr>
<tr>
<td>mvwaddstr(3XC)</td>
<td>See addnstr(3XC)</td>
</tr>
<tr>
<td>mvwadd_wch(3XC)</td>
<td>See add_wch(3XC)</td>
</tr>
<tr>
<td>mvwaddwchnstr(3X)</td>
<td>See curs_addwchstr(3X)</td>
</tr>
<tr>
<td>mvwaddwchnstr(3XC)</td>
<td>See add_wchnstr(3XC)</td>
</tr>
<tr>
<td>mvwaddwchstr(3X)</td>
<td>See curs_addwchstr(3X)</td>
</tr>
<tr>
<td>mvwaddwchstr(3XC)</td>
<td>See add_wchnstr(3XC)</td>
</tr>
<tr>
<td>mvwaddwstr(3X)</td>
<td>See curs_addwstr(3X)</td>
</tr>
<tr>
<td>mvwaddwstr(3XC)</td>
<td>See addnstr(3XC)</td>
</tr>
<tr>
<td>mvwcchgat(3XC)</td>
<td>See chgat(3XC)</td>
</tr>
<tr>
<td>mvwdelch(3X)</td>
<td>See curs_delch(3XC)</td>
</tr>
<tr>
<td>mvwdelch(3XC)</td>
<td>See delch(3XC)</td>
</tr>
<tr>
<td>mvwgetch(3X)</td>
<td>See curs_getch(3XC)</td>
</tr>
<tr>
<td>mvwgetch(3XC)</td>
<td>See getch(3XC)</td>
</tr>
<tr>
<td>mvwgetnstr(3XC)</td>
<td>See getnstr(3XC)</td>
</tr>
<tr>
<td>mvwgetnwstr(3X)</td>
<td>See curs_getwstr(3X)</td>
</tr>
<tr>
<td>mvwgetnwstr(3XC)</td>
<td>See getn_wstr(3XC)</td>
</tr>
<tr>
<td>mvwgetn_str(3X)</td>
<td>See curs_getstr(3X)</td>
</tr>
<tr>
<td>mvwgetstr(3X)</td>
<td>See getnstr(3XC)</td>
</tr>
<tr>
<td>mvwgetwch(3X)</td>
<td>See curs_getwch(3X)</td>
</tr>
<tr>
<td>mvwgetwch(3XC)</td>
<td>See get_wch(3XC)</td>
</tr>
<tr>
<td>mvwgetwstr(3X)</td>
<td>See curs_getwstr(3X)</td>
</tr>
<tr>
<td>mvwgetwstr(3XC)</td>
<td>See getn_wstr(3XC)</td>
</tr>
<tr>
<td>mvwhline(3XC)</td>
<td>See hline(3XC)</td>
</tr>
<tr>
<td>mvwhline_set(3XC)</td>
<td>See hline_set(3XC)</td>
</tr>
<tr>
<td>mvwin(3X)</td>
<td>See curs_window(3X)</td>
</tr>
<tr>
<td>mvwin(3XC)</td>
<td>move window</td>
</tr>
<tr>
<td>mvwinch(3X)</td>
<td>See curs_inch(3X)</td>
</tr>
</tbody>
</table>

modified 4 Apr 1997 SunOS 5.6 3-57
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mvwinch(3XC)</td>
<td>See inch(3XC)</td>
</tr>
<tr>
<td>mvwinchnstr(3X)</td>
<td>See curs_inchstr(3X)</td>
</tr>
<tr>
<td>mvwinchnstr(3XC)</td>
<td>See inchnstr(3XC)</td>
</tr>
<tr>
<td>mvwinchstr(3X)</td>
<td>See curs_inchstr(3X)</td>
</tr>
<tr>
<td>mvwinchstr(3XC)</td>
<td>See inchnstr(3XC)</td>
</tr>
<tr>
<td>mvwinnstr(3X)</td>
<td>See curs_instr(3X)</td>
</tr>
<tr>
<td>mvwinnstr(3XC)</td>
<td>See innstr(3XC)</td>
</tr>
<tr>
<td>mvwinnwstr(3X)</td>
<td>See curs_inwstr(3X)</td>
</tr>
<tr>
<td>mvwinnwstr(3XC)</td>
<td>See inwstr(3XC)</td>
</tr>
<tr>
<td>mvwinsch(3X)</td>
<td>See curs_insch(3X)</td>
</tr>
<tr>
<td>mvwinsch(3XC)</td>
<td>See insch(3XC)</td>
</tr>
<tr>
<td>mvwinsnstr(3X)</td>
<td>See curs_insstr(3X)</td>
</tr>
<tr>
<td>mvwinsnstr(3XC)</td>
<td>See insnstr(3XC)</td>
</tr>
<tr>
<td>mvwins_nstr(3XC)</td>
<td>See ins_nwstr(3XC)</td>
</tr>
<tr>
<td>mvwinsnwstr(3X)</td>
<td>See curs_inswwstr(3X)</td>
</tr>
<tr>
<td>mvwins_nwstr(3XC)</td>
<td>See ins_nwstr(3XC)</td>
</tr>
<tr>
<td>mvwinsstr(3X)</td>
<td>See curs_instr(3X)</td>
</tr>
<tr>
<td>mvwinsstr(3XC)</td>
<td>See insnstr(3XC)</td>
</tr>
<tr>
<td>mvwinstr(3X)</td>
<td>See curs_instr(3X)</td>
</tr>
<tr>
<td>mvwinstr(3XC)</td>
<td>See innstr(3XC)</td>
</tr>
<tr>
<td>mvwinswch(3X)</td>
<td>See curs_inswch(3X)</td>
</tr>
<tr>
<td>mvwins_wch(3XC)</td>
<td>See ins_wch(3XC)</td>
</tr>
<tr>
<td>mvwinswstr(3X)</td>
<td>See curs_inswwstr(3X)</td>
</tr>
<tr>
<td>mvwinwch(3X)</td>
<td>See curs_inwch(3X)</td>
</tr>
<tr>
<td>mvwin_wch(3XC)</td>
<td>See in_wch(3XC)</td>
</tr>
<tr>
<td>mvwinwchnstr(3X)</td>
<td>See curs_inwchstr(3X)</td>
</tr>
<tr>
<td>mvwin_wchnstr(3XC)</td>
<td>See in_wchnstr(3XC)</td>
</tr>
<tr>
<td>mvwinwchstr(3X)</td>
<td>See curs_inwchstr(3X)</td>
</tr>
<tr>
<td>mvwin_wchstr(3XC)</td>
<td>See in_wchnstr(3XC)</td>
</tr>
<tr>
<td>mvwinwstr(3X)</td>
<td>See curs_inwstr(3X)</td>
</tr>
<tr>
<td>mvwinwstr(3XC)</td>
<td>See inwstr(3XC)</td>
</tr>
<tr>
<td>mvwprintw(3X)</td>
<td>See curs_printw(3X)</td>
</tr>
<tr>
<td>mvwprintw(3XC)</td>
<td>See mvprintw(3XC)</td>
</tr>
<tr>
<td>mvwscanw(3X)</td>
<td>See curs_scanw(3X)</td>
</tr>
<tr>
<td>mvwscanw(3XC)</td>
<td>See mvscanw(3XC)</td>
</tr>
</tbody>
</table>
**C Library Functions**

- `mvwvline(3XC)`
  See `hline(3XC)`
- `mvwvline_set(3XC)`
  See `hline_set(3XC)`
- `nanosleep(3R)`
  high resolution sleep
- `napms(3X)`
  See `curs_kernel(3X)`
- `napms(3XC)`
  sleep process for a specified length of time
- `nc_perror(3N)`
  See `getnetconfig(3N)`
- `nc_sperror(3N)`
  See `getnetconfig(3N)`
- `netdir(3N)`
  generic transport name-to-address translation
- `netdir_free(3N)`
  See `netdir(3N)`
- `netdir_getbyaddr(3N)`
  See `netdir(3N)`
- `netdir_getbyname(3N)`
  See `netdir(3N)`
- `netdir_mergeaddr(3N)`
  See `netdir(3N)`
- `netdir_options(3N)`
  See `netdir(3N)`
- `netdir_perror(3N)`
  See `netdir(3N)`
- `netdir_sperror(3N)`
  See `netdir(3N)`
- `netname2host(3N)`
  See `secure_rpc(3N)`
- `netname2user(3N)`
  See `secure_rpc(3N)`
- `new_field(3X)`
  See `form_field_new(3X)`
- `new_fieldtype(3X)`
  See `form_fieldtype(3X)`
- `new_form(3X)`
  See `form_new(3X)`
- `new_item(3X)`
  See `menu_item_new(3X)`
- `new_menu(3X)`
  See `menu_new(3X)`
- `newpad(3X)`
  See `curs_pad(3X)`
- `newpad(3XC)`
  create or refresh a pad or subpad
- `new_page(3X)`
  See `form_new_page(3X)`
- `new_panel(3X)`
  See `panel_new(3X)`
- `newterm(3X)`
  See `curs_initscr(3X)`
- `newterm(3XC)`
  See `initscr(3XC)`
- `newwin(3X)`
  See `curs_window(3X)`
- `newwin(3XC)`
  See `derwin(3XC)`
- `nextafter(3M)`
  next representable double-precision floating-point number
- `nextkey(3B)`
  See `dbm(3B)`
- `nftw(3C)`
  See `ftw(3C)`

modified 4 Apr 1997

SunOS 5.6 3-59
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nice(3B)</td>
<td>change priority of a process</td>
</tr>
<tr>
<td>nis_add(3N)</td>
<td>See nis_names(3N)</td>
</tr>
<tr>
<td>nis_add_entry(3N)</td>
<td>See nis_tables(3N)</td>
</tr>
<tr>
<td>nis_add_member(3N)</td>
<td>See nis_groups(3N)</td>
</tr>
<tr>
<td>nis_checkpoint(3N)</td>
<td>See nis_ping(3N)</td>
</tr>
<tr>
<td>nis_clone_object(3N)</td>
<td>See nis_subr(3N)</td>
</tr>
<tr>
<td>nis_creategroup(3N)</td>
<td>See nis_groups(3N)</td>
</tr>
<tr>
<td>nis_db(3N)</td>
<td>NIS+ Database access functions</td>
</tr>
<tr>
<td>nis_destroy_group(3N)</td>
<td>See nis_groups(3N)</td>
</tr>
<tr>
<td>nis_destroy_object(3N)</td>
<td>See nis_subr(3N)</td>
</tr>
<tr>
<td>nis_dir_cmp(3N)</td>
<td>See nis_subr(3N)</td>
</tr>
<tr>
<td>nis_domain_of(3N)</td>
<td>See nis_subr(3N)</td>
</tr>
<tr>
<td>nis_error(3N)</td>
<td>display NIS+ error messages</td>
</tr>
<tr>
<td>nis_first_entry(3N)</td>
<td>See nis_tables(3N)</td>
</tr>
<tr>
<td>nis_freenames(3N)</td>
<td>See nis_subr(3N)</td>
</tr>
<tr>
<td>nis_freesresult(3N)</td>
<td>See nis_names(3N)</td>
</tr>
<tr>
<td>nis_freeservlist(3N)</td>
<td>See nis_server(3N)</td>
</tr>
<tr>
<td>nis_freetags(3N)</td>
<td>See nis_server(3N)</td>
</tr>
<tr>
<td>nis_getnames(3N)</td>
<td>See nis_subr(3N)</td>
</tr>
<tr>
<td>nis_getservlist(3N)</td>
<td>See nis_server(3N)</td>
</tr>
<tr>
<td>nis_groups(3N)</td>
<td>NIS+ group manipulation functions</td>
</tr>
<tr>
<td>nis_is_member(3N)</td>
<td>See nis_groups(3N)</td>
</tr>
<tr>
<td>nis_leaf_of(3N)</td>
<td>See nis_subr(3N)</td>
</tr>
<tr>
<td>nis_lerror(3N)</td>
<td>See nis_error(3N)</td>
</tr>
<tr>
<td>nis_list(3N)</td>
<td>See nis_tables(3N)</td>
</tr>
<tr>
<td>nis_local_directory(3N)</td>
<td>See nis_local_names(3N)</td>
</tr>
<tr>
<td>nis_local_group(3N)</td>
<td>See nis_local_names(3N)</td>
</tr>
<tr>
<td>nis_local_host(3N)</td>
<td>See nis_local_names(3N)</td>
</tr>
<tr>
<td>nis_local_names(3N)</td>
<td>NIS+ local names</td>
</tr>
<tr>
<td>nis_local_principal(3N)</td>
<td>See nis_local_names(3N)</td>
</tr>
<tr>
<td>nis_lookup(3N)</td>
<td>See nis_names(3N)</td>
</tr>
<tr>
<td>__nis_map_group(3N)</td>
<td>See nis_groups(3N)</td>
</tr>
<tr>
<td>nis_map_group(3N)</td>
<td>See nis_groups(3N)</td>
</tr>
<tr>
<td>nis mkdir(3N)</td>
<td>See nis_server(3N)</td>
</tr>
<tr>
<td>nis_modify(3N)</td>
<td>See nis_names(3N)</td>
</tr>
</tbody>
</table>
nis_modify_entry(3N) See nis_tables(3N)
nis_name_of(3N) See nis_subr(3N)
nis_names(3N) NIS+ namespace functions
nis_next_entry(3N) See nis_tables(3N)
nis_objects(3N) NIS+ object formats
nis_perror(3N) See nis_error(3N)
nis_ping(3N) misc NIS+ log administration functions
nis_print_group_entry(3N) See nis_groups(3N)
nis_print_object(3N) See nis_subr(3N)
nis_remove(3N) See nis_names(3N)
nis_remove_entry(3N) See nis_tables(3N)
nis_removemember(3N) See nis_groups(3N)
nis_rmdir(3N) See nis_server(3N)
nis_server(3N) miscellaneous NIS+ functions
nis_servstate(3N) See nis_server(3N)
nis_sperrno(3N) See nis_error(3N)
nis_sperror(3N) See nis_error(3N)
nis_sperror_r(3N) See nis_error(3N)
nis_stats(3N) See nis_server(3N)
nis_subr(3N) NIS+ subroutines
nis_tables(3N) NIS+ table functions
nis_verifygroup(3N) See nis_groups(3N)
nl(3X) See curs_outopts(3X)
nl(3XC) enable/disable newline control
nlist(3B) get entries from symbol table
nlist(3E) get entries from name list
nl_langinfo(3C) language information
nlscall(3N) get client’s data passed via the listener
nlsprovider(3N) get name of transport provider
nlsrequest(3N) format and send listener service request message
nocbreak(3X) See curs_inopts(3X)
nocbreak(3XC) See cbreak(3XC)
nodelay(3X) See curs_inopts(3X)
nodelay(3XC) set blocking or non-blocking read
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>noecho(3X)</td>
<td>See <code>curs_inopts(3X)</code></td>
</tr>
<tr>
<td>noecho(3XC)</td>
<td>See <code>echo(3XC)</code></td>
</tr>
<tr>
<td>nonl(3X)</td>
<td>See <code>curs_outopts(3X)</code></td>
</tr>
<tr>
<td>nonl(3XC)</td>
<td>See <code>nl(3XC)</code></td>
</tr>
<tr>
<td>noqiflush(3X)</td>
<td>See <code>curs_inopts(3X)</code></td>
</tr>
<tr>
<td>noqiflush(3XC)</td>
<td>Control flush of input and output on interrupt</td>
</tr>
<tr>
<td>noraw(3X)</td>
<td>See <code>curs_inopts(3X)</code></td>
</tr>
<tr>
<td>noraw(3XC)</td>
<td>See <code>cbreak(3XC)</code></td>
</tr>
<tr>
<td>NOTE(3X)</td>
<td>Annotate source code with info for tools</td>
</tr>
<tr>
<td>_NOTE(3X)</td>
<td>See <code>NOTE(3X)</code></td>
</tr>
<tr>
<td>notimeout(3X)</td>
<td>See <code>curs_inopts(3X)</code></td>
</tr>
<tr>
<td>notimeout(3XC)</td>
<td>Set timed blocking or non-blocking read</td>
</tr>
<tr>
<td>nrand48(3C)</td>
<td>See <code>drand48(3C)</code></td>
</tr>
<tr>
<td>ntohl(3N)</td>
<td>See <code>byteorder(3N)</code></td>
</tr>
<tr>
<td>ntohl(3XN)</td>
<td>See <code>htonl(3XN)</code></td>
</tr>
<tr>
<td>ntohs(3N)</td>
<td>See <code>byteorder(3N)</code></td>
</tr>
<tr>
<td>ntohs(3XN)</td>
<td>See <code>htonl(3XN)</code></td>
</tr>
<tr>
<td>offsetof(3C)</td>
<td>Offset of structure member</td>
</tr>
<tr>
<td>opendir(3C)</td>
<td>Open directory</td>
</tr>
<tr>
<td>openlog(3)</td>
<td>See <code>syslog(3)</code></td>
</tr>
<tr>
<td>openpl(3)</td>
<td>See <code>plot(3)</code></td>
</tr>
<tr>
<td>openvt(3)</td>
<td>See <code>plot(3)</code></td>
</tr>
<tr>
<td>overlay(3X)</td>
<td>See <code>curs_overlay(3X)</code></td>
</tr>
<tr>
<td>overlay(3XC)</td>
<td>Overlap or overwrite windows</td>
</tr>
<tr>
<td>overwrite(3X)</td>
<td>See <code>curs_overlay(3X)</code></td>
</tr>
<tr>
<td>overwrite(3XC)</td>
<td>See <code>overlay(3XC)</code></td>
</tr>
<tr>
<td>p2close(3G)</td>
<td>See <code>p2open(3G)</code></td>
</tr>
<tr>
<td>p2open(3G)</td>
<td>Open, close pipes to and from a command</td>
</tr>
<tr>
<td>pair_content(3X)</td>
<td>See <code>curs_color(3X)</code></td>
</tr>
<tr>
<td>pair_content(3XC)</td>
<td>See <code>can_change_color(3XC)</code></td>
</tr>
<tr>
<td>PAIR_NUMBER(3XC)</td>
<td>See <code>can_change_color(3XC)</code></td>
</tr>
<tr>
<td>pam(3)</td>
<td>PAM (Pluggable Authentication Module)</td>
</tr>
<tr>
<td>pam_acct_mgmt(3)</td>
<td>Perform PAM account validation procedures</td>
</tr>
<tr>
<td>pam_authenticate(3)</td>
<td>Perform authentication within the PAM</td>
</tr>
<tr>
<td>C Library Functions</td>
<td>Intro (3)</td>
</tr>
<tr>
<td>---------------------</td>
<td>----------</td>
</tr>
<tr>
<td>pam_chauthtok(3)</td>
<td>perform password related functions within the PAM framework</td>
</tr>
<tr>
<td>pam_close_session(3)</td>
<td>See pam_open_session(3)</td>
</tr>
<tr>
<td>pam_end(3)</td>
<td>See pam_start(3)</td>
</tr>
<tr>
<td>pam_get_data(3)</td>
<td>See pam_set_data(3)</td>
</tr>
<tr>
<td>pam_getenv(3)</td>
<td>returns the value for a PAM environment name</td>
</tr>
<tr>
<td>pam_getenvlist(3)</td>
<td>returns a list of all the PAM environment variables</td>
</tr>
<tr>
<td>pam_get_item(3)</td>
<td>See pam_set_item(3)</td>
</tr>
<tr>
<td>pam_get_user(3)</td>
<td>PAM routine to retrieve user name</td>
</tr>
<tr>
<td>pam_open_session(3)</td>
<td>perform PAM session creation and termination operations</td>
</tr>
<tr>
<td>pam_putenv(3)</td>
<td>change or add a value to the PAM environment</td>
</tr>
<tr>
<td>pam_setcred(3)</td>
<td>modify/delete user credentials for an authentication service</td>
</tr>
<tr>
<td>pam_set_data(3)</td>
<td>PAM routines to maintain module specific state</td>
</tr>
<tr>
<td>pam_set_item(3)</td>
<td>authentication information routines for PAM</td>
</tr>
<tr>
<td>pam_sm(3)</td>
<td>PAM Service Module APIs</td>
</tr>
<tr>
<td>pam_sm_acct_mgmt(3)</td>
<td>service provider implementation for pam_acct_mgmt</td>
</tr>
<tr>
<td>pam_sm_authenticate(3)</td>
<td>service provider implementation for pam_authenticate</td>
</tr>
<tr>
<td>pam_sm_chauthtok(3)</td>
<td>service provider implementation for pam_chauthtok</td>
</tr>
<tr>
<td>pam_sm_close_session(3)</td>
<td>See pam_sm_open_session(3)</td>
</tr>
<tr>
<td>pam_sm_open_session(3)</td>
<td>service provider implementation for pam_open_session and pam_close_session</td>
</tr>
<tr>
<td>pam_sm_setcred(3)</td>
<td>service provider implementation for pam_setcred</td>
</tr>
<tr>
<td>pam_start(3)</td>
<td>authentication transaction routines for PAM</td>
</tr>
<tr>
<td>pam_strerror(3)</td>
<td>get PAM error message string</td>
</tr>
<tr>
<td>panel_above(3X)</td>
<td>panels deck traversal primitives</td>
</tr>
</tbody>
</table>

modified 4 Apr 1997

SunOS 5.6
panel_below(3X)  See panel_above(3X)
panel_hidden(3X)  See panel_show(3X)
panel_move(3X)    move a panels window on the virtual
                  screen
panel_new(3X)    create and destroy panels
panels(3X)       character based panels package
panel_show(3X)   panels deck manipulation routines
panel_top(3X)    panels deck manipulation routines
panel_update(3X) panels virtual screen refresh routine
panel_userptr(3X) associate application data with a panels
                    panel
panel_window(3X) get or set the current window of a panels
                    panel
pathfind(3G)     search for named file in named directories
pclose(3S)       See popen(3S)
pechochar(3X)    See curs_pad(3X)
pechochar(3XC)   add character and refresh window
pechowchar(3X)   See curs_pad(3X)
pecho_wchar(3XC) See pechochar(3XC)
 perror(3C)       print system error messages
pfmt(3C)         display error message in standard format
plock(3C)        lock or unlock into memory process, text,
                  or data
plot(3)          graphics interface
pmap_getmaps(3N) See rpc_soc(3N)
pmap_getport(3N) See rpc_soc(3N)
pmap_rmtcall(3N) See rpc_soc(3N)
pmap_set(3N)     See rpc_soc(3N)
pmap_unset(3N)   See rpc_soc(3N)
pnoutrefresh(3X) See newpad(3XC)
pnoutrefresh(3XC) See plot(3)
popen(3S)        initiate pipe to/from a process
pos_form_cursor(3X) See form_cursor(3X)
pos_menu_cursor(3X) See menu_cursor(3X)
post_form(3X) See form_post(3X)
post_menu(3X) See menu_post(3X)
pow(3M) power function
prefresh(3X) See curs_pad(3X)
prefresh(3XC) See newpad(3XC)
printf(3B) formatted output conversion
printf(3S) print formatted output
printw(3X) See curs_printw(3X)
prefresh(3XC) See mvprintw(3XC)
proc_service(3T) process service interfaces
psiginfo(3C) See psignal(3C)
psignal(3B) system signal messages
psignal(3C) system signal messages
ps_kill(3T) See ps_pstop(3T)
ps_lcontinue(3T) See ps_pstop(3T)
ps_lgetfregs(3T) See ps_lgetregs(3T)
ps_lgetregs(3T) routines that access the target process register in libthread_db
ps_lgetxregs(3T) See ps_lgetregs(3T)
ps_lgetxregsize(3T) See ps_lgetregs(3T)
ps_lrolloaddr(3T) See ps_pstop(3T)
ps_lsetfregs(3T) See ps_lgetregs(3T)
ps_lsetregs(3T) See ps_lgetregs(3T)
ps_lsetxregs(3T) See ps_lgetregs(3T)
ps_lstop(3T) See ps_pstop(3T)
ps_pcontinue(3T) See ps_pstop(3T)
ps_pdread(3T) interfaces in libthread_db that target process memory access
ps_pdwrite(3T) See ps_pdread(3T)
ps_pglobal_lookup(3T) looks up the symbol in the symbol table of the load object in the target process
ps_pstop(3T) process and LWP control in libthread_db
ps_ptread(3T) See ps_pdread(3T)
ps_ptwrite(3T) See ps_pdread(3T)
pthread_atfork(3T) register fork handlers
pthread_attr_destroy(3T) See pthread_attr_init(3T)
C Library Functions

- `pthread_attr_getdetachstate(3T)`
- `pthread_attr_getinherithreads(3T)`
- `pthread_attr_getschedparam(3T)`
- `pthread_attr_getschedpolicy(3T)`
- `pthread_attr_getscope(3T)`
- `pthread_attr_getstackaddr(3T)`
- `pthread_attr_getstacksize(3T)`
- `pthread_attr_init(3T)`
- `pthread_attr_setdetachstate(3T)`
- `pthread_attr_setinherithreads(3T)`
- `pthread_attr_setschedparam(3T)`
- `pthread_attr_setschedpolicy(3T)`
- `pthread_attr_setscope(3T)`
- `pthread_attr_setstackaddr(3T)`
- `pthread_attr_setstacksize(3T)`
- `pthread_cancel(3T)`
- `pthread_cleanup_pop(3T)`
- `pthread_cleanup_push(3T)`
- `pthread_condattr_destroy(3T)`
- `pthread_condattr_getpshared(3T)`
- `pthread_condattr_init(3T)`
- `pthread_condattr_setpshared(3T)`
- `pthread_cond_broadcast(3T)`
- `pthread_cond_destroy(3T)`
- `pthread_cond_init(3T)`
- `pthread_cond_signal(3T)`
- `pthread_cond_timedwait(3T)`
- `pthread_cond_wait(3T)`
- `pthread_create(3T)`
- `pthread_detach(3T)`
- `pthread_equal(3T)`
- `pthread_exit(3T)`
- `pthread_getschedparam(3T)`
- `pthread_getspecific(3T)`
- `pthread_setschedparam(3T)`
- `pthread_setspecific(3T)`

See `pthread_attr_init(3T)`
See `pthread_attr_init(3T)`
See `pthread_attr_init(3T)`
See `pthread_attr_init(3T)`
See `pthread_attr_init(3T)`
See `pthread_attr_init(3T)`
See `pthread_attr_init(3T)`
See `pthread_attr_init(3T)`
See `pthread_attr_init(3T)`
See `pthread_attr_init(3T)`
See `pthread_attr_init(3T)`
See `pthread_attr_init(3T)`
See `pthread_attr_init(3T)`
See `pthread_attr_init(3T)`
See `condition(3T)`
See `condition(3T)`
See `condition(3T)`
See `condition(3T)`
See `condition(3T)`
See `condition(3T)`
See `condition(3T)`
dynamically detaching a thread
compare thread IDs
thread termination

thread creation attributes
condition variable initialization attributes
cancel execution of a thread
pop a thread cancellation cleanup handler
push a thread cancellation cleanup handler

SunOS 5.6

modified 4 Apr 1997
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pthread_join(3T)</code></td>
<td>Wait for thread termination</td>
</tr>
<tr>
<td><code>pthread_key_create(3T)</code></td>
<td>Thread-specific-data functions</td>
</tr>
<tr>
<td><code>pthread_key_delete(3T)</code></td>
<td>See <code>pthread_key_create(3T)</code></td>
</tr>
<tr>
<td><code>pthread_kill(3T)</code></td>
<td>Send a signal to a thread</td>
</tr>
<tr>
<td><code>pthread_mutexattr_destroy(3T)</code></td>
<td>See <code>pthread_mutexattr_init(3T)</code></td>
</tr>
<tr>
<td><code>pthread_mutexattr_getprioceiling(3T)</code></td>
<td>See <code>pthread_mutexattr_init(3T)</code></td>
</tr>
<tr>
<td><code>pthread_mutexattr_getprotocol(3T)</code></td>
<td>See <code>pthread_mutexattr_init(3T)</code></td>
</tr>
<tr>
<td><code>pthread_mutexattr_getpshared(3T)</code></td>
<td>See <code>pthread_mutexattr_init(3T)</code></td>
</tr>
<tr>
<td><code>pthread_mutex_destroy(3T)</code></td>
<td>See <code>mutex(3T)</code></td>
</tr>
<tr>
<td><code>pthread_mutex_init(3T)</code></td>
<td>See <code>mutex(3T)</code></td>
</tr>
<tr>
<td><code>pthread_mutex_lock(3T)</code></td>
<td>Change the priority ceiling of a mutex</td>
</tr>
<tr>
<td><code>pthread_mutex_setprioceiling(3T)</code></td>
<td>See <code>mutex(3T)</code></td>
</tr>
<tr>
<td><code>pthread_mutex_trylock(3T)</code></td>
<td>See <code>mutex(3T)</code></td>
</tr>
<tr>
<td><code>pthread_mutex_unlock(3T)</code></td>
<td>Dynamic package initialization</td>
</tr>
<tr>
<td><code>pthread_once(3T)</code></td>
<td>Get calling thread’s ID</td>
</tr>
<tr>
<td><code>pthread_self(3T)</code></td>
<td>Enable or disable cancellation</td>
</tr>
<tr>
<td><code>pthread_setcancelstate(3T)</code></td>
<td>Set the cancellation type of a thread</td>
</tr>
<tr>
<td><code>pthread_setschedparam(3T)</code></td>
<td>Dynamic access to thread scheduling</td>
</tr>
<tr>
<td><code>pthread_setspecific(3T)</code></td>
<td>See <code>pthread_key_create(3T)</code></td>
</tr>
<tr>
<td><code>pthread_sigmask(3T)</code></td>
<td>Change and/or examine calling thread’s signal mask</td>
</tr>
<tr>
<td><code>pthread_testsipancel(3T)</code></td>
<td>Create cancellation point in the calling thread</td>
</tr>
<tr>
<td><code>ptsname(3C)</code></td>
<td>Get name of the slave pseudo-terminal device</td>
</tr>
<tr>
<td><code>publickey(3N)</code></td>
<td>See <code>getpublickey(3N)</code></td>
</tr>
<tr>
<td><code>putc(3S)</code></td>
<td>Put character or word on a stream</td>
</tr>
<tr>
<td><code>putchar(3S)</code></td>
<td>See <code>putc(3S)</code></td>
</tr>
<tr>
<td><code>putchar_unlocked(3S)</code></td>
<td></td>
</tr>
</tbody>
</table>

modified 4 Apr 1997

SunOS 5.6

3-67
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>putc_unlocked(3S)</td>
<td>See putc(3S)</td>
</tr>
<tr>
<td>putenv(3C)</td>
<td>change or add value to environment</td>
</tr>
<tr>
<td>putmntent(3C)</td>
<td>See getmntent(3C)</td>
</tr>
<tr>
<td>putp(3X)</td>
<td>See curs_terminfo(3X)</td>
</tr>
<tr>
<td>putp(3XC)</td>
<td>apply padding information and output string</td>
</tr>
<tr>
<td>putpwent(3C)</td>
<td>write password file entry</td>
</tr>
<tr>
<td>puts(3S)</td>
<td>put a string on a stream</td>
</tr>
<tr>
<td>putspent(3C)</td>
<td>write shadow password file entry</td>
</tr>
<tr>
<td>pututline(3C)</td>
<td>See getutent(3C)</td>
</tr>
<tr>
<td>pututxline(3C)</td>
<td>See getutxent(3C)</td>
</tr>
<tr>
<td>putw(3S)</td>
<td>See putc(3S)</td>
</tr>
<tr>
<td>putwc(3S)</td>
<td>put wide character on a stream</td>
</tr>
<tr>
<td>putwchar(3S)</td>
<td>put wide character on stdout stream</td>
</tr>
<tr>
<td>putwin(3X)</td>
<td>See curs_util(3X)</td>
</tr>
<tr>
<td>putwin(3XC)</td>
<td>See getwin(3XC)</td>
</tr>
<tr>
<td>putws(3S)</td>
<td>convert a string of Process Code characters to EUC characters</td>
</tr>
<tr>
<td>qeconvert(3)</td>
<td>See econvert(3)</td>
</tr>
<tr>
<td>qfconvert(3)</td>
<td>See econvert(3)</td>
</tr>
<tr>
<td>qgconvert(3)</td>
<td>See econvert(3)</td>
</tr>
<tr>
<td>qiflush(3X)</td>
<td>See curs_inopts(3X)</td>
</tr>
<tr>
<td>qiflush(3XC)</td>
<td>See noqiflush(3XC)</td>
</tr>
<tr>
<td>qsort(3C)</td>
<td>quick sort</td>
</tr>
<tr>
<td>quadruple_to_decimal(3)</td>
<td>See floating_to_decimal(3)</td>
</tr>
<tr>
<td>rac_drop(3N)</td>
<td>See rpc_rac(3N)</td>
</tr>
<tr>
<td>rac_poll(3N)</td>
<td>See rpc_rac(3N)</td>
</tr>
<tr>
<td>rac_recv(3N)</td>
<td>See rpc_rac(3N)</td>
</tr>
<tr>
<td>rac_send(3N)</td>
<td>See rpc_rac(3N)</td>
</tr>
<tr>
<td>raise(3C)</td>
<td>send signal to program</td>
</tr>
<tr>
<td>rand(3B)</td>
<td>simple random number generator</td>
</tr>
<tr>
<td>rand(3C)</td>
<td>simple random-number generator</td>
</tr>
<tr>
<td>random(3C)</td>
<td>pseudorandom number functions</td>
</tr>
<tr>
<td>rand_r(3C)</td>
<td>See rand(3C)</td>
</tr>
<tr>
<td>raw(3X)</td>
<td>See curs_inopts(3X)</td>
</tr>
<tr>
<td>raw(3XC)</td>
<td>See cbreak(3XC)</td>
</tr>
</tbody>
</table>
rcmd(3N) routines for returning a stream to a remote command
readdir(3B) read a directory entry
readdir(3C) read directory
readdir_r(3C) See readdir(3C)
readvtoc(3X) read and write a disk’s VTOC
realloc(3C) See malloc(3C)
realloc(3X) See bsdmalloc(3X)
realloc(3X) See malloc(3X)
realloc(3X) See mapmalloc(3X)
realloc(3X) See watchmalloc(3X)
realpath(3C) resolve pathname
reboot(3C) reboot system or halt processor
re_comp(3C) compile and execute regular expressions
recv(3N) receive a message from a socket
recv(3XN) receive a message from a connected socket
recvfrom(3N) See recv(3N)
recvfrom(3XN) receive a message from a socket
recvmsg(3N) See recv(3N)
recvmsg(3XN) receive a message from a socket
redrawwin(3X) redraw screen or portion of screen
redrawwin(3XC) See re_exec(3C)
re_exec(3C) See re_comp(3C)
refresh(3X) See curs_refresh(3X)
refresh(3XC) See doupdate(3XC)
regcmp(3C) compile and execute regular expression
regcomp(3C) regular expression matching
regerror(3C) See regcomp(3C)
regex(3C) See regcmp(3C)
regexec(3C) See regcomp(3C)
regexpr(3G) regular expression compile and match routines
regfree(3C) See regcomp(3C)
registerrpc(3N) See rpc_soc(3N)
remainder(3M) remainder function

modified 4 Apr 1997 SunOS 5.6

3-69
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>remove(3C)</td>
<td>remove file</td>
</tr>
<tr>
<td>remque(3C)</td>
<td>See insque(3C)</td>
</tr>
<tr>
<td>replace_panel(3X)</td>
<td>See panel_window(3X)</td>
</tr>
<tr>
<td>reset_prog_mode(3X)</td>
<td>See curs_kernel(3X)</td>
</tr>
<tr>
<td>reset_prog_mode(3XC)</td>
<td>See def_prog_mode(3XC)</td>
</tr>
<tr>
<td>reset_shell_mode(3X)</td>
<td>See curs_kernel(3X)</td>
</tr>
<tr>
<td>reset_shell_mode(3XC)</td>
<td>See def_prog_mode(3XC)</td>
</tr>
<tr>
<td>resety(3X)</td>
<td>restore/save terminal modes</td>
</tr>
<tr>
<td>res_init(3N)</td>
<td>See resolver(3N)</td>
</tr>
<tr>
<td>res_mkquery(3N)</td>
<td>resolver routines</td>
</tr>
<tr>
<td>resolver(3N)</td>
<td>See resolver(3N)</td>
</tr>
<tr>
<td>res_query(3N)</td>
<td>See resolver(3N)</td>
</tr>
<tr>
<td>res_search(3N)</td>
<td>See resolver(3N)</td>
</tr>
<tr>
<td>res_send(3N)</td>
<td>See resolver(3N)</td>
</tr>
<tr>
<td>restartterm(3X)</td>
<td>See del_curterm(3XC)</td>
</tr>
<tr>
<td>restartterm(3XC)</td>
<td>See del_curterm(3XC)</td>
</tr>
<tr>
<td>rewind(3S)</td>
<td>reset file position indicator in a stream</td>
</tr>
<tr>
<td>rewinddir(3C)</td>
<td>reset position of directory stream to the beginning of a directory</td>
</tr>
<tr>
<td>rexec(3N)</td>
<td>return stream to a remote command</td>
</tr>
<tr>
<td>rindex(3C)</td>
<td>See index(3C)</td>
</tr>
<tr>
<td>rint(3M)</td>
<td>round-to-nearest integral value</td>
</tr>
<tr>
<td>ripoffline(3X)</td>
<td>See curs_kernel(3X)</td>
</tr>
<tr>
<td>ripoffline(3XC)</td>
<td>reserve screen line for dedicated purpose</td>
</tr>
<tr>
<td>rmdirp(3G)</td>
<td>See mkdirp(3G)</td>
</tr>
<tr>
<td>rusers(3N)</td>
<td>See rusers(3N)</td>
</tr>
<tr>
<td>rpc(3N)</td>
<td>library routines for remote procedure calls</td>
</tr>
<tr>
<td>rpcb_getaddr(3N)</td>
<td>See rpcbind(3N)</td>
</tr>
<tr>
<td>rpcb_getmaps(3N)</td>
<td>See rpcbind(3N)</td>
</tr>
<tr>
<td>rpcb_gettime(3N)</td>
<td>See rpcbind(3N)</td>
</tr>
<tr>
<td>rpcbind(3N)</td>
<td>library routines for RPC bind service</td>
</tr>
<tr>
<td>rpc_rmtcall(3N)</td>
<td>See rpcbind(3N)</td>
</tr>
<tr>
<td>rpc_broadcast(3N)</td>
<td>See rpc_clnt_calls(3N)</td>
</tr>
<tr>
<td>rpc_broadcast_exp(3N)</td>
<td>See rpc_clnt_calls(3N)</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td>rpcb_set</td>
<td>See rpcbind(3N)</td>
</tr>
<tr>
<td>rpcb_unset</td>
<td>See rpcbind(3N)</td>
</tr>
<tr>
<td>rpc_call</td>
<td>See rpc_clnt_calls(3N)</td>
</tr>
<tr>
<td>rpc_clnt_auth</td>
<td>library routines for client side remote procedure call authentication</td>
</tr>
<tr>
<td>rpc_clnt_calls</td>
<td>library routines for client side calls</td>
</tr>
<tr>
<td>rpc_clnt_create</td>
<td>library routines for dealing with creation and manipulation of CLIENT handles</td>
</tr>
<tr>
<td>rpc_control</td>
<td>library routine for manipulating global RPC attributes for client and server applications</td>
</tr>
<tr>
<td>rpc_createerr</td>
<td>See rpc_clnt_create(3N)</td>
</tr>
<tr>
<td>rpc_rac</td>
<td>remote asynchronous calls</td>
</tr>
<tr>
<td>rpc_reg</td>
<td>See rpc_svc_reg(3N)</td>
</tr>
<tr>
<td>rpc_soc</td>
<td>obsolete library routines for RPC</td>
</tr>
<tr>
<td>rpc_svc_calls</td>
<td>library routines for RPC servers</td>
</tr>
<tr>
<td>rpc_svc_create</td>
<td>library routines for the creation of server handles</td>
</tr>
<tr>
<td>rpc_svc_err</td>
<td>library routines for server side remote procedure call errors</td>
</tr>
<tr>
<td>rpc_svc_reg</td>
<td>library routines for registering servers</td>
</tr>
<tr>
<td>rpc_xdr</td>
<td>XDR library routines for remote procedure calls</td>
</tr>
<tr>
<td>rresvport</td>
<td>See rcmd(3N)</td>
</tr>
<tr>
<td>rstat</td>
<td>get performance data from remote kernel</td>
</tr>
<tr>
<td>ruserok</td>
<td>See rcmd(3N)</td>
</tr>
<tr>
<td>rusers</td>
<td>return information about users on remote machines</td>
</tr>
<tr>
<td>rwall</td>
<td>write to specified remote machines</td>
</tr>
<tr>
<td>rwlock</td>
<td>multiple readers, single writer locks</td>
</tr>
<tr>
<td>rwlock_destroy</td>
<td>See rwlock(3T)</td>
</tr>
<tr>
<td>rwlock_init</td>
<td>See rwlock(3T)</td>
</tr>
<tr>
<td>rw_rdlock</td>
<td>See rwlock(3T)</td>
</tr>
<tr>
<td>rw_tryrdlock</td>
<td>See rwlock(3T)</td>
</tr>
<tr>
<td>rw_trywrlock</td>
<td>See rwlock(3T)</td>
</tr>
<tr>
<td>rw_unlock</td>
<td>See rwlock(3T)</td>
</tr>
<tr>
<td>rw_wrlock</td>
<td>See rwlock(3T)</td>
</tr>
</tbody>
</table>
savetty(3X) See curs_kernel(3X)
savetty(3XC) See reetty(3XC)
scalb(3M) load exponent of a radix-independent floating-point number
scalbn(3M) load exponent of a radix-independent floating-point number
scale_form(3X) See form_win(3X)
scale_menu(3X) See menu_win(3X)
scandir(3B) scan a directory
scanf(3S) convert formatted input
scanw(3X) See curs_scanw(3X)
scanw(3XC) See mvscanw(3XC)
schedctl_exit(3X) See schedctl_init(3X)
schedctl_init(3X) preemption control
schedctl_lookup(3X) See schedctl_init(3X)
schedctl_start(3X) See schedctl_init(3X)
schedctl_stop(3X) See schedctl_init(3X)
sched_getparam(3R) See sched_setparam(3R)
sched_get_priority_max(3R) get scheduling parameter limits
sched_get_priority_min(3R) See sched_get_priority_max(3R)
sched_getscheduler(3R) See sched_setscheduler(3R)
sched_rr_get_interval(3R) See sched_get_priority_max(3R)
sched_setparam(3R) set/get scheduling parameters
sched_setscheduler(3R) set/get scheduling policy and scheduling parameters
sched_yield(3R) yield processor
scr_dump(3X) See curs_scr_dump(3X)
scr_dump(3XC) write screen contents to/from a file
scr_init(3X) See curs_scr_dump(3X)
scr_init(3XC) See scr_dump(3XC)
scrl(3X) See curs_scroll(3X)
scrl(3XC) scroll a window
scroll(3X) See curs_scroll(3X)
scroll(3XC) See scrl(3XC)
scrollok(3X) See curs_outopts(3X)
scrollok(3XC) See clearok(3XC)
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>scr_restore(3X)</td>
<td>See curs_scr_dump(3X)</td>
</tr>
<tr>
<td>scr_restore(3XC)</td>
<td>See scr_dump(3XC)</td>
</tr>
<tr>
<td>scr_set(3X)</td>
<td>See curs_scr_dump(3X)</td>
</tr>
<tr>
<td>scr_set(3XC)</td>
<td>See scr_dump(3XC)</td>
</tr>
<tr>
<td>seconvert(3)</td>
<td>library routines for secure remote procedure calls</td>
</tr>
<tr>
<td>secure_rpc(3N)</td>
<td>See drand48(3C)</td>
</tr>
<tr>
<td>seed48(3C)</td>
<td>set position of directory stream</td>
</tr>
<tr>
<td>seekdir(3C)</td>
<td>synchronous I/O multiplexing</td>
</tr>
<tr>
<td>select(3C)</td>
<td>See semaphore(3T)</td>
</tr>
<tr>
<td>sema_destroy(3T)</td>
<td>semaphores</td>
</tr>
<tr>
<td>sema_init(3T)</td>
<td>See semaphore(3T)</td>
</tr>
<tr>
<td>semaphore(3T)</td>
<td>close a named semaphore</td>
</tr>
<tr>
<td>sema_post(3T)</td>
<td>destroy an unnamed semaphore</td>
</tr>
<tr>
<td>sema_trywait(3T)</td>
<td>get the value of a semaphore</td>
</tr>
<tr>
<td>sema_wait(3T)</td>
<td>initialize an unnamed semaphore</td>
</tr>
<tr>
<td>sem_close(3R)</td>
<td>initialize/open a named semaphore</td>
</tr>
<tr>
<td>sem_destroy(3R)</td>
<td>increment the count of a semaphore</td>
</tr>
<tr>
<td>sem_getvalue(3R)</td>
<td>See sem_wait(3R)</td>
</tr>
<tr>
<td>sem_init(3R)</td>
<td>remove a named semaphore</td>
</tr>
<tr>
<td>sem_open(3R)</td>
<td>acquire or wait for a semaphore</td>
</tr>
<tr>
<td>sem_post(3R)</td>
<td>send a message from a socket</td>
</tr>
<tr>
<td>sem_getmsg(3N)</td>
<td>send a message on a socket</td>
</tr>
<tr>
<td>sendmsg(3N)</td>
<td>See send(3N)</td>
</tr>
<tr>
<td>sendmsg(3XN)</td>
<td>send a message on a socket using a message structure</td>
</tr>
<tr>
<td>sendto(3N)</td>
<td>See send(3N)</td>
</tr>
<tr>
<td>sendto(3XN)</td>
<td>send a message on a socket</td>
</tr>
<tr>
<td>setac(3)</td>
<td>See getacinfo(3)</td>
</tr>
<tr>
<td>setauclass(3)</td>
<td>See getauclassest(3)</td>
</tr>
<tr>
<td>setauevent(3)</td>
<td>See getauevent(3)</td>
</tr>
<tr>
<td>setauuser(3)</td>
<td>See getauusername(3)</td>
</tr>
<tr>
<td><strong>setbuf(3S)</strong></td>
<td>assign buffering to a stream</td>
</tr>
<tr>
<td>----------------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td><strong>setbuffer(3C)</strong></td>
<td>assign buffering to a stream</td>
</tr>
<tr>
<td><strong>setcat(3C)</strong></td>
<td>define default catalog</td>
</tr>
<tr>
<td><strong>setchar(3XC)</strong></td>
<td>set a cchar_t type character from a wide character and rendition</td>
</tr>
<tr>
<td><strong>set_current_field(3X)</strong></td>
<td>See form_page(3X)</td>
</tr>
<tr>
<td><strong>set_current_item(3X)</strong></td>
<td>See menu_item_current(3X)</td>
</tr>
<tr>
<td><strong>set_curterm(3XC)</strong></td>
<td>See del_curterm(3XC)</td>
</tr>
<tr>
<td><strong>set_field_back(3X)</strong></td>
<td>See form_field_attributes(3X)</td>
</tr>
<tr>
<td><strong>set_field_buffer(3X)</strong></td>
<td>See form_field_buffer(3X)</td>
</tr>
<tr>
<td><strong>set_field_fore(3X)</strong></td>
<td>See form_field_attributes(3X)</td>
</tr>
<tr>
<td><strong>set_field_init(3X)</strong></td>
<td>See form_field(3X)</td>
</tr>
<tr>
<td><strong>set_field_just(3X)</strong></td>
<td>See form_field_just(3X)</td>
</tr>
<tr>
<td><strong>set_field_opts(3X)</strong></td>
<td>See form_field_opts(3X)</td>
</tr>
<tr>
<td><strong>set_field_pad(3X)</strong></td>
<td>See form_field_attributes(3X)</td>
</tr>
<tr>
<td><strong>set_field_status(3X)</strong></td>
<td>See form_field_buffer(3X)</td>
</tr>
<tr>
<td><strong>set_field_term(3X)</strong></td>
<td>See form_hook(3X)</td>
</tr>
<tr>
<td><strong>set_field_type(3X)</strong></td>
<td>See form_field_validation(3X)</td>
</tr>
<tr>
<td><strong>set_fieldtype_arg(3X)</strong></td>
<td>See form_fieldtype(3X)</td>
</tr>
<tr>
<td><strong>set_fieldtype_choice(3X)</strong></td>
<td>See form_fieldtype(3X)</td>
</tr>
<tr>
<td><strong>set_field_userptr(3X)</strong></td>
<td>See form_field_userptr(3X)</td>
</tr>
<tr>
<td><strong>set_form_fields(3X)</strong></td>
<td>See form_field(3X)</td>
</tr>
<tr>
<td><strong>set_form_init(3X)</strong></td>
<td>See form_hook(3X)</td>
</tr>
<tr>
<td><strong>set_form_opts(3X)</strong></td>
<td>See form_opts(3X)</td>
</tr>
<tr>
<td><strong>set_form_page(3X)</strong></td>
<td>See form_page(3X)</td>
</tr>
<tr>
<td><strong>set_form_sub(3X)</strong></td>
<td>See form_win(3X)</td>
</tr>
<tr>
<td><strong>set_form_term(3X)</strong></td>
<td>See form_hook(3X)</td>
</tr>
<tr>
<td><strong>set_form_userptr(3X)</strong></td>
<td>See form_userptr(3X)</td>
</tr>
<tr>
<td><strong>set_form_win(3X)</strong></td>
<td>See form_win(3X)</td>
</tr>
<tr>
<td><strong>setgrent(3C)</strong></td>
<td>See getgrnam(3C)</td>
</tr>
<tr>
<td><strong>sethostent(3N)</strong></td>
<td>See gethostbyname(3N)</td>
</tr>
<tr>
<td><strong>sethostent(3XN)</strong></td>
<td>See endhostent(3XN)</td>
</tr>
<tr>
<td><strong>sethostname(3C)</strong></td>
<td>See gethostname(3C)</td>
</tr>
<tr>
<td><strong>set_item_init(3X)</strong></td>
<td>See menu_hook(3X)</td>
</tr>
</tbody>
</table>
set_item_opts(3X) See menu_item_opts(3X)
set_item_term(3X) See menu_hook(3X)
set_item_userptr(3X) See menu_item_userptr(3X)
set_item_value(3X) See menu_item_value(3X)
setjmp(3B) non-local goto
_setjmp(3B) See setjmp(3B)
setjmp(3C) non-local goto
_setjmp(3C) See _longjmp(3C)
setkey(3C) set encoding key
define the label for pfmt() and fmt()
setlabel(3C) See setbuffer(3C)
setlinebuf(3C) modify and query a program's locale
setlocale(3C) See syslog(3)
setlogmask(3) See form_field_buffer(3X)
set_max_field(3X) See menu_attributes(3X)
set_menu_back(3X) See menu_attributes(3X)
set_menu_fore(3X) See menu_format(3X)
set_menu_format(3X) See menu_attributes(3X)
set_menu_grey(3X) See menu_attributes(3X)
set_menu_init(3X) See menu_hook(3X)
set_menu_items(3X) See menu_items(3X)
set_menu_mark(3X) See menu_mark(3X)
set_menu_opts(3X) See menu_opts(3X)
set_menu_pad(3X) See menu_attributes(3X)
set_menu_pattern(3X) See menu_pattern(3X)
set_menu_sub(3X) See menu_win(3X)
set_menu_term(3X) See menu_hook(3X)
set_menu_userptr(3X) See menu_userptr(3X)
set_menu_win(3X) See menu_win(3X)
setnetconfig(3N) See getnetconfig(3N)
setnetent(3N) See getnetbyname(3N)
setnetent(3XN) See endnetent(3XN)
setnetgrent(3N) See getnetgrent(3N)
setnetpath(3N) See getnetpath(3N)
set_new_page(3X) See form_new_page(3X)
set_panel_userptr(3X) See panel_userptr(3X)
setpriority(3C)  See getpriority(3C)
setprotoent(3N)  See getprotobynamelisting(3N)
setprotoent(3XN)  See endprotoent(3XN)
setpwnam(3C)  See getpwnam(3C)
setrpcreent(3N)  See getrpcreent(3N)
setprotoent(3X)  See curs_outopts(3X)
setprotoent(3XC)  See clearok(3XC)
setservent(3N)  See getservbyname(3N)
setservent(3XN)  See endservent(3XN)
setsockopt(3N)  See getsockopt(3N)
setsockopt(3XN)  set the socket options
setspent(3C)  See getspnam(3C)
setstate(3C)  See random(3C)
setsysx(3X)  See curs_kernel(3X)
set_term(3X)  See curs_initscr(3X)
setterm(3X)  See curs_terminfo(3X)
setterm(3XC)  See del_curterm(3XC)
set_term(3XC)  switch between terminals
settimeofday(3B)  See gettimeofday(3B)
settimeofday(3C)  See gettimeofday(3C)
set_top_row(3X)  See menu_item_current(3X)
setupterm(3X)  See curs_terminfo(3X)
setupterm(3XC)  See del_curterm(3XC)
setusershell(3C)  See getusershell(3C)
setutent(3C)  See getutent(3C)
setutxent(3C)  See getutxent(3C)
setvbuf(3S)  See setbuf(3S)
sfconvert(3)  See econvert(3)
sgconvert(3)  See econvert(3)
shm_open(3R)  open a shared memory object
shm_unlink(3R)  remove a shared memory object
show_panel(3X)  See panel_show(3X)
shutdown(3N)  shut down part of a full-duplex connection
shutdown(3XN)  shut down socket send and receive operations
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sig2str(3C)</td>
<td>See str2sig(3C)</td>
</tr>
<tr>
<td>sigaddset(3C)</td>
<td>See sigsetops(3C)</td>
</tr>
<tr>
<td>sigblock(3B)</td>
<td>block signals</td>
</tr>
<tr>
<td>sigdelset(3C)</td>
<td>See sigsetops(3C)</td>
</tr>
<tr>
<td>sigemptyset(3C)</td>
<td>See sigsetops(3C)</td>
</tr>
<tr>
<td>sigfillset(3C)</td>
<td>See sigsetops(3C)</td>
</tr>
<tr>
<td>sigfpe(3)</td>
<td>signal handling for specific SIGFPE codes</td>
</tr>
<tr>
<td>sighold(3C)</td>
<td>See signal(3C)</td>
</tr>
<tr>
<td>sigignore(3C)</td>
<td>See signal(3C)</td>
</tr>
<tr>
<td>siginterrupt(3B)</td>
<td>allow signals to interrupt functions</td>
</tr>
<tr>
<td>sigismember(3C)</td>
<td>See sigsetops(3C)</td>
</tr>
<tr>
<td>siglongjmp(3C)</td>
<td>See sigblock(3B)</td>
</tr>
<tr>
<td>sigmask(3B)</td>
<td>simplified software signal facilities</td>
</tr>
<tr>
<td>signal(3B)</td>
<td>simplified signal management for application processes</td>
</tr>
<tr>
<td>signal(3C)</td>
<td></td>
</tr>
<tr>
<td>significand(3M)</td>
<td>significand function</td>
</tr>
<tr>
<td>sigpause(3B)</td>
<td>See sigblock(3B)</td>
</tr>
<tr>
<td>sigpause(3C)</td>
<td>See signal(3C)</td>
</tr>
<tr>
<td>sigqueue(3R)</td>
<td>queue a signal to a process</td>
</tr>
<tr>
<td>sigrelse(3C)</td>
<td>See signal(3C)</td>
</tr>
<tr>
<td>sigset(3C)</td>
<td>See signal(3C)</td>
</tr>
<tr>
<td>sigsetjmp(3C)</td>
<td>See sigblock(3B)</td>
</tr>
<tr>
<td>sigsetmask(3B)</td>
<td>See sigblock(3B)</td>
</tr>
<tr>
<td>sigsetops(3C)</td>
<td>manipulate sets of signals</td>
</tr>
<tr>
<td>sigstack(3B)</td>
<td>set and/or get signal stack context</td>
</tr>
<tr>
<td>sigstack(3C)</td>
<td>set and/or get alternate signal stack context</td>
</tr>
<tr>
<td>sigtimedwait(3R)</td>
<td>See sigwaitinfo(3R)</td>
</tr>
<tr>
<td>sigvec(3B)</td>
<td>software signal facilities</td>
</tr>
<tr>
<td>sigwaitinfo(3R)</td>
<td>wait for queued signals</td>
</tr>
<tr>
<td>sin(3M)</td>
<td>sine function</td>
</tr>
<tr>
<td>single_to_decimal(3)</td>
<td>See floating_to_decimal(3)</td>
</tr>
<tr>
<td>sinh(3M)</td>
<td>hyperbolic sine function</td>
</tr>
<tr>
<td>sleep(3B)</td>
<td>suspend execution for interval</td>
</tr>
<tr>
<td>sleep(3C)</td>
<td>suspend execution for interval</td>
</tr>
</tbody>
</table>

modified 4 Apr 1997

SunOS 5.6

3-77
slk_attroff(3X) See curs_slk(3X)
slk_attroff(3XC) manipulate soft labels
slk_attr_off(3XC) See slk_attroff(3XC)
slk_attroff(3X) See curs_slk(3X)
slk_attr_on(3XC) See slk_attroff(3XC)
slk_attroff(3XC) See slk_attroff(3XC)
slk_attr_off(3X) See curs_slk(3X)
slk_attr_set(3XC) See slk_attroff(3XC)
slk_attrset(3X) See curs_slk(3X)
slk_attr_set(3XC) See slk_attroff(3XC)
slk_attrset(3XC) See slk_attroff(3XC)
slk_clear(3X) See curs_slk(3X)
slk_clear(3XC) See slk_attroff(3XC)
slk_color(3XC) See slk_attroff(3XC)
slk_init(3X) See curs_slk(3X)
slk_init(3XC) See slk_attroff(3XC)
slk_label(3X) See curs_slk(3X)
slk_label(3XC) See slk_attroff(3XC)
slk_noutrefresh(3X) See curs_slk(3X)
slk_noutrefresh(3XC) See slk_attroff(3XC)
slk_refresh(3X) See curs_slk(3X)
slk_refresh(3XC) See slk_attroff(3XC)
slk_restore(3X) See curs_slk(3X)
slk_restore(3XC) See slk_attroff(3XC)
slk_set(3X) See curs_slk(3X)
slk_set(3XC) See slk_attroff(3XC)
slk_touch(3X) See curs_slk(3X)
slk_touch(3XC) See slk_attroff(3XC)
slk_wset(3XC) See slk_attroff(3XC)
sprintf(3S) See printf(3S)
socket(3N) create an endpoint for communication
socket(3XN) create an endpoint for communication
socketpair(3N) create a pair of connected sockets
socketpair(3XN) create a pair of connected sockets
space(3) See plot(3)
spray(3N) scatter data in order to test the network
sprintf(3B) See printf(3B)
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sprintf(3S)</code></td>
<td>See <code>printf(3S)</code></td>
</tr>
<tr>
<td><code>sqrt(3M)</code></td>
<td>square root function</td>
</tr>
<tr>
<td><code>srand(3B)</code></td>
<td>See <code>rand(3B)</code></td>
</tr>
<tr>
<td><code>srand(3C)</code></td>
<td>See <code>rand(3C)</code></td>
</tr>
<tr>
<td><code>srand48(3C)</code></td>
<td>See <code>drand48(3C)</code></td>
</tr>
<tr>
<td><code>sscanf(3S)</code></td>
<td>See <code>scanf(3S)</code></td>
</tr>
<tr>
<td><code>ssignal(3C)</code></td>
<td>software signals</td>
</tr>
<tr>
<td><code>standend(3X)</code></td>
<td>See <code>curs_attr(3X)</code></td>
</tr>
<tr>
<td><code>standend(3XC)</code></td>
<td>set/clear window attributes</td>
</tr>
<tr>
<td><code>standout(3X)</code></td>
<td>See <code>curs_attr(3X)</code></td>
</tr>
<tr>
<td><code>standout(3XC)</code></td>
<td>See <code>standend(3XC)</code></td>
</tr>
<tr>
<td><code>start_color(3X)</code></td>
<td>See <code>curs_color(3X)</code></td>
</tr>
<tr>
<td><code>SSA劑IsAlive(3X)</code></td>
<td>Sun Solstice Enterprise Agent registration and communication helper functions</td>
</tr>
<tr>
<td><code>SSAGetTrapPort(3X)</code></td>
<td>See <code>SSA劑IsAlive(3X)</code></td>
</tr>
<tr>
<td><code>SSAOidCmp(3X)</code></td>
<td>Sun Solstice Enterprise Agent OID helper functions</td>
</tr>
<tr>
<td><code>SSAOidCpy(3X)</code></td>
<td>See <code>SSAOidCmp(3X)</code></td>
</tr>
<tr>
<td><code>SSAOidDup(3X)</code></td>
<td>See <code>SSAOidCmp(3X)</code></td>
</tr>
<tr>
<td><code>SSAOidFree(3X)</code></td>
<td>See <code>SSAOidCmp(3X)</code></td>
</tr>
<tr>
<td><code>SSAOidInit(3X)</code></td>
<td>See <code>SSAOidCmp(3X)</code></td>
</tr>
<tr>
<td><code>SSAOidNew(3X)</code></td>
<td>See <code>SSAOidCmp(3X)</code></td>
</tr>
<tr>
<td><code>SSAOidString(3X)</code></td>
<td>See <code>SSAOidCmp(3X)</code></td>
</tr>
<tr>
<td><code>SSAOidStrToOid(3X)</code></td>
<td>See <code>SSAOidCmp(3X)</code></td>
</tr>
<tr>
<td><code>SSAOidZero(3X)</code></td>
<td>See <code>SSAOidCmp(3X)</code></td>
</tr>
<tr>
<td><code>SSARegSubagent(3X)</code></td>
<td>See <code>SSA劑IsAlive(3X)</code></td>
</tr>
<tr>
<td><code>SSARegSubtable(3X)</code></td>
<td>See <code>SSA劑IsAlive(3X)</code></td>
</tr>
<tr>
<td><code>SSARegSubtree(3X)</code></td>
<td>See <code>SSA劑IsAlive(3X)</code></td>
</tr>
<tr>
<td><code>SSASendTrap(3X)</code></td>
<td>See <code>SSA劑IsAlive(3X)</code></td>
</tr>
<tr>
<td><code>SSAStringCpy(3X)</code></td>
<td>Sun Solstice Enterprise Agent string helper functions</td>
</tr>
<tr>
<td><code>SSAStringInit(3X)</code></td>
<td>See <code>SSAStringCpy(3X)</code></td>
</tr>
<tr>
<td><code>SSAStringToChar(3X)</code></td>
<td>See <code>SSAStringCpy(3X)</code></td>
</tr>
<tr>
<td><code>SSAStringZero(3X)</code></td>
<td>See <code>SSAStringCpy(3X)</code></td>
</tr>
<tr>
<td><code>SSASubagentZero(3X)</code></td>
<td>See <code>SSA الجزIsAlive(3X)</code></td>
</tr>
<tr>
<td><code>sscanf(3S)</code></td>
<td>See <code>scanf(3S)</code></td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>start_color(3XC)</code></td>
<td>See <code>can_change_color(3XC)</code></td>
</tr>
<tr>
<td><code>stdio(3S)</code></td>
<td>standard buffered input/output package</td>
</tr>
<tr>
<td><code>step(3G)</code></td>
<td>See <code>regexpr(3G)</code></td>
</tr>
<tr>
<td><code>store(3B)</code></td>
<td>See <code>dbm(3B)</code></td>
</tr>
<tr>
<td><code>str(3G)</code></td>
<td>See <code>strfind(3G)</code></td>
</tr>
<tr>
<td><code>str2sig(3C)</code></td>
<td>translation between signal name and signal number</td>
</tr>
<tr>
<td><code>stradd(3G)</code></td>
<td>See <code>strccpy(3G)</code></td>
</tr>
<tr>
<td><code>stracecmp(3C)</code></td>
<td>See <code>string(3C)</code></td>
</tr>
<tr>
<td><code>strcat(3C)</code></td>
<td>See <code>string(3C)</code></td>
</tr>
<tr>
<td><code>strccpy(3G)</code></td>
<td>copy strings, compressing or expanding escape codes</td>
</tr>
<tr>
<td><code>strchr(3C)</code></td>
<td>See <code>string(3C)</code></td>
</tr>
<tr>
<td><code>strcmp(3C)</code></td>
<td>See <code>string(3C)</code></td>
</tr>
<tr>
<td><code>strcoll(3C)</code></td>
<td>string collation</td>
</tr>
<tr>
<td><code>strcpyp(3C)</code></td>
<td>See <code>string(3C)</code></td>
</tr>
<tr>
<td><code>strcspsn(3C)</code></td>
<td>See <code>string(3C)</code></td>
</tr>
<tr>
<td><code>strdup(3C)</code></td>
<td>See <code>string(3C)</code></td>
</tr>
<tr>
<td><code>streadd(3G)</code></td>
<td>See <code>strccpy(3G)</code></td>
</tr>
<tr>
<td><code>strecpy(3G)</code></td>
<td>See <code>strccpy(3G)</code></td>
</tr>
<tr>
<td><code>strerror(3C)</code></td>
<td>get error message string</td>
</tr>
<tr>
<td><code>strfind(3G)</code></td>
<td>string manipulations</td>
</tr>
<tr>
<td><code>strfmon(3C)</code></td>
<td>convert monetary value to string</td>
</tr>
<tr>
<td><code>strftime(3C)</code></td>
<td>convert date and time to string</td>
</tr>
<tr>
<td><code>string(3C)</code></td>
<td>string operations</td>
</tr>
<tr>
<td><code>string_to_decimal(3)</code></td>
<td>parse characters into decimal record</td>
</tr>
<tr>
<td><code>strlen(3C)</code></td>
<td>See <code>string(3C)</code></td>
</tr>
<tr>
<td><code>strncasecmp(3C)</code></td>
<td>See <code>string(3C)</code></td>
</tr>
<tr>
<td><code>strncat(3C)</code></td>
<td>See <code>string(3C)</code></td>
</tr>
<tr>
<td><code>strncmp(3C)</code></td>
<td>See <code>string(3C)</code></td>
</tr>
<tr>
<td><code>strncpy(3C)</code></td>
<td>See <code>string(3C)</code></td>
</tr>
<tr>
<td><code>strpbrk(3C)</code></td>
<td>See <code>string(3C)</code></td>
</tr>
<tr>
<td><code>strftime(3C)</code></td>
<td>date and time conversion</td>
</tr>
<tr>
<td><code>strrchr(3C)</code></td>
<td>See <code>string(3C)</code></td>
</tr>
<tr>
<td><code>strspn(3G)</code></td>
<td>See <code>strfind(3G)</code></td>
</tr>
<tr>
<td><code>strsignal(3C)</code></td>
<td>get error message string</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>strspn</code> (3C)</td>
<td>See <code>string</code>(3C)</td>
</tr>
<tr>
<td><code>strstr</code> (3C)</td>
<td>See <code>string</code>(3C)</td>
</tr>
<tr>
<td><code>strtod</code> (3C)</td>
<td>convert string to double-precision number</td>
</tr>
<tr>
<td><code>strtok</code> (3C)</td>
<td>See <code>string</code>(3C)</td>
</tr>
<tr>
<td><code>strtok_r</code> (3C)</td>
<td>See <code>string</code>(3C)</td>
</tr>
<tr>
<td><code>strtol</code> (3C)</td>
<td>string conversion routines</td>
</tr>
<tr>
<td><code>strtoll</code> (3C)</td>
<td>See <code>strtol</code> (3C)</td>
</tr>
<tr>
<td><code>strtoul</code> (3C)</td>
<td>convert string to unsigned long</td>
</tr>
<tr>
<td><code>strtoull</code> (3C)</td>
<td>See <code>strtoul</code> (3C)</td>
</tr>
<tr>
<td><code>strtows</code> (3C)</td>
<td>code conversion for Process Code and File Code</td>
</tr>
<tr>
<td><code>strtmns</code> (3G)</td>
<td>See <code>strfind</code>(3G)</td>
</tr>
<tr>
<td><code>strxfrm</code> (3C)</td>
<td>string transformation</td>
</tr>
<tr>
<td><code>subpad</code> (3X)</td>
<td>See <code>curs_pad</code>(3X)</td>
</tr>
<tr>
<td><code>subpad</code> (3XC)</td>
<td>See <code>newpad</code>(3XC)</td>
</tr>
<tr>
<td><code>subwin</code> (3X)</td>
<td>See <code>curs_window</code>(3X)</td>
</tr>
<tr>
<td><code>subwin</code> (3XC)</td>
<td>See <code>derwin</code>(3XC)</td>
</tr>
<tr>
<td><code>svc_auth_reg</code> (3N)</td>
<td>See <code>rpc_svc_reg</code>(3N)</td>
</tr>
<tr>
<td><code>svc_control</code> (3N)</td>
<td>See <code>rpc_svc_create</code>(3N)</td>
</tr>
<tr>
<td><code>svc_create</code> (3N)</td>
<td>See <code>rpc_svc_create</code>(3N)</td>
</tr>
<tr>
<td><code>svc_destroy</code> (3N)</td>
<td>See <code>rpc_svc_create</code>(3N)</td>
</tr>
<tr>
<td><code>svc_dg_create</code> (3N)</td>
<td>See <code>rpc_svc_create</code>(3N)</td>
</tr>
<tr>
<td><code>svc_dg_enablecache</code> (3N)</td>
<td>See <code>rpc_svc_calls</code>(3N)</td>
</tr>
<tr>
<td><code>svc_done</code> (3N)</td>
<td>See <code>rpc_svc_calls</code>(3N)</td>
</tr>
<tr>
<td><code>svc_err_auth</code> (3N)</td>
<td>See <code>rpc_svc_err</code>(3N)</td>
</tr>
<tr>
<td><code>svc_err_decode</code> (3N)</td>
<td>See <code>rpc_svc_err</code>(3N)</td>
</tr>
<tr>
<td><code>svc_err_noproc</code> (3N)</td>
<td>See <code>rpc_svc_err</code>(3N)</td>
</tr>
<tr>
<td><code>svc_err_noprog</code> (3N)</td>
<td>See <code>rpc_svc_err</code>(3N)</td>
</tr>
<tr>
<td><code>svc_err_progvers</code> (3N)</td>
<td>See <code>rpc_svc_err</code>(3N)</td>
</tr>
<tr>
<td><code>svc_err_systemerr</code> (3N)</td>
<td>See <code>rpc_svc_err</code>(3N)</td>
</tr>
<tr>
<td><code>svc_err_weakauth</code> (3N)</td>
<td>See <code>rpc_svc_err</code>(3N)</td>
</tr>
<tr>
<td><code>svc_exit</code> (3N)</td>
<td>See <code>rpc_svc_create</code>(3N)</td>
</tr>
<tr>
<td><code>svcfd_create</code> (3N)</td>
<td>See <code>rpc_soc</code>(3N)</td>
</tr>
<tr>
<td><code>svc_fd_create</code> (3N)</td>
<td>See <code>rpc_svc_create</code>(3N)</td>
</tr>
<tr>
<td><code>svc_fds</code> (3N)</td>
<td>See <code>rpc_soc</code>(3N)</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>svc_fdset(3N)</td>
<td>See rpc_svc_calls(3N)</td>
</tr>
<tr>
<td>svc_freeargs(3N)</td>
<td>See rpc_svc_calls(3N)</td>
</tr>
<tr>
<td>svc_getargs(3N)</td>
<td>See rpc_svc_calls(3N)</td>
</tr>
<tr>
<td>svc_getcaller(3N)</td>
<td>See rpc_soc(3N)</td>
</tr>
<tr>
<td>svc_getreq(3N)</td>
<td>See rpc_soc(3N)</td>
</tr>
<tr>
<td>svc_getreq_common(3N)</td>
<td>See rpc_svc_calls(3N)</td>
</tr>
<tr>
<td>svc_getreq_poll(3N)</td>
<td>See rpc_svc_calls(3N)</td>
</tr>
<tr>
<td>svc_getreqset(3N)</td>
<td>See rpc_svc_calls(3N)</td>
</tr>
<tr>
<td>svc_getrpccaller(3N)</td>
<td>See rpc_svc_calls(3N)</td>
</tr>
<tr>
<td>svc_kerb_reg(3N)</td>
<td>See kerberos_rpc(3N)</td>
</tr>
<tr>
<td>svc_max_pollfd(3N)</td>
<td>See rpc_svc_calls(3N)</td>
</tr>
<tr>
<td>svc_pollfd(3N)</td>
<td>See rpc_svc_calls(3N)</td>
</tr>
<tr>
<td>svcraw_create(3N)</td>
<td>See rpc_soc(3N)</td>
</tr>
<tr>
<td>svc_raw_create(3N)</td>
<td>See rpc_svc_create(3N)</td>
</tr>
<tr>
<td>svc_reg(3N)</td>
<td>See rpc_svc_reg(3N)</td>
</tr>
<tr>
<td>svc_register(3N)</td>
<td>See rpc_soc(3N)</td>
</tr>
<tr>
<td>svc_run(3N)</td>
<td>See rpc_svc_calls(3N)</td>
</tr>
<tr>
<td>svc_sendreply(3N)</td>
<td>See rpc_svc_calls(3N)</td>
</tr>
<tr>
<td>svctcp_create(3N)</td>
<td>See rpc_soc(3N)</td>
</tr>
<tr>
<td>svc_tli_create(3N)</td>
<td>See rpc_svc_create(3N)</td>
</tr>
<tr>
<td>svc_tp_create(3N)</td>
<td>See rpc_svc_create(3N)</td>
</tr>
<tr>
<td>svcudp_bufcreate(3N)</td>
<td>See rpc_soc(3N)</td>
</tr>
<tr>
<td>svcudp_create(3N)</td>
<td>See rpc_soc(3N)</td>
</tr>
<tr>
<td>svc_unreg(3N)</td>
<td>See rpc_svc_reg(3N)</td>
</tr>
<tr>
<td>svc_unregister(3N)</td>
<td>See rpc_soc(3N)</td>
</tr>
<tr>
<td>svc_vc_create(3N)</td>
<td>See rpc_svc_create(3N)</td>
</tr>
<tr>
<td>swab(3C)</td>
<td>swap bytes</td>
</tr>
<tr>
<td>swapcontext(3C)</td>
<td>See makecontext(3C)</td>
</tr>
<tr>
<td>sync_instruction_memory(3C)</td>
<td>make modified instructions executable</td>
</tr>
<tr>
<td>syncok(3X)</td>
<td>See curs_window(3X)</td>
</tr>
<tr>
<td>syncok(3XC)</td>
<td>synchronize window with its parents or children</td>
</tr>
<tr>
<td>syscall(3B)</td>
<td>indirect system call</td>
</tr>
<tr>
<td>sysconf(3C)</td>
<td>get configurable system variables</td>
</tr>
<tr>
<td>syslog(3)</td>
<td>control system log</td>
</tr>
</tbody>
</table>
C Library Functions

- **sysmem(3)**: return physical memory information
- **sys_siglist(3B)**: See **psignal(3B)**
- **system(3S)**: issue a shell command
- **t_accept(3N)**: accept a connection request
- **t_addr2uaddr(3N)**: See **netdir(3N)**
- **t_alloc(3N)**: allocate a library structure
- **tan(3M)**: tangent function
- **tanh(3M)**: hyperbolic tangent function
- **t_bind(3N)**: bind an address to a transport endpoint
- **tcdrain(3)**: wait for transmission of output
- **tcflow(3)**: suspend or restart the transmission or reception of data
- **tcflush(3)**: flush non-transmitted output data, non-read input data or both
- **tcgetattr(3)**: get the parameters associated with the terminal
- **tcgetpgrp(3)**: get foreground process group ID
- **tcgetattr(3)**: get process group ID for session leader for controlling terminal
- **t_close(3N)**: close a transport endpoint
- **t_connect(3N)**: establish a connection with another transport user
- **tc_sendbreak(3)**: send a “break” for a specific duration
- **tcsetattr(3)**: set the parameters associated with the terminal
- **tc_setpgid(3)**: set foreground process group ID
- **tc_getpgid(3)**: set foreground process group ID of terminal
- **tdelete(3C)**: See **tsearch(3C)**
- **td_event_addset(3T)**: See **td_ta_event_addr(3T)**
- **td_event_delset(3T)**: See **td_ta_event_addr(3T)**
- **td_event_emptyset(3T)**: See **td_ta_event_addr(3T)**
- **td_event_fillset(3T)**: See **td_ta_event_addr(3T)**
- **td_event isempty(3T)**: See **td_ta_event_addr(3T)**
- **td_eventismember(3T)**: See **td_ta_event_addr(3T)**
- **td_init(3T)**: performs initialization for libthread_db library of interfaces

Modified 4 Apr 1997

SunOS 5.6

3-83
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>td_log(3T)</td>
<td>placeholder for future logging functionality</td>
</tr>
<tr>
<td>td_sync_get_info(3T)</td>
<td>operations on a synchronization object in libthread_db</td>
</tr>
<tr>
<td>td_sync_setstate(3T)</td>
<td>See td_sync_get_info(3T)</td>
</tr>
<tr>
<td>td_sync_waiters(3T)</td>
<td>See td_sync_get_info(3T)</td>
</tr>
<tr>
<td>td_ta_clear_event(3T)</td>
<td>See td_ta_event_addr(3T)</td>
</tr>
<tr>
<td>td_ta_delete(3T)</td>
<td>See td_ta_new(3T)</td>
</tr>
<tr>
<td>td_ta_enable_stats(3T)</td>
<td>collect target process statistics for libthread_db</td>
</tr>
<tr>
<td>td_ta_event_addr(3T)</td>
<td>thread events in libthread_db</td>
</tr>
<tr>
<td>td_ta_event_getmsg(3T)</td>
<td>See td_ta_event_addr(3T)</td>
</tr>
<tr>
<td>td_ta_get_nthreads(3T)</td>
<td>gets the total number of threads in a process for libthread_db</td>
</tr>
<tr>
<td>td_ta_get_ph(3T)</td>
<td>See td_ta_new(3T)</td>
</tr>
<tr>
<td>td_ta_get_stats(3T)</td>
<td>See td_ta_enable_stats(3T)</td>
</tr>
<tr>
<td>td_ta_map_addr2sync(3T)</td>
<td>get a synchronization object handle from a synchronization object’s address</td>
</tr>
<tr>
<td>td_ta_map_id2thr(3T)</td>
<td>convert a thread id or LWP id to a thread handle</td>
</tr>
<tr>
<td>td_ta_map_lwp2thr(3T)</td>
<td>See td_ta_map_id2thr(3T)</td>
</tr>
<tr>
<td>td_ta_new(3T)</td>
<td>allocate and deallocate process handles for libthread_db</td>
</tr>
<tr>
<td>td_ta_reset_stats(3T)</td>
<td>See td_ta_enable_stats(3T)</td>
</tr>
<tr>
<td>td_ta_setconcurrency(3T)</td>
<td>set concurrency level for target process</td>
</tr>
<tr>
<td>td_ta_set_event(3T)</td>
<td>See td_ta_event_addr(3T)</td>
</tr>
<tr>
<td>td_ta_sync_iter(3T)</td>
<td>iterator functions on process handles from libthread_db library of interfaces</td>
</tr>
<tr>
<td>td_ta_thr_iter(3T)</td>
<td>See td_ta_sync_iter(3T)</td>
</tr>
<tr>
<td>td_ta_tsd_iter(3T)</td>
<td>See td_ta_sync_iter(3T)</td>
</tr>
<tr>
<td>td_thr_clear_event(3T)</td>
<td>See td_ta_event_addr(3T)</td>
</tr>
<tr>
<td>td_thr_dbresume(3T)</td>
<td>See td_thr_dbsuspend(3T)</td>
</tr>
<tr>
<td>td_thr_dbsuspend(3T)</td>
<td>suspend and resume threads in libthread_db</td>
</tr>
<tr>
<td>td_thr_event_enable(3T)</td>
<td>See td_ta_event_addr(3T)</td>
</tr>
<tr>
<td>td_thr_event_getmsg(3T)</td>
<td>See td_ta_event_addr(3T)</td>
</tr>
<tr>
<td>td_thr_getfpregs(3T)</td>
<td>See td_thr_getregs(3T)</td>
</tr>
</tbody>
</table>
td_thr_getregs(3T) reading and writing thread registers in libthread_db

td_thr_get_info(3T) get thread information in libthread_db library of interfaces

td_thr_getxregs(3T) See td_thr_getregs(3T)
td_thr_getxregsize(3T) See td_thr_getregs(3T)
td_thr_lockowner(3T) iterate over the set of locks owned by a thread

td_thr_set_event(3T) See td_ta_event_addr(3T)
td_thr_setfpregs(3T) See td_thr_getregs(3T)
td_thr_setgregs(3T) set the priority of a thread

td_thr_setsigpending(3T) manage thread signals for libthread_db

td_thr_setxregs(3T) See td_thr_getgregs(3T)
td_thr_sigsetmask(3T) See td_thr_setsigpending(3T)
td_thr_sleepinfo(3T) return the synchronization handle for the object on which a thread is blocked

td_thr_lsd(3T) get a thread’s thread-specific data for libthread_db library of interfaces

td_thr_validate(3T) test a thread handle for validity

tell(3C) return a file offset for a file descriptor
telldir(3C) current location of a named directory stream

tempnam(3S)See tmpnam(3S)
termattr(3X) See curs_termattr(3X)
termattr(3XC) return the video attributes supported by the terminal
termios(3) general terminal interface
termname(3X) See curs_termattr(3X)
termname(3XC) return the value of the environmental variable TERM
t_error(3N) produce error message
textdomain(3C) See gettext(3C)
tfind(3C) See tsearch(3C)
t_free(3N) free a library structure
tgetent(3X) See curs_termcap(3X)
tgetent(3XC) emulate the termcap database
tgetflag(3X) See curs_termcap(3X)
tgetflag(3XC) See tgetent(3XC)
t_getinfo(3N) get protocol-specific service information
tgetnum(3X) See curs_termcap(3X)
tgetnum(3XC) See tgetent(3XC)
t_getprotaddr(3N) get the protocol addresses
t_getstate(3N) get the current state
tgetstr(3X) See curs_termcap(3X)
tgetstr(3XC) See tgetent(3XC)
tgoto(3X) See curs_termcap(3X)
tgoto(3XC) See tgetent(3XC)

thr_continue(3T) See thr_suspend(3T)
thr_create(3T) See pthread_create(3T)
threads(3T) thread libraries: libpthread and libthread
thr_exit(3T) See pthread_exit(3T)
thr_getconcurrency(3T) See thr_setconcurrency(3T)
thr_getprio(3T) See pthread_setschedparam(3T)
thr_getspecific(3T) See pthread_key_create(3T)
thr_join(3T) See pthread_join(3T)
thr_keycreate(3T) See pthread_key_create(3T)
thr_kill(3T) See pthread_kill(3T)
thr_main(3T) identify the main thread
thr_min_stack(3T) returns the minimum-allowable size for a thread’s stack
thr_self(3T) See pthread_self(3T)
thr_setconcurrency(3T) get/set thread concurrency level
thr_setprio(3T) See pthread_setschedparam(3T)
thr_setspecific(3T) See pthread_key_create(3T)
thr_sigsetmask(3T) See pthread_sigmask(3T)
thr_stksegment(3T) get thread stack bottom and stack size
thr_suspend(3T) suspend or continue thread execution
thr_yield(3T) thread yield to another thread
tigetflag(3X) See curs_terminfo(3X)
tigetflag(3XC) return the value of a terminfo capability
tigetnum(3X) See curs_terminfo(3X)
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Related Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>tigetnum</code></td>
<td>See <code>tigetflag</code></td>
<td></td>
</tr>
<tr>
<td><code>tigetstr</code></td>
<td>See <code>curs_terminfo</code></td>
<td></td>
</tr>
<tr>
<td><code>timeout</code></td>
<td>See <code>curs_inopts</code></td>
<td></td>
</tr>
<tr>
<td><code>timeout</code></td>
<td>See <code>notimeout</code></td>
<td></td>
</tr>
<tr>
<td><code>timer_create</code></td>
<td>create a timer</td>
<td></td>
</tr>
<tr>
<td><code>timer_delete</code></td>
<td>delete a per-LWP timer</td>
<td></td>
</tr>
<tr>
<td><code>timer_getoverrun</code></td>
<td>See <code>timer_settime</code></td>
<td></td>
</tr>
<tr>
<td><code>timer_gettime</code></td>
<td>See <code>timer_settime</code></td>
<td></td>
</tr>
<tr>
<td><code>timer_settime</code></td>
<td>high-resolution timer operations</td>
<td></td>
</tr>
<tr>
<td><code>times</code></td>
<td>get process times</td>
<td></td>
</tr>
<tr>
<td><code>t_listen</code></td>
<td>listen for a connection indication</td>
<td></td>
</tr>
<tr>
<td><code>t_look</code></td>
<td>look at the current event on a transport endpoint</td>
<td></td>
</tr>
<tr>
<td><code>tmpfile</code></td>
<td>create a temporary file</td>
<td></td>
</tr>
<tr>
<td><code>tmpnam</code></td>
<td>create a name for a temporary file</td>
<td></td>
</tr>
<tr>
<td><code>tmpnam_r</code></td>
<td>See <code>tmpnam</code></td>
<td></td>
</tr>
<tr>
<td><code>tnfctl_buffer_alloc</code></td>
<td>allocate or deallocate a buffer for trace data</td>
<td></td>
</tr>
<tr>
<td><code>tnfctl_buffer_dealloc</code></td>
<td>See <code>tnfctl_buffer_alloc</code></td>
<td></td>
</tr>
<tr>
<td><code>tnfctl_check_libs</code></td>
<td>See <code>tnfctl_indirect_open</code></td>
<td></td>
</tr>
<tr>
<td><code>tnfctl_close</code></td>
<td>close a tnfctl handle</td>
<td></td>
</tr>
<tr>
<td><code>tnfctl_continue</code></td>
<td>See <code>tnfctl_pid_open</code></td>
<td></td>
</tr>
<tr>
<td><code>tnfctl_exec_open</code></td>
<td>See <code>tnfctl_pid_open</code></td>
<td></td>
</tr>
<tr>
<td><code>tnfctl_filter_list_add</code></td>
<td>See <code>tnfctl_trace_state_set</code></td>
<td></td>
</tr>
<tr>
<td><code>tnfctl_filter_list_delete</code></td>
<td>See <code>tnfctl_trace_state_set</code></td>
<td></td>
</tr>
<tr>
<td><code>tnfctl_filter_list_get</code></td>
<td>See <code>tnfctl_trace_state_set</code></td>
<td></td>
</tr>
<tr>
<td><code>tnfctl_filter_state_set</code></td>
<td>See <code>tnfctl_trace_state_set</code></td>
<td></td>
</tr>
<tr>
<td><code>tnfctl_indirect_open</code></td>
<td>control probes of another process where caller provides /proc functionality</td>
<td></td>
</tr>
<tr>
<td><code>tnfctl_internal_open</code></td>
<td>create handle for internal process probe control</td>
<td></td>
</tr>
<tr>
<td><code>tnfctl_kernel_open</code></td>
<td>create handle for kernel probe control</td>
<td></td>
</tr>
<tr>
<td><code>tnfctl_pid_open</code></td>
<td>interfaces for direct probe and process control for another process</td>
<td></td>
</tr>
<tr>
<td><code>tnfctl_probe_apply</code></td>
<td>iterate over probes</td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>tnfctl_probe_apply_ids(3X)</td>
<td>See tnfctl_probe_apply(3X)</td>
<td></td>
</tr>
<tr>
<td>tnfctl_probe_connect(3X)</td>
<td>See tnfctl_probe_connect(3X)</td>
<td></td>
</tr>
<tr>
<td>tnfctl_probe_disable(3X)</td>
<td>See tnfctl_probe_disable(3X)</td>
<td></td>
</tr>
<tr>
<td>tnfctl_probe_disconnect_all(3X)</td>
<td>See tnfctl_probe_disconnect_all(3X)</td>
<td></td>
</tr>
<tr>
<td>tnfctl_probe_enable(3X)</td>
<td>See tnfctl_probe_enable(3X)</td>
<td></td>
</tr>
<tr>
<td>tnfctl_probe_state_get(3X)</td>
<td>interfaces to query and to change the state of a probe</td>
<td></td>
</tr>
<tr>
<td>tnfctl_probe_trace(3X)</td>
<td>See tnfctl_probe_state_get(3X)</td>
<td></td>
</tr>
<tr>
<td>tnfctl_probe_untrace(3X)</td>
<td>See tnfctl_probe_state_get(3X)</td>
<td></td>
</tr>
<tr>
<td>tnfctl_register_funcs(3X)</td>
<td>register callbacks for probe creation and destruction</td>
<td></td>
</tr>
<tr>
<td>tnfctl_strerror(3X)</td>
<td>map a tnfctl error code to a string</td>
<td></td>
</tr>
<tr>
<td>tnfctl_trace_attrs_get(3X)</td>
<td>get the trace attributes from a tnfctl handle</td>
<td></td>
</tr>
<tr>
<td>tnfctl_trace_state_set(3X)</td>
<td>control kernel tracing and process filtering</td>
<td></td>
</tr>
<tr>
<td>TNF_DEBUG(3X)</td>
<td>See TNF_PROBE(3X)</td>
<td></td>
</tr>
<tr>
<td>TNF_DECLARE_RECORD(3X)</td>
<td>TNF type extension interface for probes</td>
<td></td>
</tr>
<tr>
<td>TNF_DEFINE_RECORD_1(3X)</td>
<td>See TNF_DECLARE_RECORD(3X)</td>
<td></td>
</tr>
<tr>
<td>TNF_DEFINE_RECORD_2(3X)</td>
<td>See TNF_DECLARE_RECORD(3X)</td>
<td></td>
</tr>
<tr>
<td>TNF_DEFINE_RECORD_3(3X)</td>
<td>See TNF_DECLARE_RECORD(3X)</td>
<td></td>
</tr>
<tr>
<td>TNF_DEFINE_RECORD_4(3X)</td>
<td>See TNF_DECLARE_RECORD(3X)</td>
<td></td>
</tr>
<tr>
<td>TNF_DEFINE_RECORD_5(3X)</td>
<td>See TNF_DECLARE_RECORD(3X)</td>
<td></td>
</tr>
<tr>
<td>TNF_PROBE_0(3X)</td>
<td>See TNF_PROBE(3X)</td>
<td></td>
</tr>
<tr>
<td>TNF_PROBE_0_DEBUG(3X)</td>
<td>See TNF_PROBE(3X)</td>
<td></td>
</tr>
<tr>
<td>TNF_PROBE_1(3X)</td>
<td>See TNF_PROBE(3X)</td>
<td></td>
</tr>
<tr>
<td>TNF_PROBE_1_DEBUG(3X)</td>
<td>See TNF_PROBE(3X)</td>
<td></td>
</tr>
<tr>
<td>TNF_PROBE_2(3X)</td>
<td>See TNF_PROBE(3X)</td>
<td></td>
</tr>
<tr>
<td>TNF_PROBE_2_DEBUG(3X)</td>
<td>See TNF_PROBE(3X)</td>
<td></td>
</tr>
<tr>
<td>TNF_PROBE_3(3X)</td>
<td>See TNF_PROBE(3X)</td>
<td></td>
</tr>
<tr>
<td>TNF_PROBE_3_DEBUG(3X)</td>
<td>See TNF_PROBE(3X)</td>
<td></td>
</tr>
<tr>
<td>TNF_PROBE_4(3X)</td>
<td>See TNF_PROBE(3X)</td>
<td></td>
</tr>
<tr>
<td>TNF_PROBE_4_DEBUG(3X)</td>
<td>See TNF_PROBE(3X)</td>
<td></td>
</tr>
<tr>
<td>TNF_PROBE_5(3X)</td>
<td>See TNF_PROBE(3X)</td>
<td></td>
</tr>
<tr>
<td>TNF_PROBE_5_DEBUG(3X)</td>
<td>See TNF_PROBE(3X)</td>
<td></td>
</tr>
<tr>
<td>tnf_process_disable(3X)</td>
<td>probe control internal interface</td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>tnf_process_enable(3X)</td>
<td>See tnf_process_disable(3X)</td>
<td></td>
</tr>
<tr>
<td>tnf_thread_disable(3X)</td>
<td>See tnf_process_disable(3X)</td>
<td></td>
</tr>
<tr>
<td>tnf_thread_enable(3X)</td>
<td>See tnf_process_disable(3X)</td>
<td></td>
</tr>
<tr>
<td>toascii(3C)</td>
<td>translate integer to a 7-bit ASCII character</td>
<td></td>
</tr>
<tr>
<td>_tolower(3C)</td>
<td>transliterate upper-case characters to lower-case</td>
<td></td>
</tr>
<tr>
<td>tolower(3C)</td>
<td>transliterate upper-case characters to lower-case</td>
<td></td>
</tr>
<tr>
<td>t_open(3N)</td>
<td>establish a transport endpoint</td>
<td></td>
</tr>
<tr>
<td>top_panel(3X)</td>
<td>See panel_top(3X)</td>
<td></td>
</tr>
<tr>
<td>top_row(3X)</td>
<td>See menu_item_current(3X)</td>
<td></td>
</tr>
<tr>
<td>t_optmgmt(3N)</td>
<td>manage options for a transport endpoint</td>
<td></td>
</tr>
<tr>
<td>touchline(3X)</td>
<td>See curs_touch(3X)</td>
<td></td>
</tr>
<tr>
<td>touchline(3XC)</td>
<td>See is_linetouched(3XC)</td>
<td></td>
</tr>
<tr>
<td>touchlock(3X)</td>
<td>See maillock(3X)</td>
<td></td>
</tr>
<tr>
<td>touchwin(3X)</td>
<td>See curs_touch(3X)</td>
<td></td>
</tr>
<tr>
<td>touchwin(3XC)</td>
<td>See is_linetouched(3XC)</td>
<td></td>
</tr>
<tr>
<td>_toupper(3C)</td>
<td>transliterate lower-case characters to upper-case</td>
<td></td>
</tr>
<tr>
<td>toupper(3C)</td>
<td>transliterate lower-case characters to upper-case</td>
<td></td>
</tr>
<tr>
<td>towctrans(3C)</td>
<td>wide-character mapping</td>
<td></td>
</tr>
<tr>
<td>towlower(3C)</td>
<td>transliterate upper-case wide-character code to lower-case</td>
<td></td>
</tr>
<tr>
<td>towupper(3C)</td>
<td>transliterate lower-case wide-character code to upper-case</td>
<td></td>
</tr>
<tr>
<td>tparm(3X)</td>
<td>See curs_terminfo(3X)</td>
<td></td>
</tr>
<tr>
<td>tparm(3XC)</td>
<td>See tigetflag(3XC)</td>
<td></td>
</tr>
<tr>
<td>tputs(3X)</td>
<td>See curs_termcap(3X)</td>
<td></td>
</tr>
<tr>
<td>tputs(3XC)</td>
<td>See curs_terminfo(3X)</td>
<td></td>
</tr>
<tr>
<td>tputc(3X)</td>
<td>See putc(3XC)</td>
<td></td>
</tr>
<tr>
<td>tracing(3X)</td>
<td>overview of tnf tracing system</td>
<td></td>
</tr>
<tr>
<td>t_rccv(3N)</td>
<td>receive data or expedited data sent over a connection</td>
<td></td>
</tr>
<tr>
<td>t_recvconnect(3N)</td>
<td>receive the confirmation from a connection request</td>
<td></td>
</tr>
<tr>
<td>t_rcvdis(3N)</td>
<td>retrieve information from disconnect</td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>------------------</td>
<td>--------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td><code>t_rcvrel(3N)</code></td>
<td>acknowledge receipt of an orderly release indication</td>
<td></td>
</tr>
<tr>
<td><code>t_rcvudata(3N)</code></td>
<td>receive a data unit</td>
<td></td>
</tr>
<tr>
<td><code>t_rcvuderr(3N)</code></td>
<td>receive a unit data error indication</td>
<td></td>
</tr>
<tr>
<td><code>truncate(3N)</code></td>
<td>set a file to a specified length</td>
<td></td>
</tr>
<tr>
<td><code>tsearch(3C)</code></td>
<td>manage binary search trees</td>
<td></td>
</tr>
<tr>
<td><code>t_snd(3N)</code></td>
<td>send data or expedited data over a connection</td>
<td></td>
</tr>
<tr>
<td><code>t_snddis(3N)</code></td>
<td>send user-initiated disconnection request</td>
<td></td>
</tr>
<tr>
<td><code>t_sndrel(3N)</code></td>
<td>initiate an orderly release</td>
<td></td>
</tr>
<tr>
<td><code>t_sndudata(3N)</code></td>
<td>send a data unit</td>
<td></td>
</tr>
<tr>
<td><code>t_strerror(3N)</code></td>
<td>get error message string</td>
<td></td>
</tr>
<tr>
<td><code>t_sync(3N)</code></td>
<td>synchronize transport library</td>
<td></td>
</tr>
<tr>
<td><code>ttyname(3C)</code></td>
<td>find pathname of a terminal</td>
<td></td>
</tr>
<tr>
<td><code>ttyname_r(3C)</code></td>
<td>See <code>ttyname(3C)</code></td>
<td></td>
</tr>
<tr>
<td><code>ttyslot(3C)</code></td>
<td>find the slot in the utmp file of the current user</td>
<td></td>
</tr>
<tr>
<td><code>t_unbind(3N)</code></td>
<td>disable a transport endpoint</td>
<td></td>
</tr>
<tr>
<td><code>twalk(3C)</code></td>
<td>See <code>tsearch(3C)</code></td>
<td></td>
</tr>
<tr>
<td><code>typeahead(3X)</code></td>
<td>See <code>curs_inopts(3X)</code></td>
<td></td>
</tr>
<tr>
<td><code>typeahead(3XC)</code></td>
<td>check for type-ahead characters</td>
<td></td>
</tr>
<tr>
<td><code>tzset(3C)</code></td>
<td>See <code>ctime(3C)</code></td>
<td></td>
</tr>
<tr>
<td><code>tzsetwall(3C)</code></td>
<td>See <code>ctime(3C)</code></td>
<td></td>
</tr>
<tr>
<td><code>uaddr2taddr(3N)</code></td>
<td>See <code>netdir(3N)</code></td>
<td></td>
</tr>
<tr>
<td><code>ulalarm(3C)</code></td>
<td>schedule signal after interval in microseconds</td>
<td></td>
</tr>
<tr>
<td><code>ulckpwdf(3C)</code></td>
<td>See <code>lckpwdf(3C)</code></td>
<td></td>
</tr>
<tr>
<td><code>ulltostr(3C)</code></td>
<td>See <code>strtol(3C)</code></td>
<td></td>
</tr>
<tr>
<td><code>unctrl(3X)</code></td>
<td>See <code>curs_util(3X)</code></td>
<td></td>
</tr>
<tr>
<td><code>unctrl(3XC)</code></td>
<td>convert character to printable form</td>
<td></td>
</tr>
<tr>
<td><code>ungetc(3S)</code></td>
<td>push character back onto input stream</td>
<td></td>
</tr>
<tr>
<td><code>ungetch(3X)</code></td>
<td>See <code>curs_getch(3X)</code></td>
<td></td>
</tr>
<tr>
<td><code>ungetch(3XC)</code></td>
<td>push character back onto the input queue</td>
<td></td>
</tr>
<tr>
<td><code>ungetwc(3S)</code></td>
<td>push wide-character code back into input stream</td>
<td></td>
</tr>
<tr>
<td><code>ungetwch(3X)</code></td>
<td>See <code>curs_getwch(3X)</code></td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>unget_wch(3XC)</td>
<td>See ungetch(3XC)</td>
<td></td>
</tr>
<tr>
<td>unlockpt(3C)</td>
<td>unlock a pseudo-terminal master/slave pair</td>
<td></td>
</tr>
<tr>
<td>unordered(3C)</td>
<td>See isnan(3C)</td>
<td></td>
</tr>
<tr>
<td>unpost_form(3X)</td>
<td>See form_post(3X)</td>
<td></td>
</tr>
<tr>
<td>unpost_menu(3X)</td>
<td>See menu_post(3X)</td>
<td></td>
</tr>
<tr>
<td>untouched(3X)</td>
<td>See curs_touch(3X)</td>
<td></td>
</tr>
<tr>
<td>untouchedwin(3XC)</td>
<td>See is_linetouched(3XC)</td>
<td></td>
</tr>
<tr>
<td>update_panels(3X)</td>
<td>See panel_update(3X)</td>
<td></td>
</tr>
<tr>
<td>updwtmp(3C)</td>
<td>See getutxent(3C)</td>
<td></td>
</tr>
<tr>
<td>updwtmpx(3C)</td>
<td>See getutxent(3C)</td>
<td></td>
</tr>
<tr>
<td>use_env(3X)</td>
<td>See curs_util(3X)</td>
<td></td>
</tr>
<tr>
<td>use_env(3XC)</td>
<td>set values of lines and columns</td>
<td></td>
</tr>
<tr>
<td>user2netname(3N)</td>
<td>See secure_rpc(3N)</td>
<td></td>
</tr>
<tr>
<td>usleep(3C)</td>
<td>suspend execution for interval in microseconds</td>
<td></td>
</tr>
<tr>
<td>utmpname(3C)</td>
<td>See getutent(3C)</td>
<td></td>
</tr>
<tr>
<td>utmpxname(3C)</td>
<td>See getutxent(3C)</td>
<td></td>
</tr>
<tr>
<td>valloc(3C)</td>
<td>See malloc(3C)</td>
<td></td>
</tr>
<tr>
<td>valloc(3X)</td>
<td>See watchmalloc(3X)</td>
<td></td>
</tr>
<tr>
<td>vfprintf(3B)</td>
<td>See printf(3B)</td>
<td></td>
</tr>
<tr>
<td>vfprintf(3S)</td>
<td>See vprintf(3S)</td>
<td></td>
</tr>
<tr>
<td>vidattr(3X)</td>
<td>See cursTermino(3X)</td>
<td></td>
</tr>
<tr>
<td>vidattr(3XC)</td>
<td>display string with video attributes</td>
<td></td>
</tr>
<tr>
<td>vid_attr(3XC)</td>
<td>See vidattr(3XC)</td>
<td></td>
</tr>
<tr>
<td>vidputs(3X)</td>
<td>See cursTermino(3X)</td>
<td></td>
</tr>
<tr>
<td>vid_puts(3XC)</td>
<td>See vidattr(3XC)</td>
<td></td>
</tr>
<tr>
<td>vidputs(3XC)</td>
<td>See vidattr(3XC)</td>
<td></td>
</tr>
<tr>
<td>vlfmt(3C)</td>
<td>display error message in standard format and pass to logging and monitoring services</td>
<td></td>
</tr>
<tr>
<td>vline(3XC)</td>
<td>See hline(3XC)</td>
<td></td>
</tr>
<tr>
<td>vline_set(3XC)</td>
<td>See hline_set(3XC)</td>
<td></td>
</tr>
<tr>
<td>volmgt_acquire(3X)</td>
<td>reserve removable media device</td>
<td></td>
</tr>
<tr>
<td>volmgt_check(3X)</td>
<td>have Volume Management check for media</td>
<td></td>
</tr>
</tbody>
</table>
volmgt_feature_enabled(3X)  check whether specific Volume Management features are enabled
volmgt_inuse(3X)  check whether or not Volume Management is managing a pathname
volmgt_release(3X)  release removable media device reservation
volmgt_root(3X)  return the Volume Management root directory
volmgt_running(3X)  return whether or not Volume Management is running
volmgt_symdev(3X)  See volmgt_symname(3X)
volmgt_symname(3X)  convert between Volume Management symbolic names, and the devices that correspond to them
vpfmt(3C)  display error message in standard format and pass to logging and monitoring services
vprintf(3B)  See printf(3B)
vprintf(3S)  print formatted output of a variable argument list
vsnprintf(3S)  See vprintf(3S)
vsprintf(3B)  See printf(3B)
vsprintf(3S)  See vprintf(3S)
vsyslog(3)  log message with a varargs argument list
vwprintw(3X)  See curs_printw(3X)
vw_printw(3XC)  See mvprintw(3XC)
vwprintw(3XC)  See mvprintw(3XC)
vwscanw(3X)  See curs_scanw(3X)
vw_scanw(3XC)  See mvscanw(3XC)
vwscanw(3XC)  See mvscanw(3XC)
waddch(3X)  See curs_addch(3X)
waddch(3XC)  See addch(3XC)
waddchstr(3X)  See curs_addchstr(3X)
waddchstr(3XC)  See addchstr(3XC)
waddchstr(3XC)  See curs_addchstr(3X)
waddchstr(3XC)  See addchstr(3XC)
waddnstr(3X)  See curs_addnstr(3X)
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>See Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>waddnstr(3XC)</td>
<td></td>
<td>addnstr(3XC)</td>
</tr>
<tr>
<td>waddnwsstr(3X)</td>
<td></td>
<td>curs_addwsstr(3X)</td>
</tr>
<tr>
<td>waddnwsstr(3XC)</td>
<td></td>
<td>addnwsstr(3XC)</td>
</tr>
<tr>
<td>waddstr(3X)</td>
<td></td>
<td>curs_addstr(3X)</td>
</tr>
<tr>
<td>waddstr(3XC)</td>
<td></td>
<td>addnstr(3XC)</td>
</tr>
<tr>
<td>waddwch(3X)</td>
<td></td>
<td>curs_addwch(3X)</td>
</tr>
<tr>
<td>wadd_wch(3XC)</td>
<td></td>
<td>add_wch(3XC)</td>
</tr>
<tr>
<td>waddwchnstr(3X)</td>
<td></td>
<td>curs_addwchnstr(3X)</td>
</tr>
<tr>
<td>wadd_wchnstr(3XC)</td>
<td></td>
<td>add_wchnstr(3XC)</td>
</tr>
<tr>
<td>waddwchstr(3X)</td>
<td></td>
<td>curs_addwchstr(3X)</td>
</tr>
<tr>
<td>wadd_wchstr(3XC)</td>
<td></td>
<td>add_wchstr(3XC)</td>
</tr>
<tr>
<td>waddwstr(3X)</td>
<td></td>
<td>curs_addwstr(3X)</td>
</tr>
<tr>
<td>waddwstr(3XC)</td>
<td></td>
<td>addnwsstr(3XC)</td>
</tr>
<tr>
<td>wadjcurspos(3X)</td>
<td></td>
<td>curs_alecompat(3X)</td>
</tr>
<tr>
<td>wait(3B)</td>
<td>wait for process to terminate or stop</td>
<td>wait(3B)</td>
</tr>
<tr>
<td>wait3(3B)</td>
<td></td>
<td>wait(3B)</td>
</tr>
<tr>
<td>wait3(3C)</td>
<td></td>
<td>wait(3B)</td>
</tr>
<tr>
<td>wait4(3B)</td>
<td></td>
<td>wait(3B)</td>
</tr>
<tr>
<td>wait4(3C)</td>
<td></td>
<td>wait3(3C)</td>
</tr>
<tr>
<td>waitpid(3B)</td>
<td></td>
<td>wait(3B)</td>
</tr>
<tr>
<td>watof(3C)</td>
<td></td>
<td>wcstod(3C)</td>
</tr>
<tr>
<td>watoi(3C)</td>
<td></td>
<td>wcstol(3C)</td>
</tr>
<tr>
<td>watol(3C)</td>
<td></td>
<td>wcstol(3C)</td>
</tr>
<tr>
<td>watoll(3C)</td>
<td></td>
<td>wcstol(3C)</td>
</tr>
<tr>
<td>wattr_get(3XC)</td>
<td></td>
<td>attr_get(3XC)</td>
</tr>
<tr>
<td>wattroff(3X)</td>
<td></td>
<td>curs_attr(3X)</td>
</tr>
<tr>
<td>wattr_off(3XC)</td>
<td></td>
<td>attr_get(3XC)</td>
</tr>
<tr>
<td>wattroff(3XC)</td>
<td></td>
<td>attr_get(3XC)</td>
</tr>
<tr>
<td>wattron(3X)</td>
<td></td>
<td>curs_attr(3X)</td>
</tr>
<tr>
<td>wattr_on(3XC)</td>
<td></td>
<td>attr_get(3XC)</td>
</tr>
<tr>
<td>wattron(3XC)</td>
<td></td>
<td>curs_attr(3X)</td>
</tr>
<tr>
<td>wattr_set(3XC)</td>
<td></td>
<td>attr_get(3XC)</td>
</tr>
<tr>
<td>wattrset(3X)</td>
<td></td>
<td>attr_get(3XC)</td>
</tr>
<tr>
<td>wattr_set(3XC)</td>
<td></td>
<td>attr_get(3XC)</td>
</tr>
<tr>
<td>wbkgd(3X)</td>
<td></td>
<td>curs_bkgd(3X)</td>
</tr>
</tbody>
</table>

modified 4 Apr 1997    SunOS 5.6    3-93
<table>
<thead>
<tr>
<th>Function</th>
<th>See Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>wbkgd(3XC)</td>
<td>bkgd(3XC)</td>
</tr>
<tr>
<td>wbkgdset(3X)</td>
<td>curs_bkgd(3X)</td>
</tr>
<tr>
<td>wbkgdset(3XC)</td>
<td>bkgd(3XC)</td>
</tr>
<tr>
<td>wbkgnd(3XC)</td>
<td>bkgrnd(3XC)</td>
</tr>
<tr>
<td>wbkgndset(3XC)</td>
<td>bkgrnd(3XC)</td>
</tr>
<tr>
<td>wborder(3X)</td>
<td>curs_border(3X)</td>
</tr>
<tr>
<td>wborder(3XC)</td>
<td>border(3XC)</td>
</tr>
<tr>
<td>wborder_set(3XC)</td>
<td>border_set(3XC)</td>
</tr>
<tr>
<td>wchgt(3XC)</td>
<td>chgt(3XC)</td>
</tr>
<tr>
<td>wclear(3X)</td>
<td>curs_clear(3X)</td>
</tr>
<tr>
<td>wclear(3XC)</td>
<td>clear(3XC)</td>
</tr>
<tr>
<td>wclrtobot(3X)</td>
<td>curs_clear(3X)</td>
</tr>
<tr>
<td>wclrtobot(3XC)</td>
<td>clrtobot(3XC)</td>
</tr>
<tr>
<td>wclretoeol(3X)</td>
<td>curs_clear(3X)</td>
</tr>
<tr>
<td>wclretoeol(3XC)</td>
<td>clrtoeol(3XC)</td>
</tr>
<tr>
<td>wcsetno(3C)</td>
<td>attr_get(3XC)</td>
</tr>
<tr>
<td>wcstrcat(3C)</td>
<td>wcharstring(3C)</td>
</tr>
<tr>
<td>wcstrchr(3C)</td>
<td>wcharstring(3C)</td>
</tr>
<tr>
<td>wcstrcoll(3C)</td>
<td>wcharstring(3C)</td>
</tr>
<tr>
<td>wcstrcoll(3C)</td>
<td>wide character string comparison using collating information</td>
</tr>
<tr>
<td>wcscpy(3C)</td>
<td>wcharstring(3C)</td>
</tr>
<tr>
<td>wcscspn(3C)</td>
<td>wcharstring(3C)</td>
</tr>
<tr>
<td>wcsetno(3C)</td>
<td>cset(3C)</td>
</tr>
<tr>
<td>wcstrftime(3C)</td>
<td>wcharstring(3C)</td>
</tr>
<tr>
<td>wcslen(3C)</td>
<td>wcharstring(3C)</td>
</tr>
<tr>
<td>wcscat(3C)</td>
<td>wcharstring(3C)</td>
</tr>
<tr>
<td>wcscmp(3C)</td>
<td>wcharstring(3C)</td>
</tr>
<tr>
<td>wcscmp(3C)</td>
<td>wcharstring(3C)</td>
</tr>
<tr>
<td>wcscpy(3C)</td>
<td>wcharstring(3C)</td>
</tr>
<tr>
<td>wcscspn(3C)</td>
<td>wcharstring(3C)</td>
</tr>
<tr>
<td>wcsetno(3C)</td>
<td>wcharstring(3C)</td>
</tr>
<tr>
<td>wcstok(3C)</td>
<td>wcharstring(3C)</td>
</tr>
<tr>
<td>wcstod(3C)</td>
<td>wcharstring(3C)</td>
</tr>
<tr>
<td>wcsstrcat(3C)</td>
<td>wcharstring(3C)</td>
</tr>
<tr>
<td>wcstrchr(3C)</td>
<td>wcharstring(3C)</td>
</tr>
<tr>
<td>wcstok(3C)</td>
<td>wcharstring(3C)</td>
</tr>
</tbody>
</table>

3-94 SunOS 5.6 modified 4 Apr 1997
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wcstol(3C)</td>
<td>convert wide character string to long integer</td>
</tr>
<tr>
<td>wcstombs(3C)</td>
<td>convert a wide-character string to a character string</td>
</tr>
<tr>
<td>wcstoul(3C)</td>
<td>convert wide character string to unsigned long</td>
</tr>
<tr>
<td>wcstring(3C)</td>
<td>wide character string operations</td>
</tr>
<tr>
<td>wcswcs(3C)</td>
<td>See wcstring(3C)</td>
</tr>
<tr>
<td>wcswcs(3C)</td>
<td>number of column positions of a wide-character string</td>
</tr>
<tr>
<td>wcswcterm(3C)</td>
<td>wide character string transformation</td>
</tr>
<tr>
<td>wcstrans(3C)</td>
<td>convert a wide-character code to a character</td>
</tr>
<tr>
<td>wctype(3C)</td>
<td>define wide-character mapping</td>
</tr>
<tr>
<td>wcurset(3X)</td>
<td>define character class</td>
</tr>
<tr>
<td>wcursyncup(3X)</td>
<td>See curs_window(3X)</td>
</tr>
<tr>
<td>wcursyncup(3XC)</td>
<td>See syncok(3XC)</td>
</tr>
<tr>
<td>wcwidth(3C)</td>
<td>number of column positions of a wide-character code</td>
</tr>
<tr>
<td>wdelch(3X)</td>
<td>See curs_delch(3X)</td>
</tr>
<tr>
<td>wdelch(3XC)</td>
<td>See delch(3XC)</td>
</tr>
<tr>
<td>wdelete(3X)</td>
<td>See curs_deleteeln(3X)</td>
</tr>
<tr>
<td>wdelete(3XC)</td>
<td>See deleteeln(3XC)</td>
</tr>
<tr>
<td>wechochar(3X)</td>
<td>See curs_addch(3X)</td>
</tr>
<tr>
<td>wechochar(3XC)</td>
<td>See echochar(3XC)</td>
</tr>
<tr>
<td>wechowchar(3X)</td>
<td>See curs_addwch(3X)</td>
</tr>
<tr>
<td>wecho_wchar(3XC)</td>
<td>See echo_wchar(3XC)</td>
</tr>
<tr>
<td>werase(3X)</td>
<td>See curs_clear(3X)</td>
</tr>
<tr>
<td>werase(3XC)</td>
<td>See clear(3XC)</td>
</tr>
<tr>
<td>wgetbkgrnd(3XC)</td>
<td>See bkgrnd(3XC)</td>
</tr>
<tr>
<td>wgetch(3X)</td>
<td>See curs_getch(3X)</td>
</tr>
<tr>
<td>wgetch(3XC)</td>
<td>See getch(3XC)</td>
</tr>
<tr>
<td>wgetnstr(3X)</td>
<td>See curs_getstr(3X)</td>
</tr>
<tr>
<td>wgetnstr(3XC)</td>
<td>See getnstr(3XC)</td>
</tr>
<tr>
<td>wgetnwsr(3X)</td>
<td>See curs_getwstr(3X)</td>
</tr>
<tr>
<td>wgetn_wstr(3XC)</td>
<td>See getn_wstr(3XC)</td>
</tr>
</tbody>
</table>
wgetstr(3X)  See curs_getstr(3X)
wgetstr(3XC)  See getnstr(3XC)
wgetcwh(3X)  See curs_getcwh(3X)
wgetwch(3XC)  See get_wch(3XC)
wgetwstr(3X)  See curs_getwstr(3X)
wget_wstr(3XC)  See getn_wstr(3XC)
whline(3X)  See curs_border(3X)
whline(3XC)  See hline(3XC)
whline_set(3XC)  See hline_set(3XC)
WEXITED(3B)  See wait(3B)
WIFSIGNALED(3B)  See wait(3B)
WIFSTOPPED(3B)  See wait(3B)
winch(3X)  See curs_inch(3X)
winch(3XC)  See inch(3XC)
winchnstr(3X)  See curs_inchstr(3X)
winchnstr(3XC)  See inchnstr(3XC)
winchstr(3X)  See curs_inchstr(3X)
winchstr(3XC)  See inchnstr(3XC)
windex(3C)  See wstring(3C)
winstrn(3X)  See curs_insstr(3X)
winstrn(3XC)  See innstr(3XC)
winnwstr(3X)  See curs_inwstr(3X)
winnwstr(3XC)  See innwstr(3XC)
winsch(3X)  See curs_insch(3X)
winsch(3XC)  See insch(3XC)
winsdelln(3X)  See curs_deleteln(3X)
winsdelln(3XC)  See insdelln(3XC)
winsertln(3X)  See curs_deleteln(3X)
winsertln(3XC)  See insertln(3XC)
winsnstr(3X)  See curs_insstr(3X)
winsnstr(3XC)  See insnstr(3XC)
winsnwstr(3X)  See curs_inswnstr(3X)
wins_nwstr(3XC)  See ins_nwstr(3XC)
winsstr(3X)  See curs_insstr(3X)
winsstr(3XC)  See insnstr(3XC)
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>winstr(3X)</td>
<td>See curs_instr(3X)</td>
</tr>
<tr>
<td>winstr(3XC)</td>
<td>See innstr(3XC)</td>
</tr>
<tr>
<td>winswch(3X)</td>
<td>See curs_inswch(3X)</td>
</tr>
<tr>
<td>wins_wch(3XC)</td>
<td>See ins_wch(3XC)</td>
</tr>
<tr>
<td>winswstr(3X)</td>
<td>See curs_inswstr(3XC)</td>
</tr>
<tr>
<td>wins_wstr(3XC)</td>
<td>See ins_nwstr(3XC)</td>
</tr>
<tr>
<td>winwch(3X)</td>
<td>See curs_inwch(3X)</td>
</tr>
<tr>
<td>win_wch(3XC)</td>
<td>See in_wch(3XC)</td>
</tr>
<tr>
<td>winwchnstr(3X)</td>
<td>See curs_inwchnstr(3XC)</td>
</tr>
<tr>
<td>win_wchnstr(3XC)</td>
<td>See ins_wchnstr(3XC)</td>
</tr>
<tr>
<td>winwchstr(3X)</td>
<td>See curs_inwchstr(3X)</td>
</tr>
<tr>
<td>win_wchstr(3XC)</td>
<td>See ins_wchnstr(3XC)</td>
</tr>
<tr>
<td>winwstr(3X)</td>
<td>See curs_inwstr(3XC)</td>
</tr>
<tr>
<td>winwstr(3XC)</td>
<td>See innwstr(3XC)</td>
</tr>
<tr>
<td>wmove(3X)</td>
<td>See curs_move(3X)</td>
</tr>
<tr>
<td>wmove(3XC)</td>
<td>See move(3XC)</td>
</tr>
<tr>
<td>wmovenextch(3X)</td>
<td>See curs_alecompat(3X)</td>
</tr>
<tr>
<td>wmoveprevch(3X)</td>
<td>See curs_alecompat(3X)</td>
</tr>
<tr>
<td>wnoutrefresh(3X)</td>
<td>See curs_refresh(3X)</td>
</tr>
<tr>
<td>wnoutrefresh(3XC)</td>
<td>See doupdate(3XC)</td>
</tr>
<tr>
<td>wordexp(3C)</td>
<td>perform word expansions</td>
</tr>
<tr>
<td>wordfree(3C)</td>
<td>See wordexp(3C)</td>
</tr>
<tr>
<td>wprintw(3X)</td>
<td>See curs_printw(3X)</td>
</tr>
<tr>
<td>wprintw(3XC)</td>
<td>See mvprintw(3XC)</td>
</tr>
<tr>
<td>wredrawln(3X)</td>
<td>See curs_refresh(3X)</td>
</tr>
<tr>
<td>wredrawln(3XC)</td>
<td>See redrawwin(3XC)</td>
</tr>
<tr>
<td>wrefresh(3X)</td>
<td>See curs_refresh(3X)</td>
</tr>
<tr>
<td>wrefresh(3XC)</td>
<td>See doupdate(3XC)</td>
</tr>
<tr>
<td>wrindex(3C)</td>
<td>See wcharstring(3C)</td>
</tr>
<tr>
<td>write_vtoc(3X)</td>
<td>See read_vtoc(3X)</td>
</tr>
<tr>
<td>wscanw(3X)</td>
<td>See curs_scanw(3X)</td>
</tr>
<tr>
<td>wscanw(3XC)</td>
<td>See mvscanw(3XC)</td>
</tr>
<tr>
<td>wscasecmp(3C)</td>
<td>See wstring(3C)</td>
</tr>
<tr>
<td>wscat(3C)</td>
<td>See wcharstring(3C)</td>
</tr>
<tr>
<td>wschr(3C)</td>
<td>See wcharstring(3C)</td>
</tr>
</tbody>
</table>

modified 4 Apr 1997 SunOS 5.6 3-97
Intro (3) C Library Functions

wscmp(3C)  See wcstring(3C)
wscoll(3C)  See wstring(3C)
wsccoll(3C) See wcsccoll(3C)
wscpy(3C)  See wcstring(3C)
wstrcpy(3C) See wcsstrcpy(3C)
wscrl(3X)  See scrl(3X)
wscrl(3XC) See scrll(3XC)
wscsnp(3C) See wcstring(3C)
wscdup(3C) See wstring(3C)
wsetsccreg(3X) See cursc_tropt(3X)
wsetsccreg(3XC) See clearok(3XC)
wslen(3C)  See wcstring(3C)
wscasecmp(3C) See wcstring(3C)
wscat(3C)  See wcstring(3C)
wncompare(3C) See wcstring(3C)
wsncmp(3C) See wcstring(3C)
wsncpy(3C) See wcstring(3C)
wspbrk(3C)  See wcstring(3C)
wsprintf(3C) formatted output conversion
wsprintf(3C) See wstring(3C)
wsrchr(3C) See wcstring(3C)
wsscanf(3C) formatted input conversion
wsscanf(3C) See wcstring(3C)
wsspn(3C)  See wcstring(3C)
wstandend(3X) See curs_attr(3X)
wstandend(3XC) See standend(3XC)
wstandout(3X) See curs_attr(3X)
wstandout(3XC) See standend(3XC)
wstod(3C)  See wcstod(3C)
wstok(3C)  See wcstring(3C)
wstol(3C)  See wcstol(3C)
wtoString(3C) See strtos(3C)
wcsxfrm(3C) Process Code string operations
wsxfrm(3C) See wcsxfrm(3C)
wsyncdown(3X) See curs_window(3X)
wsyncdown(3XC) See syncok(3XC)
wsyncup(3X) See curs_window(3X)
wsyncup(3XC) See syncok(3XC)
wtimeout(3X) See curs_inopts(3X)
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>wtimeout(3XC)</code></td>
<td>See <code>notimeout(3XC)</code></td>
</tr>
<tr>
<td><code>wtouchln(3X)</code></td>
<td>See <code>curs_touch(3X)</code></td>
</tr>
<tr>
<td><code>wtouchln(3XC)</code></td>
<td>See <code>is_linetouched(3XC)</code></td>
</tr>
<tr>
<td><code>wunctrl(3XC)</code></td>
<td>convert a wide character to printable form</td>
</tr>
<tr>
<td><code>wvline(3X)</code></td>
<td>See <code>curs_border(3X)</code></td>
</tr>
<tr>
<td><code>wvline(3XC)</code></td>
<td>See <code>hline(3XC)</code></td>
</tr>
<tr>
<td><code>wvline_set(3XC)</code></td>
<td>See <code>hline_set(3XC)</code></td>
</tr>
<tr>
<td><code>xdr(3N)</code></td>
<td>library routines for external data representation</td>
</tr>
<tr>
<td><code>xdr_accepted_reply(3N)</code></td>
<td>See <code>rpc_xdr(3N)</code></td>
</tr>
<tr>
<td><code>xdr_admin(3N)</code></td>
<td>library routines for external data representation</td>
</tr>
<tr>
<td><code>xdr_array(3N)</code></td>
<td>See <code>xdr_complex(3N)</code></td>
</tr>
<tr>
<td><code>xdr_authsys_parms(3N)</code></td>
<td>See <code>rpc_xdr(3N)</code></td>
</tr>
<tr>
<td><code>xdr_authunix_parms(3N)</code></td>
<td>See <code>rpc_soc(3N)</code></td>
</tr>
<tr>
<td><code>xdr_bool(3N)</code></td>
<td>See <code>xdr_simple(3N)</code></td>
</tr>
<tr>
<td><code>xdr_bytes(3N)</code></td>
<td>See <code>xdr_complex(3N)</code></td>
</tr>
<tr>
<td><code>xdr_callhdr(3N)</code></td>
<td>See <code>rpc_xdr(3N)</code></td>
</tr>
<tr>
<td><code>xdr_callmsg(3N)</code></td>
<td>See <code>rpc_xdr(3N)</code></td>
</tr>
<tr>
<td><code>xdr_char(3N)</code></td>
<td>See <code>xdr_simple(3N)</code></td>
</tr>
<tr>
<td><code>xdr_complex(3N)</code></td>
<td>library routines for external data representation</td>
</tr>
<tr>
<td><code>xdr_control(3N)</code></td>
<td>See <code>xdr_admin(3N)</code></td>
</tr>
<tr>
<td><code>xdr_create(3N)</code></td>
<td>library routines for external data representation stream creation</td>
</tr>
<tr>
<td><code>xdr_destroy(3N)</code></td>
<td>See <code>xdr_create(3N)</code></td>
</tr>
<tr>
<td><code>xdr_double(3N)</code></td>
<td>See <code>xdr_simple(3N)</code></td>
</tr>
<tr>
<td><code>xdr_enum(3N)</code></td>
<td>See <code>xdr_simple(3N)</code></td>
</tr>
<tr>
<td><code>xdr_float(3N)</code></td>
<td>See <code>xdr_simple(3N)</code></td>
</tr>
<tr>
<td><code>xdr_free(3N)</code></td>
<td>See <code>xdr_simple(3N)</code></td>
</tr>
<tr>
<td><code>xdr_getpos(3N)</code></td>
<td>See <code>xdr_admin(3N)</code></td>
</tr>
<tr>
<td><code>xdr_hyper(3N)</code></td>
<td>See <code>xdr_simple(3N)</code></td>
</tr>
<tr>
<td><code>xdr_inline(3N)</code></td>
<td>See <code>xdr_admin(3N)</code></td>
</tr>
<tr>
<td><code>xdr_int(3N)</code></td>
<td>See <code>xdr_simple(3N)</code></td>
</tr>
<tr>
<td><code>xdr_long(3N)</code></td>
<td>See <code>xdr_simple(3N)</code></td>
</tr>
</tbody>
</table>
xdr_longlong_t(3N)  See xdr_simple(3N)
xdrmem_create(3N)  See xdr_create(3N)
xdr_opaque(3N)  See xdr_complex(3N)
xdr_opaque_auth(3N)  See rpc_xdr(3N)
xdr_pointer(3N)  See xdr_complex(3N)
xdr_quadruple(3N)  See xdr_simple(3N)
xdrrec_create(3N)  See xdr_create(3N)
xdrrec_endofrecord(3N)  See xdr_admin(3N)
xdrrec_eof(3N)  See xdr_admin(3N)
xdrrec_readbytes(3N)  See xdr_admin(3N)
xdrrec_skiprecord(3N)  See xdr_admin(3N)
xdr_reference(3N)  See xdr_complex(3N)
xdr_rejected_reply(3N)  See rpc_xdr(3N)
xdr_replymsg(3N)  See rpc_xdr(3N)
xdr_setpos(3N)  See xdr_admin(3N)
xdr_short(3N)  See xdr_simple(3N)
xdr_simple(3N)  library routines for external data representation
xdr_sizeof(3N)  See xdr_admin(3N)
xdrstdio_create(3N)  See xdr_create(3N)
xdr_string(3N)  See xdr_complex(3N)
xdr_u_char(3N)  See xdr_simple(3N)
xdr_u_hyper(3N)  See xdr_simple(3N)
xdr_u_int(3N)  See xdr_simple(3N)
xdr_u_long(3N)  See xdr_simple(3N)
xdr_u_longlong_t(3N)  See xdr_simple(3N)
xdr_union(3N)  See xdr_complex(3N)
xdr_u_short(3N)  See xdr_simple(3N)
xdr_vector(3N)  See xdr_complex(3N)
xdr_void(3N)  See xdr_simple(3N)
xdr_wrapstring(3N)  See xdr_complex(3N)
xfn(3N)  overview of the XFN interface
xfn_attributes(3N)  an overview of XFN attribute operations
xfn_composite_names(3N)  XFN composite syntax: an overview of the syntax for XFN composite name
xfn_compound_names(3N)  XFN compound syntax: an overview of
C Library Functions

XFN model for compound name parsing
XFN links: an overview of XFN links
descriptions of XFN status codes
See rpc_svc_reg(3N)
See rpc_svc_reg(3N)
Bessel functions of the second kind
See y0(3M)
See y0(3M)
See ypclnt(3N)
See ypclnt(3N)
NIS Version 2 client interface
See ypclnt(3N)
See ypclnt(3N)
See ypclnt(3N)
See ypclnt(3N)
See ypclnt(3N)
See ypclnt(3N)
See ypclnt(3N)
change NIS information

modified 4 Apr 1997
SunOS 5.6
3-101
NAME
a64l, l64a – convert between long integer and base-64 ASCII string

SYNOPSIS
#include <stdlib.h>
long a64l(const char *s);
char *l64a(long l);

DESCRIPTION
These functions 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 char-
acters; each character represents a “digit” in a radix-64 notation.
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.
a64l() takes a pointer to a null-terminated base-64 representation and returns a
 corresponding long value. If the string pointed to by s contains more than six characters,
a64l() will use the first six.
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.
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.

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

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
attributes(5)
NAME abort – terminate the process abnormally

SYNOPSIS
#include <stdlib.h>

void abort(void);

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.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO exit(2), getrlimit(2), kill(2), wait(2), waitpid(2), fclose(3S), raise(3C), signal(3C), attributes(5)

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
#include <stdlib.h>
int abs(int val);
long labs(long lval);
long long llabs(long long llval);

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.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
attributes(5)

NOTES
In 2’s-complement representation, the absolute value of the largest magnitude negative integral value is undefined.
Network Functions

NAME
accept – accept a connection on a socket

SYNOPSIS
cc [ flag ...] file ... -lsocket -lns [ library ...]
#include <sys/types.h>
#include <sys/socket.h>

int accept(int s, struct sockaddr *addr, int *addrlen);

DESCRIPTION
The argument s is a socket that has been created with socket(3N) and bound to an address with bind(3N), and that is listening for connections after a call to listen(3N). The accept() function extracts the first connection on the queue of pending connections, creates a new socket with the properties of s, and allocates a new file descriptor, ns, for the socket. If no pending connections are present on the queue and the socket is not marked as non-blocking, accept() blocks the caller until a connection is present. If the socket is marked as non-blocking and no pending connections are present on the queue, accept() returns an error as described below. The accept() function uses the netconfig(4) file to determine the STREAMS device file name associated with s. This is the device on which the connect indication will be accepted. The accepted socket, ns, is used to read and write data to and from the socket that connected to ns; it is not used to accept more connections. The original socket (s) remains open for accepting further connections.

The argument addr 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 addr parameter is determined by the domain in which the communication occurs.

The argument addrlen is a value-result parameter. Initially, it contains the amount of space pointed to by addr; on return it contains the length in bytes of the address returned.

The accept() function 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().

RETURN VALUES
The accept() function returns −1 on error. If it succeeds, it returns a non-negative integer that is a descriptor for the accepted socket.

ERRORS
accept() will fail if:
EBADF The descriptor is invalid.
EINTR The accept attempt was interrupted by the delivery of a signal.
EMFILE The per-process descriptor table is full.
ENODEV The protocol family and type corresponding to s could not be found in the netconfig file.
ENOMEM There was insufficient user memory available to complete the operation.

modified 16 May 1997

SunOS 5.6

3N-105
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 or the connection has already been released.

EWOULDBLOCK  The socket is marked as non-blocking and no connections are present to be accepted.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  poll(2), bind(3N), connect(3N), listen(3N), select(3C), socket(3N), netconfig(4), attributes(5), socket(5)
NAME
accept – accept a new connection on a socket

SYNOPSIS
cc [ flag ... ] file ... -lxnet [ library ... ]
#include <sys/socket.h>
int accept (int socket, struct sockaddr *address, size_t *address_len);

DESCRIPTION
The accept() function extracts the first connection on the queue of pending connections,
creates a new socket with the same socket type protocol and address family as the
specified socket, and allocates a new file descriptor for that socket.
The function takes the following arguments:

socket     Specifies a socket that was created with socket(3XN), has been bound to
an address with bind(3XN), and has issued a successful call to
listen(3XN).

address    Either a null pointer, or a pointer to a sockaddr structure where the
address of the connecting socket will be returned.

address_len Points to a size_t which on input speciﬁes the length of the supplied
sockaddr structure, and on output speciﬁes the length of the stored
address.

If address is not a null pointer, the address of the peer for the accepted connection is
stored in the sockaddr structure pointed to by address, and the length of this address is
stored in the object pointed to by address_len.
If the actual length of the address is greater than the length of the supplied sockaddr
structure, the stored address will be truncated.
If the protocol permits connections by unbound clients, and the peer is not bound, then
the value stored in the object pointed to by address is unspeciﬁed.
If the listen queue is empty of connection requests and O_NONBLOCK is not set on the
file descriptor for the socket, accept() will block until a connection is present. If the
listen() queue is empty of connection requests and O_NONBLOCK is set on the file
descriptor for the socket, accept() will fail and set errno to EAGAIN.
The accepted socket cannot itself accept more connections. The original socket remains
open and can accept more connections.

RETURN VALUES
Upon successful completion, accept() returns the nonnegative file descriptor of the
accepted socket. Otherwise, −1 is returned and errno is set to indicate the error.

ERRORS
The accept() function will fail if:

EBADF       The socket argument is not a valid file descriptor.
ECONNABORTED A connection has been aborted.
ENOTSOCK    The socket argument does not refer to a socket.

modified 16 May 1997
SunOS 5.6
3XN-107
The socket type of the specified socket does not support accepting connections.

O_NONBLOCK is set for the socket file descriptor and no connections are present to be accepted.

The accept() function was interrupted by a signal that was caught before a valid connection arrived.

The socket is not accepting connections.

OPEN_MAX file descriptors are currently open in the calling process.

The maximum number of file descriptors in the system are already open.

The accept() function may fail if:

There was insufficient memory available to complete the operation.

No buffer space is available.

There was insufficient STREAMS resources available to complete the operation.

A protocol error has occurred; for example, the STREAMS protocol stack has not been initialized.

When a connection is available, select(3C) will indicate that the file descriptor for the socket is ready for reading.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See also: bind(3XN), connect(3XN), listen(3XN), select(3C), socket(3XN), attributes(5) socket(5)
NAME
aclcheck – check the validity of an ACL

SYNOPSIS
#include <sys/acl.h>
int aclcheck(aclent_t *aclbufp, int nentries, int *which);

DESCRIPTION
aclcheck() checks the validity of an ACL pointed to by aclbufp. nentries is the number of entries contained in the buffer. which returns the index of the first entry that is invalid.
The function verifies that an ACL pointed to by aclbufp is valid according to the following rules:
- There must be exactly one group_obj ACL entry.
- There must be exactly one user_obj ACL entry.
- There must be exactly one other_obj ACL entry.
- If there are any group ACL entries, then the group ID in each group ACL entry must be unique.
- If there are any user ACL entries, then the user ID in each user ACL entry must be unique.
- If there are any group or user ACL entries, then there must be exactly one class_obj ACL entry.
- If there are any default ACL entries, then the following apply:
  - There must be exactly one default group_obj ACL entry.
  - There must be exactly one default other_obj ACL entry.
  - There must be exactly one default user_obj ACL entry.
  - If there are any default group entries, then the group ID in each default group ACL entry must be unique.
  - If there are any default user entries, then the user ID in each default user ACL entry must be unique.
  - If there are any default group or user entries, then there must be exactly one default class_obj ACL entry.
- If any of the above rules are violated, then the function fails with errno set to EINVAL.

RETURN VALUES
If the ACL is valid, aclcheck() will return 0. Otherwise errno is set to EINVAL and return code is set to one of the following.
- GRP_ERROR: There is more than one (default) group_obj ACL entry.
- USER_ERROR: There is more than one (default) user_obj ACL entry.
- CLASS_ERROR: There is more than one (default) class_obj ACL entry.
- OTHER_ERROR: There is more than one (default) other_obj ACL entry.
- DUPLICATE_ERROR: Duplicate (default) entries of user or group.
- ENTRY_ERROR: The entry type is invalid.
<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MISS_ERROR</td>
<td>Missing (default) <em>group_obj, user_obj, class_obj,</em> or <em>other_obj</em> entries. which returns -1 in this case.</td>
</tr>
<tr>
<td>MEM_ERROR</td>
<td>The system can’t allocate any memory. which returns -1 in this case.</td>
</tr>
</tbody>
</table>

**SEE ALSO** acl(2), aclsort(3)
NAME
acsor       — sort an ACL

SYNOPSIS
#include <sys/acl.h>
int aclsort(int nentries, int calclass, aclent_t *aclbufp);

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

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

Entries will be in the order USER_OBJ, USER, GROUP_OBJ, GROUP, CLASS_OBJ,
OTHER_OBJ, DEF_USER_OBJ, DEF_USER, DEF_GROUP_OBJ, DEF_GROUP,
DEF_CLASS_OBJ, and DEF_OTHER_OBJ.

Entries of type USER, GROUP, DEF_USER, and DEF_GROUP will sorted in increasing order by id.

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

There is exactly one entry each of type USER_OBJ, GROUP_OBJ, CLASS_OBJ,
and OTHER_OBJ.

There is exactly one entry each of type DEF_USER_OBJ, DEF_GROUP_OBJ,
DEF_CLASS_OBJ, and DEF_OTHER_OBJ if there are any default entries.

Entries of type USER, GROUP, DEF_USER, or DEF_GROUP may not contain duplicate entries. A duplicate entry is one of the same type containing the same numeric id.

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

SEE ALSO
acl(2), aclcheck(3)

modified 27 Oct 1994
<table>
<thead>
<tr>
<th>NAME</th>
<th>acltomode, aclfrommode – convert an ACL to/from permission bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYNOPSIS</td>
<td>#include &lt;sys/types.h&gt;</td>
</tr>
<tr>
<td></td>
<td>#include &lt;sys/acl.h&gt;</td>
</tr>
<tr>
<td></td>
<td>int acltomode(aclent_t *aclbufp, int nentries, mode_t *modep);</td>
</tr>
<tr>
<td></td>
<td>int aclfrommode(aclent_t *aclbufp, int nentries, mode_t *modep);</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>acltomode() converts an ACL pointed to by aclbufp into permission bits. If the USER_OBJ ACL entry, GROUP_OBJ ACL entry, or the OTHER_OBJ ACL entry cannot be found in the ACL buffer, then the function fails with errno set to EINVAL. The USER_OBJ ACL entry permission bits are copied to the file owner class bits in the permission bits buffer. The OTHER_OBJ ACL entry permission bits are copied to the file other class bits in the permission bits buffer. If there is a CLASS_OBJ ACL entry, then the CLASS_OBJ ACL entry permission bits are copied to the file group class bits in the permission bits buffer. Otherwise, the GROUP_OBJ ACL entry permission bits are copied to the file group class bits in the permission bits buffer. aclfrommode() converts permission bits into an ACL pointed to by aclbufp. If the USER_OBJ ACL entry, GROUP_OBJ ACL entry, or the OTHER_OBJ ACL entry cannot be found in the ACL buffer, then the function fails with errno set to EINVAL. The file owner class bits from permission bits buffer are copied to the USER_OBJ ACL entry. The file other class bits from permission bits buffer are copied to the USER_OBJ ACL entry. If there is a CLASS_OBJ ACL entry, then the file group class bits from permission bits buffer are copied to the CLASS_OBJ ACL entry, and the GROUP_OBJ ACL entry is not modified. Otherwise, the file group class bits from permission bits buffer are copied to the GROUP_OBJ ACL entry. nentries is the number of ACL entries in the buffer pointed to by aclbufp.</td>
</tr>
<tr>
<td>RETURN VALUES</td>
<td>Upon successful completion, the function returns 0. Otherwise, a value of -1 is returned and errno is set to indicate the error.</td>
</tr>
<tr>
<td>SEE ALSO</td>
<td>acl(2)</td>
</tr>
</tbody>
</table>

3-112 SunOS 5.6 modified 27 Oct 1994
NAME  acltopbits, aclfrompbits - convert an ACL to/from permission bits

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

int acltopbits(aclent_t *aclbufp, int nentries, mode_t *pbitsp);

int aclfrompbits(aclent_t *aclbufp, int nentries, mode_t *pbitsp);

DESCRIPTION  acltopbits() converts an ACL pointed to by aclbufp into permission bits. If the
USER_OWNER ACL entry, GROUP_OWNER ACL entry, or the OTHER ACL entry cannot
be found in the ACL buffer, then the function fails with errno set to EINVAL.
The USER_OWNER ACL entry permission bits are copied to the file owner class bits in the
permission bits buffer. The OTHER ACL entry permission bits are copied to the file other
class bits in the permission bits buffer. If there is a MASK ACL entry, then the MASK
ACL entry permission bits are copied to the file group class bits in the permission bits
buffer. Otherwise, the GROUP_OWNER ACL entry permission bits are copied to the file
group class bits in the permission bits buffer.

aclfrompbits() converts permission bits into an ACL pointed to by aclbufp. If the
USER_OWNER ACL entry, GROUP_OWNER ACL entry, or the OTHER ACL entry cannot
be found in the ACL buffer, then the function fails with errno set to EINVAL.
The file owner class bits from permission bits buffer are copied to the USER_OWNER ACL
entry. The file other class bits from permission bits buffer are copied to the OTHER ACL
entry. If there is a MASK ACL entry, then the file group class bits from permission bits
buffer are copied to the MASK ACL entry, and the GROUP_OWNER ACL entry is not
modified. Otherwise, the file group class bits from permission bits buffer are copied to
the GROUP_OWNER ACL entry.

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

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

SEE ALSO  acl(2)
NAME  acltotext, aclfromtext – convert an internal representation to/from external representation

SYNOPSIS  
#include <sys/acl.h>
char *acltotext(aclent_t *aclbufp, int aclcnt);
aclent_t *aclfromtext(char *acltextp, int *aclcnt);

DESCRIPTION  
acltotext() converts an internal ACL representation pointed to by aclbufp into an external
ACL representation. The space for the external text string is obtained using malloc(3C).
The caller is responsible for freeing the space when it’s done.

aclfromtext() converts an external ACL representation pointed to by acltextp into an inter-
nal ACL representation. The space for the list of ACL entries is obtained using malloc(3C). The caller is responsible for freeing the space when it’s done. aclcnt is
returned to indicate the number of acl entries found.

An external ACL representation is defined as follows:

<acl_entry>,<acl_entry>...

Each <acl_entry> contains one ACL entry. The external representation of an ACL entry
contains three colon-separated fields. The first field contains the ACL entry tag type. The
entry type keywords are defined as:

user  This ACL entry with no uid specified in the ACL entry id field specifies
the access granted to the owner of the object. Otherwise, this ACL entry
specifies the access granted to a specific user-name or user-id number.

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

other  This ACL entry specifies the access granted to any user or group that
does not match any other ACL entry.

mask  This ACL entry specifies the maximum access granted to user or group
entries.

defaultuser  This ACL entry with no uid specified in the ACL entry id field specifies
the default access granted to the owner of the object. Otherwise, this
ACL entry specifies the default access granted to a specific user-name or
user-id number.

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

defaultother  This ACL entry specifies the default access for other entry.

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

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

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

empty It is used by user, group, other, and mask ACL entry types.

The third field contains the following symbolic discretionary access permissions:

r read permission
w write permission
x execute/search permission
- no access

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

SEE ALSO acl(2), malloc(3C)
NAME
acos – arc cosine function

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

DESCRIPTION
The acos() function computes the principal value of the arc cosine of x. The value of x
should be in the range [-1,1].

RETURN VALUES
Upon successful completion, acos() returns the arc cosine of x, in the range [0,π] radians.
If the value of x is not in the range [-1,1], and is not ±Inf or NaN, either 0.0 or NaN is
returned and errno is set to EDOM.
If x is NaN, NaN is returned. If x is ±Inf, either 0.0 is returned and errno is set to EDOM,
or NaN is returned and errno may be set to EDOM.
For exceptional cases, matherr(3M) tabulates the values to be returned as dictated by
Standards other than XPG4.

ERRORS
The acos() function will fail if:
EDOM The value x is not ±Inf or NaN and is not in the range [-1,1].
The acos() function may fail if:
EDOM The value x is ±Inf.

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

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
cos(3M), isnan(3M), matherr(3M), attributes(5), standards(5)
NAME  acosh, asinh, atanh – inverse hyperbolic functions

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

DESCRIPTION  The acosh(), asinh() and atanh() functions compute the inverse hyperbolic cosine, sine, and tangent of their argument, respectively.

RETURN VALUES  The acosh(), asinh() and atanh() functions return the inverse hyperbolic cosine, sine, and tangent of their argument, respectively.
   The acosh() function returns NaN and sets errno to EDOM when its argument is less than 1.0.
   The atanh() function returns NaN and sets errno to EDOM when its argument has absolute value greater than 1.0.
   The atanh() function returns ±Inf and sets errno to ERANGE when its argument is ±1.0.
   If x is NaN, the asinh(), acosh() and atanh() functions return NaN.
   For exceptional cases, matherr(3M) tabulates the values to be returned as dictated by Standards other than XPG4.

ERRORS  The acosh() function will fail if:
   EDOM  The x argument is less than 1.0.

The atanh() function will fail if:
   EDOM  The x argument has an absolute value greater than 1.0.
   ERANGE  The x argument has an absolute value equal to 1.0

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  cosh(3M), matherr(3M), sinh(3M), tanh(3M), attributes(5), standards(5)
NAME  addch, mvaddch, mvwaddch, waddch – add a character (with rendition) to a window

SYNOPSIS  

#include <curses.h>

int addch(const chtype ch);
int mvaddch(int y, int x, const chtype ch);
int mvwaddch(WINDOW *win, int y, int x, const chtype ch);
int waddch(WINDOW *win, const chtype ch);

ARGUMENTS  

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ch</td>
<td>Is the character/attribute pair to be written to the window.</td>
</tr>
<tr>
<td>y</td>
<td>Is the y (row) coordinate of the character’s position in the window.</td>
</tr>
<tr>
<td>x</td>
<td>Is the x (column) coordinate of the character’s position in the window.</td>
</tr>
<tr>
<td>win</td>
<td>Is a pointer to the window in which the character is to be written.</td>
</tr>
</tbody>
</table>

DESCRIPTION  

The addch() function writes a character to the stdscr window at the current cursor position. The mvaddch() and mvwaddch() functions write the character to the position indicated by the x (column) and y (row) parameters. The mvaddch() function writes the character to the stdscr window, while mvwaddch() writes the character to the window specified by win. The waddch() function is identical to addch(), but writes the character to the window specified by win.

These functions advance the cursor after writing the character. Characters that do not fit on the end of the current line are wrapped to the beginning of the next line unless the current line is the last line of the window and scrolling is disabled. In that situation, characters which extend beyond the end of the line are discarded.

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

RETURN VALUES  

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

ERRORS  

None.

SEE ALSO  

attroff(3XC), bkgdset(3XC), doupdate(3XC), inch(3XC), insch(3XC), nl(3XC), printw(3XC), scrrollok(3XC), scr(3XC), terminfo(4)
NAME
addchstr, addchnstr, mvaddchstr, mvaddchnstr, mvwaddchstr, mvwaddchnstr,
waddchstr, waddchnstr – copy a character string (with renditions) to a window

SYNOPSIS
#include <curses.h>

int addchstr(const chtype *chstr);
int addchnstr(const chtype *chstr, int n);
int mvaddchstr(int y, int x, const chtype *chstr, int n);
int mvaddchnstr(int y, int x, const chtype *chstr);
int mvwaddchstr(WINDOW *win, int y, int x,
                const chtype *chstr, int n);
int mvwaddchnstr(WINDOW *win, int y, int x,
                const chtype *chstr);
int waddchstr(WINDOW *win, const chtype *chstr);
int waddchnstr(WINDOW *win, const chtype *chstr, int n);

ARGUMENTS
chstr Is a pointer to the chtype string to be copied to the window.
n Is the maximum number of characters to be copied from chstr. If n is less than 0, the entire string is written or as much of it as fits on the line.
y Is the y (row) coordinate of the starting position of chstr in the window.
x Is the x (column) coordinate of the starting position of chstr in the window.
win Is a pointer to the window to which the string is to be copied.

DESCRIPTION
The addchstr() function copies the chtype character string to the stdscr window at the current cursor position. The mvaddchstr() and mvwaddchstr() functions copy the character string to the starting position indicated by the x (column) and y (row) parameters (the former to the stdscr window; the latter to window win). The waddchstr() is identical to addchstr(), but writes to the window specified by win.

The addchnstr(), waddchnstr(), mvaddchnstr(), and mvwaddchnstr() functions write n characters to the window, or as many as will fit on the line. If n is less than 0, the entire string is written, or as much of it as fits on the line. The former two functions place the string at the current cursor position; the latter two commands use the position specified by the x and y parameters.

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

RETURN VALUES
On success, these functions return OK. Otherwise, they return ERR.

modified 1 Jun 1996

SunOS 5.6

3XC-119
ERRORS
None.

SEE ALSO
addch(3XC), addnstr(3XC), attroff(3XC)
NAME
addnstr, addstr, mvaddnstr, mvaddstr, mvwaddnstr, mvwaddstr, waddnstr, waddstr – add a multi-byte character string (without rendition) to a window

SYNOPSIS
#include <curses.h>
int addnstr(const char *str, int n);
int addstr(const char *str);
int mvaddnstr(int y, int x, const char *str, int n);
int mvaddstr(int y, int x, const char *str);
int mvwaddnstr(WINDOW *win, int y, int x, const char *str, int n);
int mvwaddstr(WINDOW *win, int y, int x, const char *str);
int waddstr(WINDOW *win, const char *str);
int waddnstr(WINDOW *win, const char *str, int n);

ARGUMENTS
str Is a pointer to the character string that is to be written to the window.
n Is the maximum number of characters to be copied from str. If n is less than 0, the entire string is written or as much of it as fits on the line.
y Is the y (row) coordinate of the starting position of str in the window.
x Is the x (column) coordinate of the starting position of str in the window.
win Is a pointer to the window in which the string is to be written.

DESCRIPTION
The addstr() function writes a null-terminated string of multi-byte characters to the stdscr window at the current cursor position. The waddstr() function performs an identical action, but writes the character to the window specified by win. The mvaddstr() and mvwaddstr() functions write the string to the position indicated by the x (column) and y (row) parameters (the former to the stdscr window; the latter to window win).

The addnstr(), waddnstr(), mvaddnstr(), and mvwaddnstr() functions write at most n characters to the window. If n is less than 0, the entire string is written. All of these functions advance the cursor after writing the string.

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

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

modified 1 Jun 1996
SunOS 5.6
3XC-121
## RETURN VALUES
On success, these functions return **OK**. Otherwise, they return **ERR**.

## ERRORS
None.

## SEE ALSO
`addch(3XC)`, `addchstr(3XC)`, `curses(3XC)`
NAME  addnwstr, addwstr, mvaddnwstr, mvaddwstr, mvwaddnwstr, mvwaddwstr, waddnwstr, waddwstr – add a wide-character string to a window

SYNOPSIS  
#include <curses.h>

int addnwstr(const wchar_t *wstr, int n);
int addwstr(const wchar_t *wstr);
int mvaddnwstr(int y, int x, const wchar_t *wstr, int n);
int mvaddwstr(int y, int x, const wchar_t *wstr);
int mvwaddnwstr(WINDOW *win, int y, int x, const wchar_t *wstr, int n);
int mvwaddwstr(WINDOW *win, int y, int x, const wchar_t *wstr);
int waddnwstr(WINDOW *win, const wchar_t *wstr, int n);
int waddwstr(WINDOW *win, const wchar_t *wstr);

ARGUMENTS  

wstr  Is a pointer to the wide-character string that is to be written to the window.

n  Is the maximum number of characters to be copied from wstr. If n is less than 0, the entire string is written or as much of it as fits on the line.

y  Is the y (row) coordinate of the starting position of wstr in the window.

x  Is the x (column) coordinate of the starting position of wstr in the window.

win  Is a pointer to the window in which the string is to be written.

DESCRIPTION  
The addwstr() function writes a null-terminated wide-character string to the stdscr window at the current cursor position. The waddwstr() function performs an identical action, but writes the string to the window specified by win. The mvaddwstr() and mvwaddwstr() functions write the string to the position indicated by the x (column) and y (row) parameters (the former to the stdscr window; the latter to window win).

The addnwstr(), waddnwstr(), mvaddnwstr(), and mvwaddnwstr() functions write at most n characters to the window. If n is less than 0, the entire string is written. The former two functions place the characters at the current cursor position; the latter two commands use the position specified by the x and y parameters.

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

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

modified 1 Jun 1996  SunOS 5.6  3XC-123
Note that these functions differ from the `add_wchnstr(3XC)` set of functions in that the latter copy the string as is (without combining each character with the foreground and background attributes of the window).

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

**ERRORS**
None.

**SEE ALSO**
`add_wch(3XC)`, `add_wchnstr(3XC)`, `curses(3XC)`
NAME
addsev – define additional severities

SYNOPSIS
int addsev(int int_val, const char *string);

DESCRIPTION
The function addsev() defines additional severities for use in subsequent calls to pfmt() or lfmt(). addsev() associates an integer value int_val in the range [5-255] with a character string. It overwrites any previous string association with int_val and string. If int_val is ORed with the flags passed to subsequent calls pfmt() or lfmt(), string will be used as severity.

Passing a NULL string removes the severity.
Add-on severities are only effective within the applications defining them.

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 runtime. An application must provide the means for displaying locale-specific versions of add-on severities.

EXAMPLE
#define Panic 5
setlabel("APPL");
setcat("my_appl");
addsev(Panic, gettxt(":26", "PANIC"));
/* ...
 lfmt(stderr, MM_SOFT|MM_APPL|PANIC, ":12:Cannot locate database\n");

will display the message to stderr and forward to the logging service:
APPL: PANIC: Cannot locate database

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-safe</td>
</tr>
</tbody>
</table>

SEE ALSO
gettext(3C), lfmt(3C), pfmt(3C), attributes(5)
NAME
addseverity – build a list of severity levels for an application for use with fmtmsg

SYNOPSIS
#include <fmtmsg.h>

int addseverity(int severity, const char *string);

DESCRIPTION
The addseverity() function builds a list of severity levels for an application to be used with the message formatting facility, fmtmsg(). severity is an integer value indicating the seriousness of the condition, and string is a pointer to a string describing the condition (string is not limited to a specific size).

If addseverity() is called with an integer value that has not been previously defined, the function adds that new severity value and print string to the existing set of standard severity levels.

If addseverity() is called with an integer value that has been previously defined, the function redefines that value with the new print string. Previously defined severity levels may be removed by supplying the NULL string. If addseverity() is called with a negative number or an integer value of 0, 1, 2, 3, or 4, the function fails and returns −1. The values 0–4 are reserved for the standard severity levels and cannot be modified. 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 that is 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.

Severity levels may also be defined at run time using the SEV_LEVEL environment variable (see fmtmsg(3C)).

EXAMPLES
When the function addseverity() is used as follows:
addseverity(7,"ALERT")
the following call to fmtmsg():
fmtmsg(MM_PRINT, "UX:cat", 7, "invalid syntax", "refer to manual", "UX:cat:001")
produces:
UX:cat: ALERT: invalid syntax
TO FIX: refer to manual UX:cat:001
RETURN VALUES

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

ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO

fmtmsg(1), fmtmsg(3C), gettext(3C), printf(3S), attributes(5)
NAME  add_wch, mvadd_wch, mvwadd_wch, wadd_wch – add a complex character (with rendition) to a window

SYNOPSIS  #include <curses.h>
            int add_wch(const cchar_t *wch);
            int wadd_wch(WINDOW *win, const cchar_t *wch);
            int mvadd_wch(int y, int x, const cchar_t *wch);
            int mvwadd_wch(WINDOW *win, int y, int x, const cchar_t *wch);

ARGUMENTS  wch  Is the character/attribute pair (rendition) to be written to the window.
           win  Is a pointer to the window in which the character is to be written.
           y    Is the y (row) coordinate of the character’s position in the window.
           x    Is the x (column) coordinate of the character’s position in the window.

DESCRIPTION The add_wch() function writes a complex character to the stdscr window at the current cursor position. The mvadd_wch() and mvwadd_wch() functions write the character to the position indicated by the x (column) and y (row) parameters. The mvadd_wch() function writes the character to the stdscr window, while mvwadd_wch() writes the character to the window specified by win. The wadd_wch() function is identical to add_wch(), but writes the character to the window specified by win. These functions advance the cursor after writing the character.

If wch is a spacing complex character, X/Open Curses replaces any previous character at the specified location with wch (and its rendition). If wch is a non-spacing complex character, X/Open Curses preserves all existing characters at the specified location and adds the non-spacing characters of wch to the spacing complex character. It ignores the rendition associated with wch.

Characters that do not fit on the end of the current line are wrapped to the beginning of the next line unless the current line is the last line of the window and scrolling is disabled. In that situation, X/Open Curses discards characters which extend beyond the end of the line.

When wch is a backspace, carriage return, newline, or tab, X/Open Curses moves the cursor appropriately as described in the curses(3XC) man page. Each tab character moves the cursor to the next tab stop. By default, tab stops occur every eight columns. When wch is a control character other than a backspace, carriage return, newline, or tab, it is written using “x notation, where x is a printable character. When X/Open Curses writes wch to the last character position on a line, it automatically generates a newline. When wch is written to the last character position of a scrolling region and scrollok() is enabled, X/Open Curses scrolls the scrolling region up one line (see clearok(3XC)).
<table>
<thead>
<tr>
<th>RETURN VALUES</th>
<th>On success, these functions return <strong>OK</strong>. Otherwise, they return <strong>ERR</strong>.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERRORS</td>
<td>None.</td>
</tr>
<tr>
<td>SEE ALSO</td>
<td><code>attr_off(3XC)</code>, <code>bkgrndset(3XC)</code>, <code>curses(3XC)</code>, <code>doupdate(3XC)</code>, <code>in_wch(3XC)</code>, <code>ins_wch(3XC)</code>, <code>nl(3XC)</code>, <code>printw(3XC)</code>, <code>scroll(3XC)</code>, <code>scroll_w(3XC)</code>, <code>scroll(3XC)</code>, <code>setscrreg(3XC)</code>, <code>terminfo(4)</code></td>
</tr>
</tbody>
</table>
add_wchnstr(3XC)  X/Open Curses Library Functions

NAME
add_wchnstr, add_wchstr, mvadd_wchnstr, mvadd_wchstr, mvwadd_wchnstr,
mvadd_wchstr, wadd_wchnstr, wadd_wchstr – copy a string of complex characters
(with renditions) to a window

SYNOPSIS
#include <curses.h>

int add_wchnstr(const cchar_t *wchstr, int n);
int add_wchstr(const cchar_t *wchstr);
int mvadd_wchnstr(int y, int x, const cchar_t *wchstr,
       int n);
int mvadd_wchstr(int y, int x, const cchar_t *wchstr);
int mvwadd_wchnstr(WINDOW *win, int y, int x,
       const cchar_t *wchstr, int n);
int mvwadd_wchstr(WINDOW *win, int y, int x,
       const cchar_t *wchstr);
int mvwadd_wchnstr(WINDOW *win, int y, int x,
       const cchar_t *wchstr, int n);
int wadd_wchstr(WINDOW *win, const cchar_t *wchstr);
int wadd_wchnstr(WINDOW *win, const cchar_t *wchstr,
       int n);

ARGUMENTS
wchstr  Is a pointer to the cchar_t string to be copied to the window.
n  Is the maximum number of characters to be copied from wchstr. If n is less than
   0, the entire string is written or as much of it as fits on the line.
y  Is the y (row) coordinate of the starting position of wchstr in the window.
x  Is the x (column) coordinate of the starting position of wchstr in the window.
win  Is a pointer to the window to which the string is to be copied.

DESCRIPTION
The add_wchstr() function copies the string of cchar_t characters to the stdscr window at
the current cursor position. The mvadd_wchstr() and mvwadd_wchstr() functions copy
the string to the starting position indicated by the x (column) and y (row) parameters (the
former to the stdscr window; the latter to window win). The wadd_wchstr() is identical
to add_wchstr(), but writes to the window specified by win.

The add_wchnstr(), wadd_wchnstr(), mvadd_wchnstr(), and mvwadd_wchnstr() functions
write n characters to the window, or as many as will fit on the line. If n is less than
0, the entire string is written, or as much of it as fits on the line. The former two functions
place the string at the current cursor position; the latter two commands use the position
specified by the x and y parameters.

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

3XC-130            SunOS 5.6             modified 1 Jun 1996
### RETURN VALUES
On success, these functions return **OK**. Otherwise, they return **ERR**.

### ERRORS
None.

### SEE ALSO
- `addnwstr(3XC)`, `add_wch(3XC)`, `attr_off(3XC)`
NAME  
aiocancel – cancel an asynchronous operation

SYNOPSIS  
cc [ flag ...] file ... -laio [ library ...]
#include <sys/asynch.h>
int aiocancel(aio_result_t *resultp);

DESCRIPTION  
aiocancel() cancels the asynchronous operation associated with the result buffer pointed to by resultp. 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. 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.

RETURN VALUES  
Upon successful completion, aiocancel() returns 0. Upon failure, aiocancel() returns -1 and sets errno to indicate the error.

ERRORS  
aiocancel() will fail if any of the following are true:

EACCES  
The parameter resultp does not correspond to any outstanding asynchronous operation, although there is at least one currently outstanding.

EFAULT  
resultp points to an address outside the address space of the requesting process. See NOTES.

EINVAL  
There are not any outstanding requests to cancel.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  
aioread(3), aiowait(3), attributes(5)

NOTES  
Passing an illegal address as resultp will result in setting errno to EFAULT only if it is detected by the application process.
NAME  aio_cancel – cancel asynchronous I/O request

SYNOPSIS  cc [ flag ... ] file ... -lpthread [ library ... ]
#include <aio.h>
int aio_cancel(int fildes, struct aiocb *aiocbp);

DESCRIPTION  The aio_cancel() function attempts to cancel either one or all outstanding asynchronous I/O requests pending on the file descriptor specified by fildes. If aiocbp is NULL, then all such outstanding cancelable requests are canceled; otherwise, the individual request referenced by aiocbp references will be canceled.

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.

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 */
};

RETURN VALUES  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.

ERRORS  EBADF  fildes is not a valid file descriptor.
ENOSYS  The aio_cancel() function is not supported.

modified 30 Dec 1996  SunOS 5.6  3R-133
**USAGE**
The `aio_cancel()` function has an explicit 64-bit equivalent. See `interface64(5)`.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**SEE ALSO**
`aio_read(3R), aio_return(3R), attributes(5), interface64(5), standards(5)`

**NOTES**
Applications compiled under Solaris 2.3 and 2.4 and using POSIX (see `standards(5)`) Asynchronous Input and Output option must be recompiled to work correctly when Solaris supports this option.

**BUGS**
In Solaris 2.5, these functions always return −1 and set `errno ENOSYS`, because this release does not support the Asynchronous Input and Output option. Beginning with Solaris 2.6, these interfaces are supported.
NAME  aio_fsync – asynchronous file synchronization

SYNOPSIS  cc [ flag . . . file . . . -lposix4 [ library . . . ]
#include <aio.h>
int aio_fsync(int op, aiocb *aiocbp);

DESCRIPTION  The aio_fsync() function queues an asynchronous fsync(3C) or fdatasync(3R) request for all the currently queued I/O operations on the file referenced by aiocbp->aio_fildes, and returns control immediately. This request is serviced concurrently with other activity of the process. If op is O_DSYNC, all I/O operations are completed by a call to fdatasync(3R) (synchronized I/O data integrity completion). If op 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.)

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. The aio_return(3R) and aio_error(3R) functions may be used with this aiocbp value to monitor both the return and the error status of the asynchronous operation while it is proceeding.

aiocbp->aio_sigevent defines the signal to be generated upon I/O completion. If aiocbp->aio_sigevent.sigev_signo is non-zero, then a signal will be generated when all I/O operations have achieved synchronized I/O completion.

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 */
    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 */
};

modified 30 Dec 1996  SunOS 5.6  3R-135
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
The `aio_fsync()` function will fail if:

- **EAGAIN** The requested asynchronous operation was not queued due to temporary resource limitations.
- **EBADF** `aiocbp->aio_fildes` is not a valid file descriptor open for writing.
- **EINVAL** Synchronized I/O is not supported for this file, or a value of `op` other than `O_DSYNC` or `O_SYNC` was specified.
- **ENOSYS** `aio_fsync()` is not supported by this implementation.

USAGE
The `aio_fsync()` function has an explicit 64-bit equivalent. See `interface64(5)`.

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

```
ATTRIBUTE TYPE    ATTRIBUTE VALUE
MT-Level          MT-Safe
```

SEE ALSO
`fcntl(2)`, `open(2)`, `read(2)`, `write(2)`, `aio_error(3R)`, `aio_return(3R)`, `fdatasync(3R)`, `fsync(3C)`, attributes(5), `fcntl(5)`, `interface64(5)`, `standards(5)`

NOTES
If `aio_fsync()` fails, outstanding I/O operations are not guaranteed to have been completed.

Applications compiled under Solaris 2.3 and 2.4 and using POSIX (see `standards(5)`)
Asynchronous Input and Output option must be recompiled to work correctly when Solaris supports this option.

BUGS
In Solaris 2.5, 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  aioread, aiowrite, aioread64, aiowrite64 – read or write asynchronous I/O operations

SYNOPSIS  

Use the following for 32-bit offset types:

```
#include <sys/types.h>
#include <sys/asynch.h>

int aioread(int fildes, char *bufp, int bufs, off_t offset, int whence,
             aio_result_t *resultp);

int aiowrite(int fildes, const char *bufp, int bufs, off_t offset, int whence,
              aio_result_t *resultp);
```

Use the following for 64-bit offset types:

```
#include <sys/types.h>
#include <sys/asynch.h>

int aioread64(int fildes, char *bufp, int bufs, off64_t offset, int whence,
              aio_result_t *resultp);

int aiowrite64(int fildes, const char *bufp, int bufs, off64_t offset, int whence,
               aio_result_t *resultp);
```

DESCRIPTION  

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 bufs bytes of data from the object referenced by the descriptor fildes into the buffer pointed to by bufp.

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 bufs bytes of data from the buffer pointed to by bufp to the object referenced by the descriptor fildes.

On objects capable of seeking, the I/O operation starts at the position specified by whence and offset. 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 whence and offset 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 whence and offset parameters.

aioread64() and aiowrite64() have the same functionality as aioread() and aiowrite() with the added enhancement of 64-bit offset values.

The result of the asynchronous operation is stored in the structure pointed to by resultp:

```
int aio_return;  /* return value of read() or write() */
int aio_errno;  /* value of errno for read() or write() */
```
Upon completion of the operation both `aio_return` and `aio_errno` are set to reflect the result of the operation. `AIO_INPROGRESS` is not a value used by the system so the client may detect a change in state by initializing `aio_return` to this value.

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

Notification of the completion of an asynchronous I/O operation may be obtained synchronously through the `aiowait` 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` returns nothing (using a polling timeout) before returning from the signal handler, no asynchronous I/O notifications are lost. The `aiowait` function is the only way to dequeue an asynchronous notification. Note: `SIGIO` may have several meanings simultaneously: for example, that a descriptor generated `SIGIO` and an asynchronous operation completed. Further, issuing an asynchronous request successfully guarantees that space exists to queue the completion notification.

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

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

### RETURN VALUES

Upon successful completion, `aioread()`, `aiowrite()`, `aioread64()`, and `aiowrite64()` return 0. Upon failure, `aioread()`, `aiowrite()`, `aioread64()`, and `aiowrite64()` return −1 and set `errno` to indicate the error.

### ERRORS

`aioread()`, `aiowrite()`, `aioread64()`, and `aiowrite64()` will fail if any of the following are true:

- **EAGAIN**: The number of asynchronous requests that the system can handle at any one time has been exceeded.
- **EBADF**: `fd` 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.
- **EINVAL**: The parameter `resultp` is currently being used by an outstanding asynchronous request.
- **EINVAL**: `offset` is not a valid offset for this file system type.
- **ENOMEM**: Memory resources are unavailable to initiate request.

---

3-138 SunOS 5.6 modified 10 Mar 1997
ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

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

NOTES
Passing an illegal address to bufp will result in setting errno to EFAULT only if it is detected by the application process.
NAME aio_read, aio_write – asynchronous read and write operations

SYNOPSIS

```
cc [ flag ... ] file ... -lposix4 [ library ... ]
#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 */
};
```

DESCRIPTION

The `aio_read()` function 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.

Upon enqueuing the request, the calling process reads `aiocbp->nbytes` from the file referred to by `aiocbp->fildes` into the buffer pointed to by `aiocbp->aio_buf`. `aiocbp->offset` marks the absolute position from the beginning of the file (in bytes) at which the read begins.

The `aio_write()` function 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.

Upon enqueuing the request, the calling process writes `aiocbp->nbytes` from the buffer pointed to by `aiocbp->aio_buf` into the file referred to by `aiocbp->fildes`. If `O_APPEND` is set for `aiocbp->fildes`, `aio_write()` operations append to the file in the same order as the calls were made.

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.

For regular files, no data transfer will occur past the offset maximum established in the open file description associated with aiocbp->fildes.

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
The aio_read() and aio_write() function will fail if:

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

ENOSYS The aio_read() or aio_write() functions are not supported.

EBADF If the calling function is aio_read(), and aiocbp->fildes is not a valid file descriptor open for reading. If the calling function is aio_write(), and aiocbp->fildes is not a valid file descriptor open for writing.

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

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 aiocbp->aio_offset would be invalid.

The following are additional conditions which may be detected synchronously or asynchronously:

aio_read() OVERFLOW The file is a regular file, aiocbp->aio_nbytes is greater than 0 and the starting offset in aiocbp->aio_offset is before the end-of-file and is at or beyond the offset maximum in the open file description associated with aiocbp->fildes.

aio_write() EFBIG The file is a regular file, aiocbp->aio_nbytes is greater than 0 and the starting offset in aiocbp->aio_offset is at or beyond the offset maximum in the open file description associated with aiocbp->fildes.
The `aio_read()` and `aio_write()` functions have explicit 64-bit equivalents. See `interface64(5)`.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
<tr>
<td><code>close(2)</code>, <code>exec(2)</code>, <code>exit(2)</code>, <code>fork(2)</code>, <code>lseek(2)</code>, <code>read(2)</code>, <code>write(2)</code>, <code>aio_cancel(3R)</code>, <code>aio_return(3R)</code>, <code>lio_listio(3R)</code>, <code>attributes(5)</code>, <code>interface64(5)</code>, <code>siginfo(5)</code>, <code>standards(5)</code></td>
<td></td>
</tr>
</tbody>
</table>

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

Applications compiled under Solaris 2.3 and 2.4 and using POSIX (see `standards(5)`)
Asynchronous Input and Output option must be recompiled to work correctly when Solaris supports this option.

In Solaris 2.5, these functions always return –1 and set `errno` to `ENOSYS`, because this release does not support the Asynchronous Input and Output option. Beginning with Solaris 2.6, these interfaces are supported.
NAME  
aio_return, aio_error – retrieve return or error status of asynchronous I/O operation

SYNOPSIS  
c
cc [ flag . . . ] file . . . -lposix4 [ library . . . ]
#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 */
};

DESCRIPTION  
The aio_return() function returns the return status of the asynchronous I/O request associated with the aiocb structure pointed to by aiocbp.

aio_error() returns the error status of the asynchronous I/O request associated with the aiocb structure pointed to by aiocbp.

The aio_return() function 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.

RETURN VALUES  
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).

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.

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

modified 30 Dec 1996  SunOS 5.6  3R-143
The `aio_return()` and `aio_error()` functions will fail if:

- **EINVAL** `aiocbp` does not reference an asynchronous operation which has completed or failed.
- **ENOSYS** The `aio_return()` or `aio_error()` function is not supported.

The `aio_return()` and `aio_error()` functions have explicit 64-bit equivalents. See `interface64(5)`.

```c
#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 = &my_aiocb;
...
(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 */
      ...
    }
  }
}
```
ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Async-Signal-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO

close(2), exec(2), exit(2), fork(2), lseek(2), read(2), write(2), aio_cancel(3R),
aio_fsync(3R), aio_read(3R), fsync(3C), lio_listio(3R), attributes(5), interface64(5), standards(5)

NOTES

Applications compiled under Solaris 2.3 and 2.4 and using POSIX (see standards(5))
Asynchronous Input and Output option must be recompiled to work correctly when
Solaris supports this option.

BUGS

In Solaris 2.5, these functions always return -1 and set errno to ENOSYS, because this
release does not support the Asynchronous Input and Output option. Beginning with
Solaris 2.6, these interfaces are supported.
NAME  aio_suspend – wait for asynchronous I/O request

SYNOPSIS  cc [ flag . . . ] file . . . -lpthread -lposix4 [ library . . . ]

#include <aio.h>

int aio_suspend(const struct aiocb ** const list[], int nent,
                const struct timespec *timeout);

DESCRIPTION  The aio_suspend() function suspends the caller until at least one of the asynchronous
I/O operations referenced by list has completed, until a signal interrupts the function, or,
if timeout is not NULL, until the time interval specified by timeout has passed. If any of the
aiocb structures in the list corresponds to a completed asynchronous I/O operation (that
is, the error status for the operation is not equal to EINPROGRESS), at the time of the call,
the function returns without suspending the caller.

If the time interval indicated in the timespec structure pointed to by timeout passes before
any of the I/O operations referenced by list are completed, then aio_suspend() returns
with an error.

The list argument is an array of pointers to asynchronous I/O control blocks. The nent
argument 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.

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 */
};
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 com-
pleted, it returns 0. Otherwise, it returns −1, and sets `errno` to indicate the error condi-
tion.

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
The `aio_suspend()` function will fail if:

- **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** The `aio_suspend()` function is not supported.

USAGE
The `aio_suspend()` function has an explicit 64-bit equivalent. See `interface64(5)`.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Async-Signal-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO `aio_fsync(3R)`, `aio_read(3R)`, `aio_return(3R)`, `aio_write(3R)`,
`lio_listio(3R)`, `attributes(5)`, `interface64(5)`, `standards(5)`

NOTES
Applications compiled under Solaris 2.3 and 2.4 and using POSIX (see `standards(5)`)
Asynchronous Input and Output option must be recompiled to work correctly when
Solaris supports this option.

BUGS
In Solaris 2.5, these functions always return −1 and set `errno` to `ENOSYS`, because this
release does not support the Asynchronous Input and Output option. Beginning with
Solaris 2.6, these interfaces are supported.
NAME
aiowait — wait for completion of asynchronous I/O operation

SYNOPSIS
cc [ flag ... ] file ... -laio [ library ... ]
#include <sys/asynch.h>
#include <sys/time.h>
aio_result_t *aiowait(const struct timeval *timeout);

DESCRIPTION
aiowait() suspends the calling process until one of its outstanding asynchronous I/O
operations completes. This provides a synchronous method of notification.
If timeout is a non-zero pointer, it specifies a maximum interval to wait for the completion
of an asynchronous I/O operation. If timeout is a zero pointer, then aiowait() blocks
indefinitely. To effect a poll, the timeout parameter should be non-zero, pointing to a
zero-valued timeval structure.
The timeval structure is defined in <sys/time.h> and contains the following members:

long tv_sec; /* seconds */
long tv_usec; /* and microseconds */

RETURN VALUES
Upon successful completion, aiowait() returns a pointer to the result structure used
when the completed asynchronous I/O operation was requested. Upon failure,
aiowait() returns −1 and sets errno to indicate the error. aiowait() returns 0 if the time
limit expires.

ERRORS
aiowait() will fail if any of the following are true:
EFAULT timeout points to an address outside the address space of the requesting pro-
cess. See NOTES.
EINVAL aiowait() was interrupted by a signal.
EINVAL There are no outstanding asynchronous I/O requests.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

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

NOTES
aiowait() is the only way to dequeue an asynchronous notification. It may be used either
inside a SIGIO signal handler or in the main program. One SIGIO signal may represent
several queued events.
Passing an illegal address as timeout will result in setting errno to EFAULT only if it is
detected by the application process.
NAME  asin – arc sine function

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

DESCRIPTION  The asin() function computes the principal value of the arc sine of x. The value of x should be in the range \([-1,1]\).

RETURN VALUES  Upon successful completion, asin() returns the arc sine of x, in the range \([-\pi/2,\pi/2]\] radians. If the value of x is not in the range \([-1,1]\) and is not \(\pm\text{Inf} or NaN\), either 0.0 or NaN is returned and errno is set to EDOM. If x is NaN, NaN is returned. If x is \(\pm\text{Inf}\), either 0.0 is returned and errno is set to EDOM or NaN is returned and errno may be set to EDOM.

For exceptional cases, matherr(3M) tabulates the values to be returned as dictated by Standards other than XPG4.

ERRORS  The asin() function will fail if:

EDOM The value x is not \(\pm\text{Inf} or NaN\) and is not in the range \([-1,1]\).

The asin() function may fail if:

EDOM The value of x is \(\pm\text{Inf}\).

USAGE  An application wishing to check for error situations should set errno to 0, then call asin(). If errno is non-zero on return, or the return value is NaN, an error has occurred.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  isnan(3M), matherr(3M), sin(3M), attributes(5), standards(5)
NAME   assert – verify program assertion

SYNOPSIS  
#include <assert.h>  
void assert(int expression);

DESCRIPTION  
This macro is useful for putting diagnostics into programs. When it is executed, if expression is false (zero), assert() prints

Assertion failed: expression, file xyz, line nnn

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

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.

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

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

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

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).

Since assert() is implemented as a macro, the expression may not contain any string literals.
NAME
atan2 – arc tangent function

SYNOPSIS
cc [ flag ...] file ... -lm [ library ...]
#include <math.h>
double atan2(double y, double x);

DESCRIPTION
The atan2() function computes the principal value of the arc tangent of y/x, using the
signs of both arguments to determine the quadrant of the return value.

RETURN VALUES
Upon successful completion, atan2() returns the arc tangent of y/x in the range [-π,π]
radians. If both arguments are 0.0, 0.0 is returned and errno may be set to EDOM.
If x or y is NaN, NaN is returned.
In IEEE 754 mode (the -xlibmieee cc compilation option), atan2() handles the following

atan2(±0, x) returns ±0 for x > 0 or x = +0;
atan2(±0, x) returns ±π for x < 0 or x = −0;
atan2(y, ±0) returns π/2 for y > 0;
atan2(y, ±0) returns −π/2 for y < 0;
atan2(±y, Inf) returns ±0 for finite y > 0;
atan2(±Inf, x) returns ±π/2 for finite x;
atan2(±y, −Inf) returns ±π for finite y > 0;
atan2(±Inf, Inf) returns ±π/4;
atan2(±Inf, −Inf) returns ±3π/4.

For exceptional cases, matherr(3M) tabulates the values to be returned as dictated by
Standards other than XPG4.

ERRORS
The atan2() function may fail if:
EDOM Both arguments are 0.0.

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

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

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

modified 29 Dec 1996

SunOS 5.6
3M-151
NAME    atan – arc tangent function

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

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

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

ERRORS No errors will occur.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO atan2(3M), isnan(3M), tan(3M), attributes(5)
NAME
atexit – add program termination routine

SYNOPSIS
#include <stdlib.h>
int atexit(void (*func)(void));

DESCRIPTION
atexit() adds the function func() 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.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
exit(3C), attributes(5)
NAME
attr_get, attr_off, attr_on, attr_set, color_set, wattr_get, wattr_off, wattr_on, wattr_set, wcolor_set – control window attributes

SYNOPSIS
#include <curses.h>

int attr_get(attr_t *attrs, short *color, void *opts);
int attr_off(attr_t attrs, void *opts);
int attr_on(attr_t attrs, void *opts);
int attr_set(attr_t attrs, short color, void *opts);
int color_set(short *color, void *opts);
int wattr_get(WINDOW *win, attr_t attrs, short *color, void *opts);
int wattr_off(WINDOW *win, attr_t attrs, void *opts);
int wattr_on(WINDOW *win, attr_t attrs, void *opts);
int wattr_set(WINDOW *win, attr_t attrs, short color, void *opts);
int wcolor_set(WINDOW *win, short color, void *opts);

ARGUMENTS
attrs Is a pointer to the foreground window attributes to be set or unset.
color Is a pointer to a color pair number.
opts Is reserved for future use.
win Is a pointer to the window in which attribute changes are to be made.

DESCRIPTION
The attr_get() function retrieves the current rendition of stdscr. The wattr_get() function retrieves the current rendition of window win. If attrs or color is a null pointer, no information is retrieved.

The attr_off() and attr_on() functions unset and set, respectively, the specified window attributes of stdscr. These functions only affect the attributes specified; attributes that existed before the call are retained.

The wattr_off() and wattr_on() functions unset or set the specified attributes for window win.

The attr_set() and wattr_set() functions change the rendition of stdscr and win; the old values are not retained.

The color_set() and wcolor_set() functions set the window color of stdscr and win to color.

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

RETURN VALUES
These functions always return OK.
None.

SEE ALSO
add_wch(3XC), addnwstr(3XC),attroff(3XC),bkgrndset(3XC),curses(3XC),init_color(3XC),start_color(3XC)
NAME
attroff, attron, attrset, wattroff, wattron, wattrset – change foreground window attributes

SYNOPSIS
#include <curses.h>
int attroff(int attrs);
int attron(int attrs);
int attrset(int attrs);
int wattroff(WINDOW *win, int attrs);
int wattron(WINDOW *win, int attrs);
int wattrset(WINDOW *win, int attrs);

ARGUMENTS
attrs are the foreground window attributes to be set or unset.
win Is a pointer to the window in which attribute changes are to be made.

DESCRIPTION
The attroff() and attron() functions unset and set, respectively, the specified window attributes of stdscr. These functions only affect the attributes specified; attributes that existed before the call are retained. The wattroff() and wattron() functions unset or set the specified attributes for window win.

The attrset() and wattrset() functions change the specified window renditions of stdscr and win to new values; the old values are not retained.

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

Here is an example that prints some text using the current window rendition, adds underlining, changes the attributes, prints more text, then changes the attributes back.

    printw("This word is");
    attron(A_UNDERLINE);
    printw("underlined.");
    attroff(A_NORMAL);
    printw("This is back to normal text.\n");
    refresh();

USAGE
All of these functions may be macros.

RETURN VALUES
These functions always return OK or 1.

ERRORS
None.

SEE ALSO
addch(3XC), addnstr(3XC), attr_get(3XC), bkgdset(3XC), curses(3XC), init_color(3XC), start_color(3XC)
NAME     au_open, au_close, au_write – construct and write audit records

SYNOPSIS cc [ flag ... ] file ... -lbsm -lssocket -lnsl -lintl [ library ... ]
#include <bsm/libbsm.h>
int au_close(int d, int keep, short event);
int au_open(void);
int au_write(int d, token_t *m);

DESCRIPTION au_open() returns an audit record descriptor to which audit tokens can be written using au_write(). The audit record descriptor is an integer value that identifies a storage area where audit records are accumulated. au_close() terminates the life of an audit record d of type event started by au_open(). If the keep 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 sequence and trailer tokens may be added to the record. au_close() finally writes the record to the audit trail by calling audit(2).

au_write() adds the audit token pointed to by m to the audit record identified by the descriptor d. After this call is made the audit token is no longer available to the caller.

RETURN VALUES A successful invocation of au_write() and au_close() will return a 0.
A successful invocation of au_open() returns an audit record descriptor. au_open() returns −1 if a descriptor could not be allocated. au_open() returns −1 if d is not a valid descriptor or if audit(2) experienced an error. errno is set to indicate the error. au_write() will return −1 if d is an invalid descriptor or if m is an invalid token.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

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

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

au_preselect – preselect an audit event

**SYNOPSIS**

```c
cc [ flag ... ] file ... -lbsm -lsocket -lsl -lintl [ library ... ]
#include <bsm/libbsm.h>
int au_preselect(au_event_t event, au_mask_t *mask_p, int sofr, int flag);
```

**DESCRIPTION**

`au_preselect()` determines whether or not the audit event `event` is preselected against the binary preselection mask pointed to by `mask_p` (usually obtained by a call to `getaudit(2)`). `au_preselect()` looks up the classes associated with `event` in `audit_event(4)` and compares them with the classes in `mask_p`. If the classes associated with `event` match the classes in the specified portions of the binary preselection mask pointed to by `mask_p`, the event is said to be preselected.

`sofr` indicates whether the comparison is made with the success portion, the failure portion or both portions of the mask pointed to by `mask_p`.

The following are the valid values of `sofr`:

- **AU_PRS_SUCCESS**
  - Compare the event class with the success portion of the preselection mask.

- **AU_PRS_FAILURE**
  - Compare the event class with the failure portion of the preselection mask.

- **AU_PRS_BOTH**
  - Compare the event class with both the success and failure portions of the preselection mask.

`flag` tells `au_preselect()` how to read the `audit_event(4)` database. Upon initial invocation, `au_preselect()` reads the `audit_event(4)` database and allocates space in an internal cache for each entry with `malloc(3C)`. In subsequent invocations, the value of `flag` determines where `au_preselect()` obtains audit event information. The following are the valid values of `flag`:

- **AU_PRS_REREAD**
  - Get audit event information by searching the `audit_event(4)` database.

- **AU_PRS_USECACHE**
  - Get audit event information from internal cache created upon the initial invocation. This option is much faster.

**RETURN VALUES**

`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

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

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

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

The functionality described in this man page is available only if the Basic Security Module (BSM) has been enabled. See bsmconv(1M) for more information.
NAME
au_to, au_to_arg, au_to_attr, au_to_data, au_to_groups, au_to_in_addr, au_to_ipc,
au_to_ipc_perm, au_to_iport, au_to_me, au_to_opaque, au_to_path, au_to_process,
au_to_return, au_to_socket, au_to_text – create audit record tokens

SYNOPSIS
cc [ flag ... ] file ... –lbsm –lsocket –lnsl –lintl [ library ... ]
#include <sys/types.h>
#include <sys/vnode.h>
#include <netinet/in.h>
#include <bsm/libbsm.h>
token_t *au_to_arg(char n, char *text, u_long v);
token_t *au_to_attr(struct vattr *attr);
token_t *au_to_cmd(u_long argc, char **argv, char **envp);
token_t *au_to_data(char unit_print, char unit_type, char unit_count, char *p);
token_t *au_to_groups(int *groups);
token_t *au_to_in_addr(struct inaddr *internet_addr);
token_t *au_to_iport(u_short iport);
token_t *au_to_ipc(int id);
token_t *au_to_ipc_perm(struct ipc_perm *perm);
token_t *au_to_iport(u_short iport);
token_t *au_to_me(void);
token_t *au_to_newgroups(int n, int *groups);
token_t *au_to_opaque(char *data, short bytes);
token_t *au_to_path(char *path);
token_t *au_to_process (au_id_t auid, uid_t euid, gid_t egid, uid_t ruid, gid_t rgid,
pid_t pid, au_asid_t sid, au_tid_t *tid);
token_t *au_to_return(char number, u_int value);
token_t *au_to_socket(struct socket *so);
token_t *au_to_subject(au_id_t auid, uid_t euid, gid_t egid, uid_t ruid, gid_t rgid,
pid_t pid, au_asid_t sid, au_tid_t *tid);
token_t *au_to_text(char *text);

DESCRIPTION
au_to_arg() formats the data in v into an “argument token.” The n argument indicates the argument number. The text argument is a null terminated string describing the argument.
au_to_attr() formats the data pointed to by attr into a “vnode attribute token.”
au_to_data() formats the data pointed to by p into an “arbitrary data token.” The unit_print parameter determines the preferred display base of the data and is one of AUP_BINARY, AUP_OCTAL, AUP_DECIMAL, AUP_HEX, or AUP_STRING. The unit_type
The parameter defines the basic unit of data and is one of AUR_BYTE, AUR_CHAR, AUR_SHORT, AUR_INT, or AUR_LONG. The unit_count parameter specifies the number of basic data units to be used and must be positive.

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

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

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

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

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

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

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

au_to_subject() formats an 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.
ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO

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

NOTES

The functionality described in this man page is available only if the Basic Security Module (BSM) has been enabled. See bsmconv(1M) for more information.
NAME  au_user_mask — get user's binary preselection mask

SYNOPSIS  cc [ flag ... ] file ...  -lbsm -lsocket -lnsl -lintl [ library ... ]
#include <bsm/libbsm.h>
int au_user_mask( char *username, au_mask_t *mask_p);

DESCRIPTION  au_user_mask() reads the default, system wide audit classes from audit_control(4), combines
them with the per-user audit classes from the audit_user(4) database, and updates
the binary preselection mask pointed to by mask_p with the combined value.
The audit flags in the flags field of the audit_control(4) database and the always-audit-flags
and never-audit-flags from the audit_user(4) database represent binary audit classes.
These fields are combined by au_preselect(3) as follows:

    mask = ( flags + always-audit-flags ) − never-audit-flags

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

RETURN VALUES  au_user_mask() returns:
  0     Success.
-1     Failure. Both the audit_control(4) and the audit_user(4) database
        entries could not be retrieved.

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

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  login(1), bsmconv(1M), getaudit(2), setaudit(2), au_preselect(3), getacinfo(3),
getauusername(3), audit_control(4), audit_user(4), attributes(5)

NOTES  au_user_mask() should be called by programs like login(1) which set a process's
preselection mask with setaudit(2). getaudit(2) should be used to obtain audit characteristics for
the current process.
The functionality described in this man page is available only if the Basic Security
Module (BSM) has been enabled. See bsmconv(1M) for more information.

modified 29 Dec 1996 SunOS 5.6  3-163
NAME  basename – return the last element of a path name

SYNOPSIS  

```c
#include <libgen.h>

char *basename(char *path);
```

DESCRIPTION  The basename() function takes the pathname pointed to by path and returns a pointer to the final component of the pathname, deleting any trailing ’/’ characters.

If the string consists entirely of the ’/’ character, basename() returns a pointer to the string ”/”.

If path is a null pointer or points to an empty string, basename() returns a pointer to the string ”.”.

RETURN VALUES  The basename() function returns a pointer to the final component of path.

EXAMPLES  

<table>
<thead>
<tr>
<th>Input String</th>
<th>Output String</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>

USAGE  The basename() function may modify the string pointed to by path, and may return a pointer to static storage that may then be overwritten by a subsequent call to basename().

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  basename(1), dirname(3C), attributes(5)

NOTES  When compiling multi-thread applications, the _REENTRANT flag must be defined on the compile line. This flag should only be used in multi-thread applications.
<table>
<thead>
<tr>
<th>NAME</th>
<th>baudrate – return terminal baud rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYNOPSIS</td>
<td>#include &lt;curses.h&gt;</td>
</tr>
<tr>
<td></td>
<td>int baudrate(void);</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>The <strong>baudrate</strong>() function returns the terminal’s data communication line and output speed in bits per second (for example, 9600).</td>
</tr>
<tr>
<td>RETURN VALUES</td>
<td>The <strong>baudrate</strong>() function returns the output speed of the terminal.</td>
</tr>
<tr>
<td>ERRORS</td>
<td>None.</td>
</tr>
</tbody>
</table>
NAME
beep, flash – activate audio-visual alarm

SYNOPSIS
#include <curses.h>
int beep(void);
int flash(void);

DESCRIPTION
The beep() and flash() functions produce an audio and visual alarm on the terminal, respectively. If the terminal has the capability, beep() sounds a bell or beep and flash() flashes the screen. One alarm is substituted for another if the terminal does not support the capability called (see terminfo(4) bel and flash capabilities). For example, a call to beep() for a terminal without that capability results in a flash.

RETURN VALUES
These functions always return OK.

ERRORS
None.

SEE ALSO
terminfo(4)
NAME
bgets – read stream up to next delimiter

SYNOPSIS
cc [ flag ...] file ... -ligen [ library ...]
#include <libgen.h>
char *bgets(char *buffer, size_t *count, FILE *stream, const char *breakstring);

DESCRIPTION
bgets() reads characters from stream into buffer until either count is exhausted or one of
the characters in breakstring 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 breakstring charac-
ter is encountered, the last non-null is the delimiter character that terminated the scan.
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

bgets(buffer, sizeof buffer, stream, "\n");

is identical to

fgets(buffer, sizeof buffer, stream);
There is always enough room reserved in the buffer for the trailing null.
If breakstring is a null pointer, the value of breakstring from the previous call is used. If
breakstring is null at the first call, no characters will be used to delimit the string.

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
#include <libgen.h>
char buffer[8];
/* read in first user name from /etc/passwd */
fp = fopen("/etc/passwd","r");
bgets(buffer, 8, fp, ";");

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
gets(3S), attributes(5)

NOTES
When compiling multi-thread applications, the _REENTRANT flag must be defined on
the compile line. This flag should only be used in multi-thread applications.
bind (3N) Network Functions

NAME
bind – bind a name to a socket

SYNOPSIS
cc [ flag ...] file ... -lsocket -lnsl [ library ... ]
#include <sys/types.h>
#include <sys/socket.h>

int bind(int s, const struct sockaddr *name, int namelen);

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 name 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
The bind() call will fail if:
EACCES The requested address is protected and the current user has inadequate permission to access it.
EADDRINUSE The specified address is already in use.
EADDRNOTAVAIL The specified address is not available on the local machine.
EBADF s is not a valid descriptor.
EINVAL namelen is not the size of a valid address for the specified address family.
EINVAL The socket is already bound to an address.
ENOSR There were insufficient STREAMS resources for the operation to complete.
ENOTSOCK s is a descriptor for a file, not a socket.

The following errors are specific to binding names in the UNIX domain:
EACCES Search permission is denied for a component of the path prefix of the pathname in name.
EIO An I/O error occurred while making the directory entry or allocating the inode.
EISDIR A null pathname was specified.
ELOOP Too many symbolic links were encountered in translating the pathname in name.
ENOENT A component of the path prefix of the pathname in name does not exist.
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.
ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO

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

NOTES

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

The rules used in name binding vary between communication domains.
NAME
bind – bind a name to a socket

SYNOPSIS
cc [ flag ...] file ... -l xnet [ library ...]
#include <sys/socket.h>
int bind(int socket, const struct sockaddr *address, size_t address_len);

DESCRIPTION
The bind() function assigns an address to an unnamed socket. Sockets created with
socket(3XN) function are initially unnamed; they are identified only by their address
family.
The function takes the following arguments:
socket Specifies the file descriptor of the socket to be bound.
address Points to a sockaddr structure containing the address to be bound to the
socket. The length and format of the address depend on the address
family of the socket.
address_len Specifies the length of the sockaddr structure pointed to by the address
argument.

RETURN VALUES
Upon successful completion, bind() returns 0. Otherwise, −1 is returned and errno is set
to indicate the error.

ERRORS
The bind() function will fail if:
EBADF The socket argument is not a valid file descriptor.
ENOTSOCK The socket argument does not refer to a socket.
EADDRNOTAVAIL The specified address is not available from the local machine.
EADDRINUSE The specified address is already in use.
EINVAL The socket is already bound to an address, and the protocol does not
support binding to a new address; or the socket has been shut down.
EACCESS The specified address is protected and the current user does not have
permission to bind to it.
EAFNOSUPPORT The specified address is not a valid address for the address family of the
specified socket.
EOPNOTSUPP The socket type of the specified socket does not support binding to an
address.

If the address family of the socket is AF_UNIX, then bind() will fail if:
EDESTADDRREQ or EISDIR
The address argument is a null pointer.
EACCESS A component of the path prefix denies search permission, or the
requested name requires writing in a directory with a mode that denies
write permission.

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

ENAMETOOLONG
   A component of a pathname exceeded NAME_MAX characters, or an entire pathname exceeded PATH_MAX characters.

ENOENT    A component of the pathname does not name an existing file or the pathname is an empty string.

ELOOP     Too many symbolic links were encountered in translating the pathname in address.

EIO       An I/O error occurred.

EROS      The name would reside on a read-only filesystem.

The bind() function may fail if:

EINVAL    The address_len argument is not a valid length for the address family.

EISCONN   The socket is already connected.

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

ENOBUIDS  Insufficient resources were available to complete the call.

ENOSR     There were insufficient STREAMS resources for the operation to complete.

USAGE  An application program can retrieve the assigned socket name with the getsockname(3XN) function.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  connect(3XN), getsockname(3XN), listen(3XN), socket(3XN), attributes(5), socket(5)
NAME
bkgd, bkgdset, wbkgd, wbkgdset – set the background character (and rendition) of window

SYNOPSIS
#include <curses.h>

int bkgd(chtype ch);
int wbkgd(WINDOW *win, chtype ch);
void bkgdset(chtype ch);
void wbkgdset(WINDOW *win, chtype ch);

ARGUMENTS
ch Is a pointer to the background character to be set.
win Is a pointer to the window in which the background character is to be set.

DESCRIPTION
All characters except space are part of the foreground. The character and its attributes make up a character/rendition pair defined as a chtype. The character is any single-byte value; the attribute consists of highlighting attributes that affect the appearance of the character on the screen (for example, bold, underline, color).

The bkgdset() function sets the current background character and rendition for the stdscr window. wbkgdset() sets the current background character and rendition for window win. You must specify the complete character/rendition pair; for example:

bkgdset(A_BOLD|COLORPAIR(1)|' ');

sets the background rendition to bold with color and the background character to a space. The default background character/rendition pair is

bkgdset(A_NORMAL|COLORPAIR(0)|' ');

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

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

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

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

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

ERRORS
None.

SEE ALSO
addch(3XC), addchstr(3XC), attroff(3XC), bgkrd(3XC), clear(3XC), clrtoeol(3XC), clrtobot(3XC), erase(3XC), inch(3XC), mvprintw(3XC)
NAME | bkgrnd, bkgrndset, getbkgrnd, wbkgrnd, wbkgrndset, wgetbkgrnd — set or get the background character (and rendition) of window using a complex character

SYNOPSIS | #include <curses.h>

int bkgrnd(const cchar_t *wch);
void bkgrndset(const cchar_t *wch);
int getbkgrnd(cchar_t *wch);
int wbkgrnd(WINDOW *win, const cchar_t *wch);
void wbkgrndset(WINDOW *win, const cchar_t *wch);
int wgetbkgrnd(WINDOW *win, cchar_t *wch);

ARGUMENTS | wch | Is a pointer to the complex background character to be set.
win | Is a pointer to the window in which the complex background character is to be set.

DESCRIPTION | All characters except space are part of the foreground. The character and its attributes make up a character/rendition pair defined as a chtype. The character is any single-byte value; the attribute consists of highlighting attributes that affect the appearance of the character on the screen (for example, bold, underline, color).

If wch is a multicolumn character that cannot fit on the window line, these functions return ERR.

The bkgrndset() function sets the current background character and rendition for the stdscr window. wbkgdset() sets the current background character and rendition for window win. You must specify the complete character/rendition pair; for example:

```
bkgrndset(A_BOLD|COLOR_PAIR(1)|' ');
```

sets the background rendition to bold with color and the background character to a space. The default background character/rendition pair is

```
bkgrndset(A_NORMAL|COLOR_PAIR(0)|' ');
```

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

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

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

The getbkgrnd() and wgetbkgrnd() functions retrieve the value of the window’s background character (with rendition) and store it in the area pointed to by wch.
RETURN VALUES
The `bkgrndset()` and `wbkgrndset()` functions do not return a value. On success, the other functions return OK. Otherwise, they return ERR.

ERRORS
None.

SEE ALSO
`add_wch(3XC)`, `add_wchnstr(3XC)`, `addch(3XC)`, `addchstr(3XC)`, `attroff(3XC)`, `bkgd(3XC)`, `clear(3XC)`, `clrtoeol(3XC)`, `clrtobot(3XC)`, `erase(3XC)`, `inch(3XC)`, `mvprintw(3XC)`
NAME  
border, box, wborder – add a single-byte border to a window

SYNOPSIS  
```c
#include <curses.h>

int border(chtype ls, chtype rs, chtype ts, chtype bs,
           chtype tl, chtype tr, chtype bl, chtype br);

int wborder(WINDOW *win, chtype ls, chtype rs,
            chtype ts, chtype bs, chtype tl, chtype tr,
            chtype bl, chtype br);

int box(WINDOW *win, chtype verch, chtype horch)
```

ARGUMENTS  
- `ls`: Is the character and rendition used for the left side of the border.
- `rs`: Is the character and rendition used for the right side of the border.
- `ts`: Is the character and rendition used for the top of the border.
- `bs`: Is the character and rendition used for the bottom of the border.
- `tl`: Is the character and rendition used for the top-left corner of the border.
- `tr`: Is the character and rendition used for the top-right corner of the border.
- `bl`: Is the character and rendition used for the bottom-left corner of the border.
- `br`: Is the character and rendition used for the bottom-right corner of the border.
- `win`: Is the pointer to the window in which the border or box is to be drawn.
- `verch`: Is the character and rendition used for the left and right columns of the box.
- `horch`: Is the character and rendition used for the top and bottom rows of the box.

DESCRIPTION  
The `border()` and `wborder()` functions draw a border around the specified window. All parameters must be single-byte characters whose rendition can be expressed using only constants beginning with `ACS_`. A parameter with the value of 0 is replaced by the default value.

### Constant Values for Borders

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default Constant</th>
<th>Default Character</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>verch</code></td>
<td>ACS_VLINE</td>
<td></td>
</tr>
<tr>
<td><code>horch</code></td>
<td>ACS_HLINE</td>
<td>-</td>
</tr>
<tr>
<td><code>ls</code></td>
<td>ACS_VLINE</td>
<td></td>
</tr>
<tr>
<td><code>rs</code></td>
<td>ACS_VLINE</td>
<td></td>
</tr>
<tr>
<td><code>ts</code></td>
<td>ACS_HLINE</td>
<td>-</td>
</tr>
<tr>
<td><code>bs</code></td>
<td>ACS_HLINE</td>
<td>-</td>
</tr>
<tr>
<td><code>bl</code></td>
<td>ACS_BLCORNER</td>
<td>+</td>
</tr>
<tr>
<td><code>br</code></td>
<td>ACS_BRCORNER</td>
<td>+</td>
</tr>
<tr>
<td><code>tl</code></td>
<td>ACS_ULCORNER</td>
<td>+</td>
</tr>
<tr>
<td><code>tr</code></td>
<td>ACS_URCORNER</td>
<td>+</td>
</tr>
</tbody>
</table>

modified 1 Jun 1996  SunOS 5.6  3XC-175
The call

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

is a short form for

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

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

**RETURN VALUES**

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

**ERRORS**

None.

**SEE ALSO**

`add_wch(3XC)`, `addch(3XC)`, `attr_get(3XC)`, `attroff(3XC)`, `border_set(3XC)`
NAME  
border_set, box_set, wborder_set – use complex characters (and renditions) to draw borders

SYNOPSIS  
#include <curses.h>

int border_set(const cchar_t *ls, const cchar_t *rs, 
    const cchar_t *ts, const cchar_t *bs, 
    const cchar_t *tl, const cchar_t *tr, 
    const cchar_t *bl, const cchar_t *br);

int wborder_set(WINDOW *win, const cchar_t *ls, 
    const cchar_t *rs, const cchar_t *ts, 
    const cchar_t *bs, const cchar_t *tl, 
    const cchar_t *tr, const cchar_t *bl, 
    const cchar_t *br);

int box_set(WINDOW *win, const cchar_t *verch, 
    const cchar_t *horch)

ARGUMENTS  
ls  Is the character and rendition used for the left side of the border.
rs  Is the character and rendition used for the right side of the border.
ts  Is the character and rendition used for the top of the border.
bs  Is the character and rendition used for the bottom of the border.
tl  Is the character and rendition used for the top-left corner of the border.
tr  Is the character and rendition used for the top-right corner of the border.
bl  Is the character and rendition used for the bottom-left corner of the border.
br  Is the character and rendition used for the bottom-right corner of the border.
win Is the pointer to the window in which the border or box is to be drawn.
verch Is the character and rendition used for the left and right columns of the box.
horch Is the character and rendition used for the top and bottom rows of the box.

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default Constant</th>
<th>Default Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>verch</td>
<td>WACS_VLINE</td>
<td></td>
</tr>
<tr>
<td>horch</td>
<td>WACS_HLINE</td>
<td>-</td>
</tr>
<tr>
<td>ls</td>
<td>WACS_VLINE</td>
<td></td>
</tr>
<tr>
<td>rs</td>
<td>WACS_VLINE</td>
<td></td>
</tr>
<tr>
<td>ts</td>
<td>WACS_HLINE</td>
<td>-</td>
</tr>
<tr>
<td>bs</td>
<td>WACS_HLINE</td>
<td>-</td>
</tr>
<tr>
<td>bl</td>
<td>WACS_BLCORNER</td>
<td>+</td>
</tr>
<tr>
<td>br</td>
<td>WACS_BRCORNER</td>
<td>+</td>
</tr>
<tr>
<td>tl</td>
<td>WACS_ULCORNER</td>
<td>+</td>
</tr>
<tr>
<td>tr</td>
<td>WACS_URCORNER</td>
<td>+</td>
</tr>
</tbody>
</table>

The call

```
box_set(win, verch, horch)
```

is a short form for

```
wborder(win, verch, verch, horch, horch, NULL, NULL, NULL, NULL)
```

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

**RETURN VALUES**

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

**ERRORS**

None.

**SEE ALSO**

`add_wch(3XC)`, `addch(3XC)`, `attr_get(3XC)`, `attroff(3XC)`, `border(3XC)`
NAME
bsdmalloc, malloc, free, realloc – memory allocator

SYNOPSIS
cc [ flag ...] file ... -lbsdmalloc [ library ...]
char *malloc(size)
unsigned size;
int free(ptr)
char *ptr;
char *realloc(ptr, size)
char *ptr;
unsigned size;

DESCRIPTION
These routines provide a general-purpose memory allocation package. They maintain a
\( \text{table of free blocks for efficient allocation and coalescing of free storage.} \) When there is
no suitable space already free, the allocation routines call \text{sbrk(2)} to get more memory
from the system. Each of the allocation routines returns a pointer to space suitably
\( \text{aligned for storage of any type of object.} \) Each returns a null pointer if the request can-
not be completed (see DIAGNOSTICS).

\text{malloc()} returns a pointer to a block of at least size bytes, which is appropriately
\text{aligned.}

\text{free()} releases a previously allocated block. Its argument is a pointer to a block previously allocated by \text{malloc()} or \text{realloc()}.\n
\text{realloc()} changes the size of the block referenced by \text{ptr} to size 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, \text{realloc()} leaves its
first argument unaltered. For backwards compatibility, \text{realloc()} accepts a pointer to a
block freed since the most recent \text{call to malloc()} or \text{realloc()}.\n
RETURN VALUES
\text{malloc()} and \text{realloc()} return a null pointer if there is not enough available memory.
When \text{realloc()} returns NULL, the block pointed to by \text{ptr} is left intact.

ERRORS
If \text{malloc()} or \text{realloc()} returns unsuccessfully, \text{errno} will be set to indicate the following:

\text{ENOMEM} size bytes of memory exceeds the physical limits of your system, and
cannot be allocated.

\text{EAGAIN} There is not enough memory available at this point in time to allocate
\text{size bytes of memory;} but the application could try again later.

SEE ALSO
\text{brk(2), malloc(3C), malloc(3X), mapmalloc(3X)}\n
WARNINGS
Use of libbsdmalloc renders an application non-SCD compliant.

libbsdmalloc routines are incompatible with the memory allocation routines in the stan-
dard C-library (libc): \text{malloc(3C), alloca(3C), calloc(3C), free(3C), memalign(3C),
realloc(3C), and valloc(3C)}.

modified 11 Feb 1993 SunOS 5.6 3X-179
NOTES

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

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

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

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

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

`free()` does not set `errno`. 
NAME     bsd_signal – simplified signal facilities

SYNOPSIS  #include <signal.h>
            void (*bsd_signal(int sig, void (*func) (int))) (int);

DESCRIPTION The bsd_signal() function provides a partially compatible interface for programs written
to historical system interfaces (see USAGE below).

The function call bsd_signal(sig, func) has an effect as if implemented as:

    void (*bsd_signal(int sig, void (*func) (int))) (int)
    {
        struct sigaction act, oact;
        act.sa_handler = func;
        act.sa_flags = SA_RESTART;
        sigemptyset(&act.sa_mask);
        sigaddset(&act.sa_mask, sig);
        if (sigaction(sig, &act, &oact) == -1)
            return(SIG_ERR);
        return(oact.sa_handler);
    }

The handler function should be declared:

    void handler(int sig);

where sig is the signal number. The behavior is undefined if func is a function that takes
more than one argument, or an argument of a different type.

RETURN VALUES Upon successful completion, bsd_signal() returns the previous action for sig. Otherwise,
SIG_ERR is returned and errno is set to indicate the error.

ERRORS Refer to sigaction(2).

USAGE This function is a direct replacement for the BSD signal(3B) function for simple applica-
tions that are installing a single-argument signal handler function. If a BSD signal handler
function is being installed that expects more than one argument, the application has to be
modified to use sigaction(2). The bsd_signal() function differs from signal(3B) in that
the SA_RESTART flag is set and the SA_RESETHAND will be clear when bsd_signal() is
used. The state of these flags is not specified for signal(3B).

SEE ALSO sigaction(2), sigaddset(3C), sigemptyset(3C), signal(3B)
NAME
bsearch – binary search a sorted table

SYNOPSIS
#include <stdlib.h>
void *bsearch(const void *key, const void *base, size_t nel, size_t size,
   int (*compar)(const void *, const void *));

DESCRIPTION
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 compar. key points to a datum
instance to be sought in the table. base points to the element at the base of the table. nel is
the number of elements in the table. size is the number of bytes in each element. The
function pointed to by compar 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.

RETURN VALUES
A null pointer is returned if the key cannot be found in the table.

EXAMPLES
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.
This program reads in strings and either finds the corresponding node and prints out the
string and its length, or prints an error message.

   #include <stdio.h>
   #include <stdlib.h>
   #include <string.h>

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

   main()
   {
      struct node *node_ptr, node;
      /∗ routine to compare 2 nodes ∗/
      static int node_compare(const void *, const void *);
C Library Functions

bsearch (3C)

```c
char str_space[20]; /* space to read string into */

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

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

ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  

hsearch(3C), lsearch(3C), qsort(3C), tsearch(3C), attributes(5)

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.

modified 29 Dec 1996
NAME
bstring, bcopy, bcmp, bzero – bit and byte string operations

SYNOPSIS
#include <strings.h>
void bcopy(const void *s1, void *s2, size_t n);
int bcmp(const void *s1, const void *s2, size_t n);
void bzero(void *s, size_t n);

DESCRIPTION
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.
bcopy() copies n bytes from string s1 to the string s2. Overlapping strings are handled correctly.
bcmp() compares byte string s1 against byte string s2, returning zero if they are identical, 1 otherwise. Both strings are assumed to be n bytes long. bcmp() using n zero bytes always returns zero.
bzero() places n 0 bytes in the string s.

WARNINGS
The bcmp() and bcopy() routines take parameters backwards from strcmp and strcpy, respectively. See string(3C).

SEE ALSO
memory(3C), string(3C)
NAME
bufsplit – split buffer into fields

SYNOPSIS
cc [ flag ...] file ... -lgen [ library ...]
#include <libgen.h>
size_t bufsplit(char *buf, size_t n, char **a);

DESCRIPTION
bufsplit() examines the buffer, buf, and assigns values to the pointer array, a, so that the
pointers point to the first n fields in buf that are delimited by tabs or new-lines.
To change the characters used to separate fields, call bufsplit() with buf pointing to the
string of characters, and n and a set to zero. For example, to use ‘:’, ‘,’ and ‘,’ as separators
along with tab and new-line:
    bufsplit (":\,\t\n", 0, (char**)0);

RETURN VALUES
The number of fields assigned in the array a. If buf is zero, the return value is zero and
the array is unchanged. Otherwise the value is at least one. The remainder of the ele-
ments in the array are assigned the address of the null byte at the end of the buffer.

EXAMPLES
/*
 * set a[0] = "This", a[1] = "is", a[2] = "a",
 * a[3] = "test"
 */
bufsplit("This\tis\ta\ttest\n", 4, a);

NOTES
bufsplit() changes the delimiters to null bytes in buf.
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.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
attributes(5)
byteorder (3N)  Network Functions

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

SYNOPSIS  
#include <sys/types.h>
#include <netinet/in.h>
ulong htonl(u_long hostlong);
short htons(u_short hostshort);
ulong ntohl(u_long netlong);
short ntohs(u_short netshort);

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

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

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  gethostbyname(3N), getservbyname(3N), attributes(5), inet(5)
cancellation – overview of concepts related to POSIX thread cancellation

<table>
<thead>
<tr>
<th>FUNCTION</th>
<th>ACTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>pthread_cancel</td>
<td>Cancels thread execution.</td>
</tr>
<tr>
<td>pthread_setcancelstate</td>
<td>Sets the cancellation state of a thread.</td>
</tr>
<tr>
<td>pthread_setcanceltype</td>
<td>Sets the cancellation type of a thread.</td>
</tr>
<tr>
<td>pthread_testcancel</td>
<td>Creates a cancellation point in the calling thread.</td>
</tr>
<tr>
<td>pthread_cleanup_push</td>
<td>Pushes a cleanup handler routine.</td>
</tr>
<tr>
<td>pthread_cleanup_pop</td>
<td>Pops a cleanup handler routine.</td>
</tr>
</tbody>
</table>

Cancellation

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

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

Planning Steps

Planning and programming for most cancellations follow this pattern:

1. Identify which threads you want to cancel, and insert `pthread_cancel(3T)` statements.
2. Identify system-defined cancellation points where a thread that might be canceled could have changed system or program state that should be restored. See the Cancellation Points for a list.
3. When a thread changes the system or program state just before a cancellation point, and should restore that state before the thread is canceled, place a cleanup handler before the cancellation point with `pthread_cleanup_push(3T)`. Wherever a thread restores the changed state, pop the cleanup handler from the cleanup stack with `pthread_cleanup_pop(3T)`.
4. Know whether the threads you are canceling call into cancel-unsafe libraries, and disable cancellation with `pthread_setcancelstate(3T)` before the call into the library. See Cancellation State and Cancel-Safe.
5. To cancel a thread in a procedure that contains no cancellation points, insert your own cancellation points with `pthread_testcancel(3T)`. `pthread_testcancel(3T)` creates cancellation points by testing for pending cancellations and performing those cancellations if they are found. Push and pop cleanup handlers around the cancellation point, if necessary (see Step 3, above).
Cancellation Points

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

The following cancellation points are defined by the system:

<table>
<thead>
<tr>
<th>System-Defined Cancellation Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>aio_suspend(3R)</td>
</tr>
<tr>
<td>creat(2)</td>
</tr>
<tr>
<td>mq_receive(3R)</td>
</tr>
<tr>
<td>msync(3C)</td>
</tr>
<tr>
<td>open(2)</td>
</tr>
<tr>
<td>pthread_cond_timedwait(3T)</td>
</tr>
<tr>
<td>pthread_join(3T)</td>
</tr>
<tr>
<td>pthread_setcanceltype(3T)</td>
</tr>
<tr>
<td>read(2)</td>
</tr>
<tr>
<td>sigwaitinfo(3R)</td>
</tr>
<tr>
<td>sigtimedwait(3R)</td>
</tr>
<tr>
<td>sleep(3C)</td>
</tr>
<tr>
<td>tcdrain(3)</td>
</tr>
<tr>
<td>waitpid(2)</td>
</tr>
<tr>
<td>fcntl(2) (when specifying F_SETLKW as the command)</td>
</tr>
<tr>
<td>close(2)</td>
</tr>
<tr>
<td>fsync(3C)</td>
</tr>
<tr>
<td>mq_send(3R)</td>
</tr>
<tr>
<td>nanosleep(3R)</td>
</tr>
<tr>
<td>pause(2)</td>
</tr>
<tr>
<td>pthread_cond_wait(3T)</td>
</tr>
<tr>
<td>pthread_setcancelstate(3T)</td>
</tr>
<tr>
<td>pthread_testcancel(3T)</td>
</tr>
<tr>
<td>sem_wait(3R)</td>
</tr>
<tr>
<td>sigsuspend(2)</td>
</tr>
<tr>
<td>sigwait(2)</td>
</tr>
<tr>
<td>system(3S)</td>
</tr>
<tr>
<td>wait(2)</td>
</tr>
<tr>
<td>write(2)</td>
</tr>
</tbody>
</table>

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

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

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

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

Most of the dangers in performing cancellations deal with properly restoring invariants and freeing shared resources. For example, a carelessly canceled thread might leave a mutex in a locked state, leading to a deadlock. Or it might leave a region of memory allocated with no way to identify it and therefore no way to free it.
Cleanup Handlers

When a thread is canceled, it should release resources and clean up the state that is shared with other threads. So, whenever a thread that might be canceled changes the state of the system or of the program, be sure to push a cleanup handler with `pthread_cleanup_push(3T)` before the cancellation point.

When a thread is canceled, all the currently-stacked cleanup handlers are executed in last-in-first-out (LIFO) order. Each handler is run in the scope in which it was pushed. When the last cleanup handler returns, the thread-specific data destructor functions are called. Thread execution terminates when the last destructor function returns.

When, in the normal course of the program, an uncanceled thread restores state that it had previously changed, be sure to pop the cleanup handler (that you had set up where the change took place) using `pthread_cleanup_pop(3T)`. That way, if the thread is canceled later, only currently-changed state will be restored by the handlers that are left in the stack.

Be sure to pop the handler in the same scope in which it was pushed. Also, make sure that each push statement has a matching pop statement, or compiler errors will be generated.

Cancellation State

Most programmers will use only the default cancellation state of `PTHREAD_CANCEL_ENABLE`, but can choose to change the state by using `pthread_setcancelstate(3T)`, which determines whether a thread is cancelable at all. With the default state of `PTHREAD_CANCEL_ENABLE`, cancellation is enabled, and the thread is cancelable at points determined by its cancellation type. See Cancellation Type.

If the state is `PTHREAD_CANCEL_DISABLE`, cancellation is disabled, and the thread is not cancelable at any point — all cancellation requests to it are held pending.

You might want to disable cancellation before a call to a cancel-unsafe library, restoring the old cancel state when the call returns from the library. See Cancel-Safe for explanations of cancel safety.

Cancellation Type

A thread’s cancellation type is set with `pthread_setcanceltype(3T)`, and determines whether the thread can be canceled anywhere in its execution, or only at cancellation points.

With the default type of `PTHREAD_CANCEL_DEFERRED`, the thread is cancelable only at cancellation points, and then only when cancellation is enabled.

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

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

Cancel-Safe

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

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

POSIX Threads Only

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

EXAMPLES

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

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

When `f2()` is called for the fifty-first time (when "i == 50"), `f2()` enables cancellation by calling `pthread_setcancelstate()`. It then establishes a cancellation point for itself by calling `pthread_testcancel()`. (Because a cancellation is pending, a call to a cancellation point such as `read(2)` or `write(2)` would also cancel the caller here.)
After the main() thread is canceled at the fifty-first iteration, all the cleanup handlers that were pushed are called in sequence; this is indicated by the calls to free_res() and the calls to the destructor for X. At each level, the C++ runtime calls the destructor for X and then the cancellation handler, free_res(). The print messages from free_res() and X’s destructor show the sequence of calls.

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

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

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

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

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

void *
```

modified 8 May 1997 SunOS 5.6 3T-191
thread2(void *tid)
{
    void *sts;

    printf("I am new thread :%d\n", pthread_self());
    pthread_cancel((pthread_t)tid);
    pthread_join((pthread_t)tid, &sts);
    printf("main thread canceled due to %d\n", sts);
    return (sts);
}

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

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
read(2), sigwait(2), write(2), Intro(3), condition(3T), pthread_cleanup_pop(3T),
pthread_cleanup_push(3T), pthread_exit(3T), pthread_join(3T),
pthread_setcancelstate(3T), pthread_setcanceltype(3T), pthread_testcancel(3T),
setjmp(3C), attributes(5), standards(5)
X/Open Curses Library Functions

NAME

can_change_color, color_content, COLOR_PAIR, has_colors, init_color, init_pair, pair_content, PAIR_NUMBER, start_color – manipulate color information

SYNOPSIS

#include <curses.h>

bool can_change_color(void);

int color_content(short color, short *r, short *g, short *b);

int COLOR_PAIR(int n);

bool has_colors(void);

int init_color(short color, short r, short g, short b);

int init_pair(short pair, short fg, short bg);

int pair_content(short pair, short *fg, short *bg);

int PAIR_NUMBER(int value);

int start_color(void);

ARGUMENTS

color Is the number of the color for which to provide information (0 to COLORS).

r Is a pointer to the RGB value for the amount of red in color.

g Is a pointer to the RGB value for the amount of green in color.

b Is a pointer to the RGB value for the amount of blue in color.

n Is the number of a color pair.

pair Is the number of the color pair for which to provide information (1 to COLOR_PAIRS).

fg Is a pointer to the number of the foreground color (0 to COLORS) in pair.

bg Is a pointer to the number of the background color (0 to COLORS) in pair.

DESCRIPTION

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

The init_pair() function initializes a color pair so that it can be used as a parameter. COLOR_PAIR() can be used as an attribute and as a parameter to functions like attr_set(3XC). Its first parameter is the number of the color pair to be changed; the second parameter is the number of the foreground color; the third parameter is the number of the background color. The maximum number of color pairs and colors the
terminal can support are defined in the global variables COLOR_PAIRS and COLORS, respectively.
Color pair 0 (zero) is reserved for use by X/Open Curses.
Each time that a color pair is initialized, the screen is refreshed and all occurrences of that color pair are updated to reflect the new definition.
The init_color() function redefines the color using the number of the color and the RGB values for red, green, and blue as parameters.
The following default colors are defined (X/Open Curses assumes that COLOR_BLACK is the default background color for all terminals):

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

Each time that a color is redefined with the init_color() function, the screen is refreshed and all occurrences of that color are updated to reflect the new definition.
The can_change_color() function returns TRUE if the terminal supports color and the colors can be changed. The has_colors() function returns TRUE if the terminal supports color. These functions are useful when writing terminal-independent programs. They can be used to determine whether to replace color with another attribute on a particular terminal.

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

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

**RETURN VALUES**

The has_colors() function returns TRUE if the terminal is able to handle colors. Otherwise, it returns FALSE.
The can_change_color() function returns TRUE if the terminal supports colors and is able to change their definitions. Otherwise, it returns FALSE.
On success, the other functions return OK. Otherwise, they return ERR.

**ERRORS**

None.
### SEE ALSO
- `attroff(3XC)`, `delscreen(3XC)`

---

modified 1 Jun 1996

SunOS 5.6

3XC-195
NAME  
catgets – read a program message

SYNOPSIS  
#include <nl_types.h>
char *catgets(nl_catd catd, int set_num, int msg_num, const char *s);

DESCRIPTION  
catgets() attempts to read message msg_num, in set set_num, from the message catalog
identified by catd. catd is a catalog descriptor returned from an earlier call to catopen(). s
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 s and errno may be set to indicate the error.

ERRORS  
The catgets() function may fail if:
EBADF   The catd argument is not a valid message catalogue descriptor open for reading.
EINTR   The read operation was terminated due to the receipt of a signal, and no data
was transferred.
EINVAL   The message catalog identified by catd is corrupted.
ENOMSG   The message identified by set_id and msg_id is not in the message catalog.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  
gencat(1), catclose(3C), catopen(3C), gettext(3C), setlocale(3C), attributes(5)

Solaris Internationalization Guide For Developers

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

#include <nl_types.h>

nl_catd catopen(const char *name, int oflag);

int catclose(nl_catd catd);

DESCRIPTION

catopen() opens a message catalog and returns a message catalog descriptor. name specifies the name of the message catalog to be opened. If name contains a “/”, then name specifies a complete pathname for the message catalog; otherwise, the environment variable NLSPATH is used and /usr/lib/locale/LC_MESSAGES must exist. If NLSPATH does not exist in the environment, or if a message catalog cannot be opened in any of the paths specified by NLSPATH, then the default path /usr/lib/locale/LC_MESSAGES is used. In the “C” locale, catopen() will always succeed without checking the default search path.

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.

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:

NLSPATH=/nlslib/%L/%N.cat:/nlslib/%N/%L

The metacharacter % introduces a substitution field, where %L substitutes the current setting of either the LANG environment variable, if the value of oflag is 0, or the LC_MESSAGES category, if the value of oflag is NL_CAT_LOCALE, and %N substitutes the value of the name parameter passed to catopen(). Thus, in the above example, catopen() will search in /nlslib/$LANG/name.cat, if oflag is 0, or in /nlslib/$LC_MESSAGES/name.cat, if oflag is NL_CAT_LOCALE.

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 catalog files transparent to both programs and users.

The full set of metacharacters is:

- %N The value of the name parameter passed to catopen().
- %L The value of LANG or LC_MESSAGES.
- %l The value of the language element of LANG or LC_MESSAGES.
- %t The value of the territory element of LANG or LC_MESSAGES.
- %c The value of the codeset element of LANG or LC_MESSAGES.
- % A single %.

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

modified 29 Dec 1996

SunOS 5.6

3C-197
LANG=language[_territory[.codeset]]
A user who speaks German as it is spoken in Austria and has a terminal which operates in ISO 8859/1 codeset, would want the setting of the LANG variable to be

LANG=De_A.88591
With this setting it should be possible for that user to find any relevant catalogs should they exist.

Should the LANG variable not be set, the value of LC_MESSAGES as returned by setlocale() is used. If this is NULL, the default path as defined in nl_types() is used.
A message catalogue descriptor remains valid in a process until that process closes it, or a successful call to one of the exec functions. A change in the setting of the LC_MESSAGES category may invalidate existing open catalogues.
If a file descriptor is used to implement message catalogue descriptors, the FD_CLOEXEC flag will be set; see <fcntl.h>.
If the value of oflag argument is 0, the LANG environment variable is used to locate the catalogue without regard to the LC_MESSAGES category. If the oflag argument is NL_CAT_LOCALE, the LC_MESSAGES category is used to locate the message catalogue.
catclose() closes the message catalog identified by catd. If a file descriptor is used to implement the type nl_catd, that file descriptor will be closed.

**RETURN VALUES**
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 and sets errno to indicate the error.

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

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

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

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

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

ENFILE 
Too many files are currently open in the system.

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

ENOMEM 
Insufficient storage space is available.

ENOTDIR 
A component of the path prefix of the message catalogue is not a directory.
The `catclose()` function may fail if:

- **EBADF**  
The catalogue descriptor is not valid.
- **EINTR**  
The `catclose()` function was interrupted by a signal.

### ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### SEE ALSO

`gencat(1), catgets(3C), gettext(3C), setlocale(3C), attributes(5), 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  cbreak, nocbreak, noraw, raw – set input mode controls

SYNOPSIS  
```
#include <curses.h>
int cbreak(void);
int nocbreak(void);
int noraw(void);
int raw(void);
```

DESCRIPTION  The `cbreak()` function enables the character input mode. This overrides any previous call to the `raw()` function and turns the `stty` flag `ICANON` off.

The `nocbreak()` function sets the line canonical mode and turns the `stty` flag `ICANON` on without touching the `ISIG` or `IXON` flags.

The `noraw()` function sets the line canonical mode and turns the the `stty` flags `ICANON`, `ISIG`, and `IXON` all on.

The `raw()` function sets the character input mode and turns the `stty` flags `ICANON`, `ISIG`, and `IXON` all off. This mode provides maximum control over input.

It is important to remember that the terminal may or may not be in character mode operation initially. Most interactive programs require `cbreak()` to be enabled.

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

ERRORS  None.

SEE ALSO  `getch(3XC)`, `halfdelay(3XC)`, `nodelay(3XC)`, `timeout(3XC)`, `termio(7I)`
NAME  cbrt – cube root function

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

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

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

ERRORS  No errors will occur.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  attributes(5)
NAME  ceil – ceiling value function

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

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

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

ERRORS  No errors will occur.

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

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  floor(3M), isnan(3M), attributes(5)
NAME

cfgetispeed, cfgetospeed – get input and output baud rate

SYNOPSIS

#include <termios.h>
speed_t cfgetispeed(const struct termios *termios_p);
speed_t cfgetospeed(const struct termios *termios_p);

DESCRIPTION

The cfgetispeed() function extracts the input baud rate from the termios structure to which the termios_p argument points.
The cfgetospeed() function extracts the output baud rate from the termios structure to which the termios_p argument points.
These functions returns exactly the value in the termios data structure, without interpretation.

RETURN VALUES

Upon successful completion, cfgetispeed() returns a value of type speed_t representing the input baud rate.
Upon successful completion, cfgetospeed() returns a value of type speed_t representing the output baud rate.

ERRORS

No errors are defined.

ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe, and Async-Signal-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO

cfgetospeed(3), tcgetattr(3), attributes(5), termio(7I)

modified 29 Dec 1996
NAME  cfsetispeed, cfsetospeed – set input and output baud rate

SYNOPSIS  #include <termios.h>
            int cfsetispeed(struct termios *termios_p, speed_t speed);
            int cfsetospeed(struct termios *termios_p, speed_t speed);

DESCRIPTION The cfsetispeed() function sets the input baud rate stored in the structure pointed to by termios_p to speed.
            The cfsetospeed() function sets the output baud rate stored in the structure pointed to by termios_p to speed.
            There is no effect on the baud rates set in the hardware until a subsequent successful call to tcsetattr(3) on the same termios structure.

RETURN VALUES Upon successful completion, cfsetispeed() and cfsetospeed() return 0. Otherwise −1 is returned, and errno may be set to indicate the error.

ERRORS The cfsetispeed() and cfsetospeed() functions may fail if:
            EINVAL   The speed value is not a valid baud rate.
            EINVAL   The value of speed is outside the range of possible speed values as specified in <termios.h>.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe, and Async-Signal-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO cfgetispeed(3), tcsetattr(3), attributes(5), termio(7I)
NAME
chgat, mvchgat, mvwchgat, wchgat – change the rendition of characters in a window

SYNOPSIS
#include <curses.h>

int chgat(int n, attr_t attr, short color, void *const opts);
int mvchgat(int y, int x, int n, attr_t attr, short color, void *const opts);
int mvwchgat(WINDOW *win, int y, int x, int n, attr_t attr, short color, void *const opts);
int wchgat(WINDOW *win, int n, attr_t attr, short color, void *const opts);

ARGUMENTS
n Is the number of characters whose rendition is to be changed.
attr Is the set of attributes to be assigned to the characters.
color Is the new color pair to be assigned to the characters.
opts Is reserved for future use. Currently, this must be a null pointer.
y Is the y (row) coordinate of the starting position in the window.
x Is the x (column) coordinate of the starting position in the window.
win Is a pointer to the window in which the rendition of characters is to be changed.

DESCRIPTION
The chgat() and wchgat() functions change the rendition (that is, the attributes and color pair) associated with the next n characters beginning at the current cursor position in the windows stdscr and win, respectively. The mvchgat() and mvwchgat() perform identical actions but beginning with the position indicated by x (column) and y (row) instead of the current cursor position. If n is less than 0, these functions change the rendition of all characters from the starting position to the end of that line. The cursor position is not changed.

ERRORS
OK Successful completion.
ERR An error occurred.

SEE ALSO
bkgrnd(3XC), setcchar(3XC)
### NAME
clear, erase, wclear, werase – clear a window

### SYNOPSIS
```c
#include <curses.h>

int clear(void);
int erase(void)
int wclear(WINDOW *win);
int werase(WINDOW *win);
```

### ARGUMENTS
- `win` Is a pointer to the window that is to be cleared.

### DESCRIPTION
The `clear()` and `erase()` functions clear `stdscr`, destroying its previous contents. The `wclear()` and `werase()` functions perform the same action, but clear the window specified by `win` instead of `stdscr`.

The `clear()` and `wclear()` functions also call the `clearok()` function. This function clears and redraws the entire screen on the next call to `refresh(3XC)` or `wrefresh(3XC)` for the window.

The current background character (and attributes) is used to clear the screen.

### ERRORS
- **OK** Successful completion.
- **ERR** An error occurred.

### SEE ALSO
- `bkgdset(3XC)`, `clearok(3XC)`, `clrtobot(3XC)`, `clrtoeol(3XC)`, `doupdate(3XC)`, `refresh(3XC)`, `wrefresh(3XC)`
NAME
clearok, idlok, leaveok, scrollok, setscrreg, wssetscrreg – set terminal output controls

SYNOPSIS
#include <curses.h>
int clearok(WINDOW *win, bool bf);
int idlok (WINDOW *win, bool bf);
int leaveok (WINDOW *win, bool bf);
int scrollok (WINDOW *win, bool bf);
int setscrreg (int top, int bot);
int wssetscrreg (WINDOW *win, int top, int bot);

ARGUMENTS
win Is a pointer to a window.
bf Is a Boolean expression.
top Is the top line of the scrolling region (top of the window is line 0).
bot Is the bottom line of the scrolling region (top of the window is line 0).

DESCRIPTION
These functions set options that deal with the output of X/Open Curses functions. The clearok() function checks the value of the Boolean expression bf. If bf is TRUE, clearok() clears and redraws the entire screen on the next call to refresh(3XC). If win is curscr, the next call to refresh() for any window clears and redraws the screen.

The idlok() function enables (bf is TRUE) or disables (bf is FALSE) the use of the insert/delete line capability of the terminal, provided that the terminal supports the operation. By default, the use of insert/delete line is disabled because its use is undesirable for most applications (screen editor applications are one exception). When disabled, X/Open Curses redraws the changed portions of all lines.

The leaveok() function controls the cursor positioning following a call to the refresh() function. If bf is TRUE, leaveok() leaves the cursor in a position that X/Open Curses finds convenient at the time that the window is refreshed. Normally, when a window is refreshed, leaveok() is disabled and the cursor is mapped from the logical window to the same location on the physical screen.

Enabling leaveok() is useful when the cursor is not used or is not important in the application. Reducing cursor movements simplifies program interaction.

Once leaveok() is set to TRUE, it remains enabled until another call sets it to FALSE, or until the program terminates.

The scrollok() function controls what happens when the cursor advances outside the scrolling region. When enabled, if the cursor advances outside the scrolling region or a call to the scrl (3XC) function is made, the screen scrolls up one line.

The terminal screen will produce a scrolling effect if idlok() is also enabled.

The setscreg() and wssetscreg() functions set up scrolling regions in the windows stdscr and win, respectively. The dimensions of the scrolling region are defined by the top and bottom parameter. If scrollok() is enabled and the cursor is on the last line of the scroll

modified 1 Jun 1996
SunOS 5.6
3XC-207
region, any attempt to move the cursor beyond the bottom margin of the scrolling region scrolls the scrolling region up by one line. By default, the scrolling region of a window is the entire window.

For full screen windows, the terminal screen produces a scrolling effect if \texttt{idlok()} is also enabled.

**RETURN VALUES**

On success, the \texttt{setscrreg()} and \texttt{wsetscrreg()} functions return \texttt{OK}. Otherwise, they return \texttt{ERR}.

The other functions always return \texttt{OK}.

**ERRORS**

None.

**SEE ALSO**

\texttt{bkgdset(3XC)}, \texttt{clear(3XC)}, \texttt{doupdate(3XC)}, \texttt{scrl(3XC)}
NAME  
clock – report CPU time used

SYNOPSIS  
#include <time.h>

clock_t clock(void);

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.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  
times(2), wait(2), pclose(3S), system(3S), attributes(5)

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   cc [ flag ... ] file ... -lpthread [ 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 */
            };

DESCRIPTION clock_settime() sets the specified clock, clock_id, to the value specified by tp. The calling process must have an effective user ID of 0.

clock_gettime() returns the current value tp for the specified clock, clock_id.

The resolution of any clock can be obtained by calling clock_getres(). If res is not NULL, the resolution of the specified clock is stored in res.

The clock_id 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.

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 EINVAL   clock_id does not specify a known clock.

            The tp argument to clock_settime() is outside the range for the given clock id.

            The tp argument to clock_settime() specified a nanosecond value less than zero or greater than or equal to 1,000,000,000.

ENOSYS   clock_settime(), clock_gettime(), or clock_getres() is not supported by this implementation.

EPERM   The requesting process does not have the appropriate privilege to set the specified clock.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>clock_gettime() is Async-Signal-Safe</td>
</tr>
</tbody>
</table>
SEE ALSO  time(2), ctime(3C), timer_gettime(3R), attributes(5)

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  
closedir – close a directory stream

SYNOPSIS  
#include <sys/types.h>
#include <dirent.h>

int closedir(DIR *dirp);

DESCRIPTION  
The closedir() function closes the directory stream referred to by the argument dirp. Upon return, the value of dirp may no longer point to an accessible object of the type DIR. If a file descriptor is used to implement type DIR, that file descriptor will be closed.

RETURN VALUES  
Upon successful completion, closedir() returns 0. Otherwise, −1 is returned and errno is set to indicate the error.

ERRORS  
The closedir() function may fail if:

EBADF   The dirp argument does not refer to an open directory stream.
EINTR   The closedir() function was interrupted by a signal.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  
opendir(3C), attributes(5)
NAME          clrtobot, wclrtobot – clear to the end of a window

SYNOPSIS      #include <curses.h>
               int clrtobot(void);
               int wclrtobot(WINDOW *win);

ARGUMENTS     win       Is a pointer to the window that is to be cleared.

DESCRIPTION   The clrtobot() function clears all characters in the stdscr window from the cursor to the end of the window. The wclrtobot() function performs the same action in the window specified by win instead of in stdscr. The current background character (and rendition) is used to clear the screen.
               If the clearing action results in clearing only a portion of a multicolumn character, background characters are displayed in place of the remaining portion.

RETURN VALUES On success, these functions return OK. Otherwise, they return ERR.

ERRORS        None.

SEE ALSO      bkgdset(3XC), clear(3XC), clearok(3XC), crltoeol(3XC)
NAME    clrtoeol, wclrtoeol – clear to the end of a line

SYNOPSIS #include <curses.h>
int clrtoeol(void);
int wclrtoeol(WINDOW *win);

ARGUMENTS    win    Is a pointer to the window in which to clear to the end of the line.

DESCRIPTION The clrtoeol() function clears the current line from the cursor to the right margin in the stdscr window. The wclrtoeol() function performs the same action, but in the window specified by win instead of stdscr. The current background character (and rendition) is used to clear the screen.
If the clearing action results in clearing only a portion of a multicolumn character, background characters are displayed in place of the remaining portion.

RETURN VALUES    On success, these functions return OK. Otherwise, they return FALSE.

ERRORS    None.

SEE ALSO    bkgdset(3XC), clear(3XC), clearok(3XC), clrtobot(3XC)
Threads Library

NAME

condition, pthread_cond_init, pthread_cond_wait, pthread_cond_timedwait,
pthread_cond_signal, pthread_cond_broadcast, pthread_cond_destroy, cond_init,
cond_wait, cond_timedwait, cond_signal, cond_broadcast, cond_destroy – condition
variables

SYNOPSIS

POSIX

cc [ flag ... ] file ... -l pthread [ library ... ]
#include <pthread.h>
int pthread_cond_init(pthread_cond_t *cond, const pthread_condattr_t *attr);
int pthread_cond_wait(pthread_cond_t *cond, pthread_mutex_t *mutex);
int pthread_cond_timedwait(pthread_cond_t *cond, pthread_mutex_t *mutex,
const struct timespec *abstime);
int pthread_cond_signal(pthread_cond_t *cond);
int pthread_cond_broadcast(pthread_cond_t *cond);
int pthread_cond_destroy(pthread_cond_t *cond);

Solaris

cc [ flag ... ] file ... -l pthread [ library ... ]
#include <thread.h>
#include <synch.h>
int cond_init(cond_t *cvp, int type, void *arg);
int cond_wait(cond_t *cvp, mutex_t *mp);
int cond_timedwait(cond_t *cvp, mutex_t *mp, timestruc_t *abstime);
int cond_signal(cond_t *cvp);
int cond_broadcast(cond_t *cvp);
int cond_destroy(cond_t *cvp);

DESCRIPTION

Occasionally, a thread running within a mutex needs to wait for an event, in which case it
blocks or sleeps. When a thread is waiting for another thread to communicate its disposi-
tion, it uses a condition variable in conjunction with a mutex. Although a mutex is
exclusive and the code it protects is sharable (at certain moments), condition variables
enable the synchronization of differing events that share a mutex, but not necessarily
data. Several condition variables may be used by threads to signal each other when a
task is complete, which then allows the next waiting thread to take ownership of the
mutex.

A condition variable enables threads to atomically block and test the condition under the
protection of a mutual exclusion lock (mutex) until the condition is satisfied. If the condi-
tion is false, a thread blocks on a condition variable and atomically releases the mutex
that is waiting for the condition to change. If another thread changes the condition, it
may wake up waiting threads by signaling the associated condition variable. The waiting
threads, upon awakening, reacquire the mutex and re-evaluate the condition.

Initialize

Condition variables and mutexes should be global. Condition variables that are allocated
in writable memory can synchronize threads among processes if they are shared by the
cooperating processes (see mmap(2)) and are initialized for this purpose.

modified 8 May 1997

SunOS 5.6

3T-215
The scope of a condition variable is either intra-process or inter-process. This is dependent upon whether the argument is passed implicitly or explicitly to the initialization of that condition variable. A condition variable does not need to be explicitly initialized. A condition variable is initialized with all zeros, by default, and its scope is set to within the calling process. For inter-process synchronization, a condition variable must be initialized once, and only once, before use.

A condition variable must not be simultaneously initialized by multiple threads or re-initialized while in use by other threads.

Condition variables attributes may be set to the default or customized at initialization. POSIX threads even allow the default values to be customized. Establishing these attributes varies depending upon whether POSIX or Solaris threads are used. Similar to the distinctions between POSIX and Solaris thread creation, POSIX condition variables implement the default, intra-process, unless an attribute object is modified for inter-process prior to the initialization of the condition variable. Solaris condition variables also implement as the default, intra-process; however, they set this attribute according to the argument, type, passed to their initialization function.

**POSIX Initialize**

POSIX condition variables mutexes, and threads use attributes objects in the same manner; they are initialized with the configuration of an attributes object (see `pthread_condattr_init(3T)`). The `pthread_cond_init()` function initializes the condition variable referenced by cond with attributes referenced by attr. If attr is NULL, the default condition variable attributes are used, which is the same as passing the address of a default condition variable attributes object. When the initialization is complete, the state of the condition variable is then initialized. If a default condition variable is used, then only threads created within the same process can operate on the initialized condition variable.

A condition variable can possess two different types of shared-scope behavior, which is determined by the second argument to `pthread_condattr_setpshared(3T)`. This argument can be set to either of the following:

**PTHREAD_PROCESS_PRIVATE**
The condition variable can synchronize threads only in this process. The PTHREAD_PROCESS_PRIVATE POSIX setting for process scope is equivalent to the USYNC_THREAD flag to `cond_init()` in the Solaris API. This is the default.

**PTHREAD_PROCESS_SHARED**
The condition variable can synchronize threads in this process and other processes. Only one process should initialize the condition variable. The PTHREAD_PROCESS_SHARED POSIX setting for system-wide scope is equivalent to the USYNC_PROCESS flag to `cond_init()` in the Solaris API. The object initialized with this attribute must be allocated in memory shared between processes, either in System V shared memory (see `shmop(2)`) or in memory mapped to a file (see `mmap(2)`). It is illegal to initialize
the object this way and to not allocate it in such shared memory.

Initializing condition variables can also be accomplished by allocating-in zerroed memory (default), in which case, `PTHREAD_PROCESS_PRIVATE` is assumed. The same condition variable must not be simultaneously initialized by multiple threads nor re-initialized while in use by other threads.

If default condition variable attributes are used, statically allocated condition variables can be initialized by the macro `PTHREAD_COND_INITIALIZER`. The effect is the same as a dynamic initialization by a call to `pthread_cond_init()` with parameter `attr` specified as NULL, except error checks are not performed.

Default condition variable initialization (intra-process):

```c
pthread_cond_t cvp;
pthread_condattr_t cv_attr;

pthread_cond_init(&cvp, NULL); /* initialize cv with defaults */
OR
pthread_condattr_init(&cv_attr); /* initialize cv_attr with defaults */
pthread_cond_init(&cvp, &cv_attr); /* initialize cv with default cv_attr */
OR
pthread_condattr_setpshared(&cv_attr, PTHREAD_PROCESS_PRIVATE);
pthread_cond_init(&cvp, &cv_attr); /* initialize cv with defaults */
OR
pthread_cond_t cond = PTHREAD_COND_INITIALIZER;
OR
pthread_cond_t cond;
cond = calloc(1, sizeof (pthread_cond_t));
```

Customized condition variable initialization (inter-process):

```c
pthread_condattr_init(&cv_attr); /* initialize cv_attr with defaults */
pthread_condattr_setpshared(&cv_attr, PTHREAD_PROCESS_SHARED);
pthread_cond_init(&cvp, &cv_attr); /* initialize cv with inter-process scope */
```

`Solaris Initialize cond_init()` initializes the condition variable pointed to by `cvp`. A condition variable can have several different types of behavior, specified by `type`. No current type uses `arg` although a future type may specify additional behavior parameters via `arg`. `type` may be one of the following:

- **USYNC_THREAD** The condition variable can synchronize threads only in this process. The `USYNC_THREAD` Solaris condition variable type for process scope is equivalent to the POSIX condition variable attribute setting `PTHREAD_PROCESS_PRIVATE`. `arg` is ignored.

- **USYNC_PROCESS** The condition variable can synchronize threads in this process and other processes. Only one process should initialize the condition variable.
variable. The USYNC_PROCESS Solaris condition variable type for system-wide scope is equivalent to the POSIX condition variable attribute setting PTHREAD_PROCESS_SHARED. arg is ignored. The object initialized with this attribute must be allocated in memory shared between processes, either in System V shared memory (see shmap(2)) or in memory mapped to a file (see mmap(2)). It is illegal to initialize the object this way and to not allocate it in such shared memory.

Initializing condition variables can also be accomplished by allocating in zeroed memory, in which case, a type of USYNC_THREAD is assumed.

If default condition variable attributes are used, statically allocated condition variables can be initialized by the macro DEFAULTCV.

Default condition variable initialization (intra-process):

```c
cond_t cvp;

cond_init(&cvp, NULL, NULL); /* initialize condition variable with default */

OR
cond_init(&cvp, USYNC_THREAD, NULL);

OR
cond_t cond = DEFAULTCV;
```

Customized condition variable initialization (inter-process):

```c
cond_init(&cvp, USYNC_PROCESS, NULL); /* initialize cv with inter-process scope */
```

Condition Wait

The condition wait interface allows a thread to wait for a condition and atomically release the associated mutex that it needs to hold to check the condition. The thread waits for another thread to make the condition true and that thread’s resulting call to signal and wakeup the waiting thread.

POSIX Wait

pthread_cond_wait() and pthread_cond_timedwait() block on a condition variable, which atomically release the mutex pointed to by mp and cause the calling thread to block on the condition variable pointed to by cond. The blocked thread may be awakened by pthread_cond_signal(), pthread_cond_broadcast(), or interrupted by a UNIX signal.

These functions atomically release the mutex, causing the calling thread to block on the condition variable cond.

Upon successful completion, the mutex is locked and owned by the calling thread. pthread_cond_timedwait() is the same as pthread_cond_wait(), except an error is returned if the system time equals or exceeds the time specified by abstime before the condition cond is signaled or broadcasted, or if the absolute time specified by abstime has already passed at the time of the call. When timeouts occur, pthread_cond_timedwait() releases and reacquires the mutex referenced by mutex.
When using condition variables, there is always a boolean predicate involving shared variables related to each condition wait that is true, if the thread should proceed. Since the return from `pthread_cond_wait()` or `pthread_cond_timedwait()` does not indicate anything about the value of this predicate, the predicate should be reevaluated on return. Unwanted wakeups from `pthread_cond_wait()` or `pthread_cond_timedwait()` may occur.

The functions `pthread_cond_wait()` and `pthread_cond_timedwait()` are cancellation points. If a cancellation request is acted upon while in a condition wait when the cancellation enable state of a thread is set to `PTHREAD_CANCEL_DEFERRED`, the mutex will be reacquired before calling the first cancellation cleanup handler. In other words, the thread is unblocked, allowed to execute up to the point of returning from the call to `pthread_cond_wait()` or `pthread_cond_timedwait()`, but then notices the cancellation request and, instead of returning to the caller of `pthread_cond_wait()` or `pthread_cond_timedwait()`, it starts the thread cancellation activities including cancellation cleanup handlers.

A thread that is unblocked because it was canceled while blocked in a call to `pthread_cond_wait()` or `pthread_cond_timedwait()` does not awaken anyone else asleep on the condition.

**Solaris Wait**

`cond_wait()` atomically releases the mutex pointed to by `mp` and causes the calling thread to block on the condition variable pointed to by `cvp`. The blocked thread may be awakened by `cond_signal()`, `cond_broadcast()`, or when interrupted by delivery of a UNIX signal or a `fork()`.

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

**Condition Signaling**

A condition signal allows a thread to unblock the next thread waiting on the condition variable, whereas, a condition broadcast allows a thread to unblock all threads waiting on the condition variable.

**POSIX Signal and Broadcast**

`pthread_cond_signal()` and `pthread_cond_broadcast()` unblock threads blocked on a condition variable.

`pthread_cond_signal()` unblocks at least one thread blocked on the specified condition variable `cond`, if any threads are blocked on `cond`.

`pthread_cond_broadcast()` unblocks all threads blocked on the condition variable `cond`.

`pthread_cond_signal()` and `pthread_cond_broadcast()` have no effect if there are no threads blocked on `cond`.

`pthread_cond_signal()` or `pthread_cond_broadcast()` may be called by a thread regardless of whether it owns the mutex which threads calling `pthread_cond_wait()` or `pthread_cond_timedwait()` have associated with the condition variable during their waits. However, if predictable scheduling behavior is required, then that mutex should be locked by the thread calling `pthread_cond_signal()` or `pthread_cond_broadcast()`.

modified 8 May 1997
**Solaris Signal and Broadcast**

- **cond_signal()** unblocks one thread that is blocked on the condition variable pointed to by *cvp*.
- **cond_broadcast()** unblocks all threads that are blocked on the condition variable pointed to by *cvp*.

If no threads are blocked on the condition variable, then **cond_signal()** and **cond_broadcast()** have no effect.

Both functions should be called under the protection of the same mutex that is used with the condition variable being signaled. Otherwise, the condition variable may be signaled between the test of the associated condition and blocking in **cond_wait()**. This can cause an infinite wait.

**Destroy**

The condition destroy functions destroy any state, but not the space, associated with the condition variable.

**POSIX Destroy**

- **pthread_cond_destroy()** destroys the condition variable specified by *cond*. The space for destroying the condition variable is not freed.

**Solaris Destroy**

- **cond_destroy()** destroys any state associated with the condition variable pointed to by *cvp*. The space for storing the condition variable is not freed.

**RETURN VALUES**

0 is returned when any of these functions are successful. A non-zero value indicates an error, except **pthread_timedwait()**, which returns **ETIME**.

**ERRORS**

These functions fail and return the corresponding value if any of the following conditions are detected:

- **EFAULT**
  - *cond*, *attr*, *cvp*, *arg*, *abstime*, or *mutex* point to an illegal address.

- **EINVAL**
  - Invalid argument.
  - For **pthread_cond_init()**, the value specified for *attr* is invalid.
  - For **cond_init()**, *type* is not a recognized type.
  - For **pthread_cond_timedwait()** or **cond_timedwait()**, the specified number of seconds, *abstime*, is greater than *current_time + 100,000,000*, where *current_time* is the current time, or the number of nanoseconds is greater than or equal to 1,000,000,000.

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

  - **EINVAL**
    - Invalid argument.

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

- **ETIME**
  - The time specified by *abstime* has passed.

- **pthread_cond_timedwait()** fails and returns the corresponding value if the following condition is detected:

  - **ETIMEDOUT**
    - The time specified by *abstime* has passed.
EXAMPLES

`pthread_cond_wait()` is normally used in a loop testing some condition, as follows:

```c
(void) pthread_mutex_lock(mp);
while (cond == FALSE) {
    (void) pthread_cond_wait(cvp, mp);
}
(void) pthread_mutex_unlock(mp);
```

`pthread_cond_timedwait()` is also normally used in a loop testing in some conditions. It uses an absolute timeout value as follows:

```c
timestruc_t to;
...
(void) pthread_mutex_lock(mp);
to.tv_sec = time(NULL) + TIMEOUT;
to.tv_nsec = 0;
while (cond == FALSE) {
    err = pthread_cond_timedwait(cvp, mp, &to);
    if (err == ETIMEDOUT) {
        /* timeout, do something */
        break;
    }
}
(void) pthread_mutex_unlock(mp);
```

The above example sets a bound on the total wait time even though `pthread_cond_timedwait()` may return several times due to the condition being signaled or the wait being interrupted.

Both of the above examples also apply to `cond_wait()` and `cond_timedwait()`, the Solaris versions of the API.

ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO

`fork(2)`, `mmap(2)`, `setitimer(2)`, `shmop(2)`, `mutex(3T)`, `pthread_condattr_init(3T)`, `signal(3C)`, `attributes(5)`, `standards(5)`

NOTES

The only policy currently supported is `SCHED_OTHER`. In Solaris, under the `SCHED_OTHER` policy, there is no established order in which threads are unblocked.

If more than one thread is blocked on a condition variable, the order in which threads are unblocked is determined by the scheduling policy. When each thread, unblocked as a result of a `pthread_cond_signal()` or `pthread_cond_broadcast()`, returns from its call to `pthread_cond_wait()` or `pthread_cond_timedwait()`, the thread owns the mutex with which it called `pthread_cond_wait()` or `pthread_cond_timedwait()`. The thread(s) that are unblocked compete for the mutex according to the scheduling policy, and as if each had called `pthread_mutex_lock(3T)`.

modified 8 May 1997  SunOS 5.6  3T-221
When \texttt{cond\_wait( )} returns the value of the condition is indeterminate and must be reevaluated.

\texttt{cond\_timedwait( )} is similar to \texttt{cond\_wait( )}, except that the calling thread will not wait for the condition to become true past the absolute time specified by \texttt{abstime}. Note that \texttt{cond\_timedwait( )} may continue to block as it tries to reacquire the mutex pointed to by \texttt{mp}, which may be locked by another thread. If \texttt{abstime} then \texttt{cond\_timedwait( )} returns because of a timeout, it returns the error code \texttt{ETIME}.
NAME

confstr – get configurable variables

SYNOPSIS

#include <unistd.h>

size_t confstr(int name, char *buf, size_t len);

DESCRIPTION

The confstr() function provides a method for applications to get configuration-defined string values. Its use and purpose are similar to the sysconf(3C) function, but it is used where string values rather than numeric values are returned.

The name argument represents the system variable to be queried. confstr() supports the following values for name, defined in <unistd.h>:

-CS_LFS_CFLAGS -CS_LFS64_CFLAGS
-CS_LFS_LDFLAGS -CS_LFS64_LDFLAGS
-CS_LFS_LIBS -CS_LFS64_LIBS
-CS_LFS_LINTFLAGS -CS_LFS64_LINTFLAGS
-CS_PATH

If len is not 0, and if name has a configuration-defined value, confstr() copies that value into the len-byte buffer pointed to by buf. If the string to be returned is longer than len bytes, including the terminating null, then confstr() truncates the string to len−1 bytes and null-terminates the result. The application can detect that the string was truncated by comparing the value returned by confstr() with len.

If len is 0, and buf is a null pointer, then confstr() still returns the integer value as defined below, but does not return the string. If len is 0 but buf is not a null pointer, the result is unspecified.

RETURN VALUES

If name has a configuration-defined value, the confstr() function returns the size of buffer that would be needed to hold the entire configuration-defined value. If this return value is greater than len, the string returned in buf is truncated.

If name is invalid, confstr() returns 0 and sets errno to indicate the error.

If name does not have a configuration-defined value, confstr() returns 0 and leaves errno unchanged.

ERRORS

The confstr() function will fail if:

EINVAL The value of the name argument is invalid.

ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Mt-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO

pathconf(2), sysconf(3C), attributes(5)

modified 29 Dec 1996 SunOS 5.6 3C-223
NAME  connect – initiate a connection on a socket

SYNOPSIS  cc [ flag ... ] file ... -lsocket -lnsl [ library ... ]
#include <sys/types.h>
#include <sys/socket.h>
int connect(int s, struct sockaddr *name, int namelen);

DESCRIPTION  The parameter s 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 s is of type SOCK_STREAM, connect() attempts to make a connection to another socket. The other socket is specified by name.
name is an address in the communication space of the socket. Each communication space
interprets the name parameter in its own way. If s 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.

RETURN VALUES  If the connection or binding succeeds, 0 is returned. Otherwise, –1 is returned and sets
errno to indicate the error.

ERRORS  The call fails if:

EACCES  Search permission is denied for a component of the path prefix of
the path name in name.

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  s is not a valid descriptor.

ECONNREFUSED  The attempt to connect was forcefully rejected. The calling pro-
gram should close(2) the socket descriptor, and issue another
socket(3N) call to obtain a new descriptor before attempting
another connect() call.
Network Functions

connect (3N)

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 namelen 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 name.

ENETUNREACH The network is not reachable from this host.

ENOENT A component of the path prefix of the pathname in name does not exist.

ENOENT The socket referred to by the pathname in name does not exist.

ENOSR There were insufficient STREAMS resources available to complete the operation.

ENXIO The server exited before the connection was complete.

ETIMEDOUT Connection establishment timed out without establishing a connection.

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 name is not a directory.

ENOTSOCK s is not a socket.

ENOTSOCK name is not a socket.

EPROTOTYPE The file referred to by name is a socket of a type other than type s (for example, s is a SOCK_DGRAM socket, while name refers to a SOCK_STREAM socket).

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

modified 16 May 1997 SunOS 5.6 3N-225
SEE ALSO close(2), accept(3N), getsockname(3N), select(3C), socket(3N), attributes(5), socket(5)
NAME  
connect – connect a socket

SYNOPSIS  
cc [ flag ...] file ... -lxnet [ library ...]
#include <sys/socket.h>
int connect(int socket, const struct sockaddr *address, size_t address_len);

DESCRIPTION  
The connect() function requests a connection to be made on a socket. The function takes the following arguments:
socket  
Specifies the file descriptor associated with the socket.
address  
Points to a sockaddr structure containing the peer address. The length and format of the address depend on the address family of the socket.
address_len  
Specifies the length of the sockaddr structure pointed to by the address argument.

If the initiating socket is not connection-mode, then connect() sets the socket’s peer address, but no connection is made. For SOCK_DGRAM sockets, the peer address identifies where all datagrams are sent on subsequent send(3XN) calls, and limits the remote sender for subsequent recv(3XN) calls. If address is a null address for the protocol, the socket’s peer address will be reset.

If the initiating socket is connection-mode, then connect() attempts to establish a connection to the address specified by the address argument.

If the connection cannot be established immediately and O_NONBLOCK is not set for the file descriptor for the socket, connect() will block for up to an unspecified timeout interval until the connection is established. If the timeout interval expires before the connection is established, connect() will fail and the connection attempt will be aborted. If connect() is interrupted by a signal that is caught while blocked waiting to establish a connection, connect() will fail and set errno to EINTR, but the connection request will not be aborted, and the connection will be established asynchronously.

If the connection cannot be established immediately and O_NONBLOCK is set for the file descriptor for the socket, connect() will fail and set errno to EINPROGRESS, but the connection request will not be aborted, and the connection will be established asynchronously. Subsequent calls to connect() for the same socket, before the connection is established, will fail and set errno to EAGAIN.

When the connection has been established asynchronously, select(3C) and poll(2) will indicate that the file descriptor for the socket is ready for writing.

RETURN VALUES  
Upon successful completion, connect() returns 0. Otherwise, −1 is returned and errno is set to indicate the error.

ERRORS  
The connect() function will fail if:
EADDRNOTAVAIL  
The specified address is not available from the local machine.

modified 16 May 1997  SunOS 5.6  3XN-227
<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EAFNOSUPPORT</td>
<td>The specified address is not a valid address for the address family of the specified socket.</td>
</tr>
<tr>
<td>EALREADY</td>
<td>A connection request is already in progress for the specified socket.</td>
</tr>
<tr>
<td>EBADF</td>
<td>The <code>socket</code> argument is not a valid file descriptor.</td>
</tr>
<tr>
<td>ECONNREFUSED</td>
<td>The target address was not listening for connections or refused the connection request.</td>
</tr>
<tr>
<td>EINPROGRESS</td>
<td><code>O_NONBLOCK</code> is set for the file descriptor for the socket and the connection cannot be immediately established; the connection will be established asynchronously.</td>
</tr>
<tr>
<td>EINTR</td>
<td>The attempt to establish a connection was interrupted by delivery of a signal that was caught; the connection will be established asynchronously.</td>
</tr>
<tr>
<td>EISCONN</td>
<td>The specified socket is connection-mode and is already connected.</td>
</tr>
<tr>
<td>ENETUNREACH</td>
<td>No route to the network is present.</td>
</tr>
<tr>
<td>ENOTSOCK</td>
<td>The <code>socket</code> argument does not refer to a socket.</td>
</tr>
<tr>
<td>EPROTOTYPE</td>
<td>The specified address has a different type than the socket bound to the specified peer address.</td>
</tr>
<tr>
<td>ETIMEDOUT</td>
<td>The attempt to connect timed out before a connection was made.</td>
</tr>
<tr>
<td>If the address family of the socket is AF_UNIX, then connect() will fail if:</td>
<td></td>
</tr>
<tr>
<td>ENOTDIR</td>
<td>A component of the path prefix of the pathname in <code>address</code> is not a directory.</td>
</tr>
<tr>
<td>ENAMETOOLONG</td>
<td>A component of a pathname exceeded <code>NAME_MAX</code> characters, or an entire pathname exceeded <code>PATH_MAX</code> characters.</td>
</tr>
<tr>
<td>EACCES</td>
<td>Search permission is denied for a component of the path prefix; or write access to the named socket is denied.</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>ELOOP</td>
<td>Too many symbolic links were encountered in translating the pathname in <code>address</code>.</td>
</tr>
<tr>
<td>ENOENT</td>
<td>A component of the pathname does not name an existing file or the pathname is an empty string.</td>
</tr>
</tbody>
</table>

The `connect()` function may fail if:

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EADDRINUSE</td>
<td>Attempt to establish a connection that uses addresses that are already in use.</td>
</tr>
<tr>
<td>ECONNRESET</td>
<td>Remote host reset the connection request.</td>
</tr>
<tr>
<td>EHOSTUNREACH</td>
<td></td>
</tr>
</tbody>
</table>
The destination host cannot be reached (probably because the host is down or a remote router cannot reach it).

**EINVAL**

The `address_len` argument is not a valid length for the address family; or invalid address family in sockaddr structure.

**ENAMETOOLONG**

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

**ENETDOWN**

The local interface used to reach the destination is down.

**ENOBUFS**

No buffer space is available.

**ENOSR**

There were insufficient STREAMS resources available to complete the operation.

**EOPNOTSUPP**

The socket is listening and cannot be connected.

**USAGE**

If `connect()` fails, the state of the socket is unspecified. Portable applications should close the file descriptor and create a new socket before attempting to reconnect.

**ATTRIBUTES**

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**SEE ALSO**

`accept(3XN), bind(3XN), close(2), getsockname(3XN), poll(2), select(3C), send(3XN), shutdown(3XN)), socket(3XN), attributes(5), socket(5)`

modified 16 May 1997
NAME copylist — copy a file into memory

SYNOPSIS cc [ flag ... ] file ... -lgen [ library ... ]
#include <libgen.h>
char *copylist(const char *filename, off_t *szptr);

DESCRIPTION The copylist() function copies a list of items from a file into freshly allocated memory, replacing new-lines with null characters. It expects two arguments: a pointer filename to the name of the file to be copied, and a pointer szptr 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.

USAGE The copylist() function has an explicit 64-bit equivalent. See interface64(5).

EXAMPLES /* 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);
}

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO malloc(3C), attributes(5), interface64(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

copysign – return magnitude of first argument and sign of second argument

SYNOPSIS

cc [ flag ... ] file ... -lm [ library ... ]

#include <math.h>

double copysign(double x, double y);

DESCRIPTION

The copysign() function returns a value with the magnitude of x and the sign of y. It produces a NaN with the sign of y if x is a NaN.

RETURN VALUES

The copysign() function returns a value with the magnitude of x and the sign of y.

ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO

attributes(5)
NAME  
copywin – overlay or overwrite any portion of window

SYNOPSIS  
#include <curses.h>

int copywin(WINDOW *srcwin, WINDOW *dstwin, int sminrow,
           int smincol, int dminrow, int dmincol,
           int dmaxrow, int dmaxcol, int overlay);

ARGUMENTS  
srcwin  Is a pointer to the source window to be copied.
dstwin  Is a pointer to the destination window to be overlayed or overwritten.
sminrow  Is the row coordinate of the upper left corner of the rectangular area on the
         source window to be copied.
smincol  Is the column coordinate of the upper left corner of the rectangular area on the
         source window to be copied.
dminrow  Is the row coordinate of the upper left corner of the rectangular area on the des-
         tination window to be overlayed or overwritten.
dmincol  Is the column coordinate of the upper left corner of the rectangular area on dest-
         ination window to be overlayed or overwritten.
dmaxrow  Is the row coordinate of the lower right corner of the rectangular area on the
         destination window to be overlayed or overwritten.
dmaxcol  Is the column coordinate of the lower right corner of the rectangular area on the
         destination window to be overlayed or overwritten.
overlay  Is a true or false value that determines whether the destination window is over-
          layed or overwritten.

DESCRIPTION  The copywin() function overlays or overwrites windows similar to the overlay(3XC) and
overwrite(3XC) functions; however, copywin() allows a finer degree of control on what
portion of the window to overlay or overwrite.

The parameters smincol and sminrow specify the upper left corner of the rectangular area
of the source window to be copied. The dminrow and dmincol parameters specify the
upper left corner of the rectangular area of the destination window to which the specified
portion of the source is to be copied. The dmaxrow and dmaxcol parameters specify the
bottom right corner of the rectangular area of the destination window to which the
specified portion of the source is to be copied.

If overlay is TRUE, only non-blank characters are copied to the destination window; if it is
FALSE, all characters are copied.

For details on how this function handles overlapping windows with multicolumn charac-
ters, see the Overlapping Windows section of the curses(3XC) man page.
<table>
<thead>
<tr>
<th>RETURN VALUES</th>
<th>On success, the <code>copywin()</code> function returns <strong>OK</strong>. Otherwise, it returns <strong>ERR</strong>.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERRORS</td>
<td>None.</td>
</tr>
<tr>
<td>SEE ALSO</td>
<td><code>curses_over(3XC)</code>, <code>curses(3XC)</code>, <code>newpad(3XC)</code>, <code>overlay(3XC)</code></td>
</tr>
</tbody>
</table>
NAME  

cos – cosine function

SYNOPSIS  

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

DESCRIPTION  
The cos() function computes the cosine of x, measured in radians.

RETURN VALUES  
Upon successful completion, cos() returns the cosine of x.
If x is NaN or ±Inf, NaN is returned.

ERRORS  
No errors will occur.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  
acos(3M), isnan(3M), sin(3M), tan(3M), attributes(5)
NAME  \texttt{cosh} – hyperbolic cosine function

SYNOPSIS  \texttt{cc [ flag … ] file … –lm [ library … ]} 
           \texttt{#include <math.h>} 
           \texttt{double cosh(double \textit{x});}

DESCRIPTION  The \texttt{cosh()} function computes the hyperbolic cosine of \textit{x}.

RETURN VALUES  Upon successful completion, \texttt{cosh()} returns the hyperbolic cosine of \textit{x}. 
    If the result would cause an overflow, \texttt{HUGE\_VAL} is returned and \texttt{errno} is set to \texttt{ERANGE}. 
    If \textit{x} is NaN, NaN is returned. 
    For exceptional cases, \texttt{matherr}(3M) tabulates the values to be returned as dictated by Standards other than XPG4.

ERRORS  The \texttt{cosh()} function will fail if: 
    \textbf{ERANGE}  The result would cause an overflow.

USAGE  An application wishing to check for error situations should set \texttt{errno} to 0 before calling \texttt{cosh()}. If \texttt{errno} is non-zero on return, or the returned value is NaN, an error has occurred.

ATTRIBUTES  See \texttt{attributes}(5) for descriptions of the following attributes:

\begin{tabular}{|c|c|}
\hline
ATTRIBUTE TYPE & ATTRIBUTE VALUE \\
\hline
MT-Level & MT-Safe \\
\hline
\end{tabular}

SEE ALSO  \texttt{acosh}(3M), \texttt{isnan}(3M), \texttt{matherr}(3M), \texttt{sinh}(3M), \texttt{tanh}(3M), \texttt{attributes}(5), \texttt{standards}(5)

modified 29 Dec 1996  SunOS 5.6  3M-235
NAME crypt – string encoding function

SYNOPSIS #include <unistd.h>

char *crypt (const char *key, const char *salt);

DESCRIPTION The crypt() function is a string encoding function, used primarily for password encryption. It is based on a one-way encryption algorithm with variations intended (among other things) to frustrate use of hardware implementations of a key search.

The key argument points to a string to be encoded (for example, the user’s password.) Only the first eight characters are used; the rest are ignored. The salt 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.

RETURN VALUES Upon successful completion, crypt() returns a pointer to the encoded string. The first two characters of the returned value are those of the salt argument. Otherwise it returns a null pointer and sets errno to indicate the error.

ERRORS The crypt() function will fail if:

ENOSYS The functionality is not supported on this implementation.

USAGE The return value of crypt() points to static data that is overwritten by each call. The values returned by this function may not be portable among XSI-conformant systems.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO passwd(1), crypt(3X), encrypt(3C), getpass(3C), setkey(3C), passwd(4), attributes(5)

NOTES In the case of multithreaded applications, the return value is a pointer to thread specific data.
NAME  cset, csetlen, csetcol, csetno, wcsetno – get information on EUC codesets

SYNOPSIS  
#include <euc.h>
int csetlen(int codeset);
int csetcol(int codeset);
int csetno(unsigned char c);
#include <widec.h>
int wcsetno(wchar_t pc);

DESCRIPTION  
Both csetlen() and csetcol() take a code set number codeset, 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.

csetno() is a macro that returns a codeset number (0, 1, 2, or 3) for the EUC character whose first byte is c. For example,
#include<euc.h>

x+=csetcol(csetno(c));

 increments a counter “x” (such as the cursor position) by the width of the character whose first byte is c.

wcsetno() is a macro that returns a codeset number (0, 1, 2, or 3) for the given process code character pc. For example,

#include<euc.h>
#include<widec.h>

x+=csetcol(wcsetno(pc));

 increments a counter “x” (such as the cursor position) by the width of the Process Code character pc.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe with exceptions</td>
</tr>
</tbody>
</table>

SEE ALSO  setlocale(3C), euclen(3C), attributes(5)

NOTES  
cset, csetlen, csetcol, csetno and wcsetno can be used safely in a multi-thread application, as long as setlocale(3C) is not being called to change the locale.
ctermid(3S) Standard I/O Functions

NAME ctermid, ctermid_r – generate path name for controlling terminal

SYNOPSIS
#include <stdio.h>

char *ctermid(char *s);

char *ctermid_r(char *s);

DESCRIPTION ctermid( ) generates the path name of the controlling terminal for the current process, and stores it in a string. If s 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, s 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 s is returned. The constant L_ctermid is defined in the header <stdio.h>.

ctermid_r( ) has the same functionality as ctermid( ) except that if s is a NULL pointer, the function returns NULL.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>See NOTES below.</td>
</tr>
</tbody>
</table>

SEE ALSO ttyname(3C), attributes(5)

NOTES 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.

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.

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.

ctermid( ) is unsafe in multithreaded applications. ctermid_r( ) is MT-Safe, and should be used instead.
NAME      ctime, ctime_r, localtime, localtime_r, gmtime, gmtime_r, asctime, asctime_r, tzset, tzsetwall – convert date and time to string

SYNOPSIS  #include <time.h>
char *ctime(const time_t *clock);
struct tm *localtime(const time_t *clock);
struct tm *gmtime(const time_t *clock);
char *asctime(const struct tm *tm);
extern time_t timezone, altzone;
extern int daylight;
extern char *tzname[2];
void tzset(void);
void tzsetwall(void);
char *ctime_r(const time_t *clock, char *buf, int buflen);
struct tm *localtime_r(const time_t *clock, struct tm *res);
struct tm *gmtime_r(const time_t *clock, struct tm *res);
char *asctime_r(const struct tm *tm, char *buf);

DESCRIPTION The ctime(), localtime(), and gmtime() functions accept arguments of type time_t, pointed to by clock(), representing the time in seconds since 00:00:00 UTC, January 1, 1970. The ctime() function returns a pointer to a 26-character string as shown below. Time zone and daylight savings corrections are made before string generation. The fields are constant width:
Fri Sep 13 00:00:00 1986
The ctime_r() function has the same functionality as ctime() except that the caller must supply a buffer buf with length buflen to store the result; buf must be at least 26 bytes. The POSIX ctime_r() function does not take a buflen parameter.

The localtime() and gmtime() functions return pointers to tm structures (see below). The localtime() function corrects for the main time zone and possible alternate (“daylight savings”) time zone; the gmtime() function converts directly to Coordinated Universal Time (UTC), which is what the UNIX system uses internally.

The localtime_r() and gmtime_r() functions have the same functionality as localtime() and gmtime() respectively, except that the caller must supply a buffer res to store the result.
The `asctime()` function converts a `tm` structure to a 26-character string, as shown in the above example, and returns a pointer to the string.

The `asctime_r()` function has the same functionality as `asctime()` except that the caller must supply a buffer `buf` with length `buflen` for the result to be stored. The `buf` argument must be at least 26 bytes. The POSIX `asctime_r()` function does not take a `buflen` parameter. The `asctime_r()` function returns a pointer to `buf` upon success. In case of failure, `NULL` is returned and `errno` is set.

Declarations of all the functions and externals, and the `tm` structure, are in the `time.h` header. The members of the `tm` structure are:

```c
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 alternate daylight savings time */
```

The value of `tm_isdst` is positive if daylight savings time is in effect, zero if daylight savings time is not in effect, and negative if the information is not available. (Previously, the value of `tm_isdst` was defined as non-zero if daylight savings was in effect.)

The external `time_t` variable `altzone` contains the difference, in seconds, between Coordinated Universal Time and the alternate time zone. The external variable `timezone` contains the difference, in seconds, between UTC and local standard time. The external variable `daylight` indicates whether time should reflect daylight savings time. Both `timezone` and `altzone` default to 0 (UTC). The external variable `daylight` is non-zero if an alternate time zone exists. The time zone names are contained in the external variable `tzname`, which by default is set to:

```c
char *tzname[2] = { "GMT", "" }
```

These functions know about the peculiarities of this conversion for various time periods for the U.S. (specifically, the years 1974, 1975, and 1987). They will handle the new daylight savings time starting with the first Sunday in April, 1987.

The `tzset()` function uses the contents of the environment variable `TZ` to override the value of the different external variables. It is called by `asctime()` and may also be called by the user. See `environ(5)` for a description of the `TZ` environment variable.

Starting and ending times are relative to the current local time zone. If the alternate time zone start and end dates and the time are not provided, the days for the United States that year will be used and the time will be 2 AM. If the start and end dates are provided but the time is not provided, the time will be 2 AM. The effects of `tzset()` change the values of the external variables `timezone`, `altzone`, `daylight`, and `tzname`.
Note that in most installations, TZ is set to the correct value by default when the user logs on, using the local /etc/default/init file (see TIMEZONE(4)).

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

ERRORS

The ctime_r() and asctime_r() functions will fail if the following is true:

ERANGE The length of the buffer supplied by the caller is not large enough to store the result.

USAGE

These functions are included for compatibility with older implementations, and do not support localized date and time formats.

EXAMPLES

The tzset() function scans the contents of the environment variable and assigns the different fields to the respective variable. For example, the most complete setting for New Jersey in 1986 could be

EST5EDT4,116/2:00:00,298/2:00:00

or simply

EST5EDT

An example of a southern hemisphere setting such as the Cook Islands could be

KDT9:30KST10:00,63/5:00,302/20:00

In the longer version of the New Jersey example of TZ, tzname[0] is EST, timezone will be set to 5*60*60, tzname[1] is EDT, altzone will be set to 4*60*60, the starting date of the alternate time zone is the 117th day at 2 AM, the ending date of the alternate time zone is the 299th day at 2 AM (using zero-based Julian days), and daylight will be set positive. Starting and ending times are relative to the current local time zone. If the alternate time zone start and end dates and the time are not provided, the days for the United States that year will be used and the time will be 2 AM. If the start and end dates are provided but the time is not provided, the time will be 2 AM. The effects of tzset() are thus to change the values of the external variables timezone, altzone, daylight, and tzname. The ctime(), localtime(), mktime(), and strftime() functions will also update these external variables as if they had called tzset() at the time specified by the time_t or struct tm value that they are converting.

ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe with exceptions</td>
</tr>
<tr>
<td>CSI</td>
<td>Enabled</td>
</tr>
</tbody>
</table>

SEE ALSO  time(2), Intro(3), getenv(3C), mktime(3C), printf(3S), putenv(3C), setlocale(3C), strftime(3C), TIMEZONE(4), attributes(5), environ(5)

modified 19 May 1997  SunOS 5.6  3C-241
NOTES

When compiling multithread programs, see Intro(3), Notes On Multithread Applications. The return values for ctime(), localtime(), and gmtime() point to static data whose content is overwritten by each call.

Setting the time during the interval of change from timezone to altzone or vice versa can produce unpredictable results. The system administrator must change the Julian start and end days annually.

The asctime(), ctime(), gmtime(), and localtime() functions are unsafe in multithread applications. The asctime_r() and gmtime_r() functions are MT-Safe. The ctime_r(), localtime_r(), tzset(), and tzsetwall() functions are MT-Safe in multithread applications, as long as no user-defined function directly modifies one of the following variables: timezone, altzone, daylight, and tzname. These four variables are not MT-Safe to access. They are modified by the tzset() function in an MT-Safe manner. The mktime(), localtime_r(), and ctime_r() functions call tzset().

Solaris 2.4 and earlier releases provided definitions of the ctime_r(), localtime_r(), gmtime_r(), and asctime_r() functions as specified in POSIX.1c Draft 6. The final POSIX.1c standard changed the interface for ctime_r() and asctime_r(). Support for the Draft 6 interface is provided for compatibility only and may not be supported in future releases. New applications and libraries should use the POSIX standard interface.

For POSIX.1c complaint applications, the _POSIX_PTHREAD_SEMANTICS and _REENTRANT flags are automatically turned on by defining the _POSIX_C_SOURCE flag with a value >= 199506L.
NAME
ctype, isdigit, isxdigit, islower, isupper, isalpha, isalnum, isspace, iscntrl, ispunct, isprint, isgraph, isascii – character handling

SYNOPSIS
#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);

DESCRIPTION
These macros classify character-coded integer values. Each is a predicate returning non-zero for true, 0 for false. The behavior of these macros, except isascii(), is affected by the current locale (see setlocale(3C)). To modify the behavior, change the 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.

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.

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

For macros described with Default and Standard-conforming versions, standard-conforming behavior will be provided for standard-conforming applications (see standards(5)) and for applications that define __XPG4_CHAR_CLASS__ before including <ctype.h>.

<table>
<thead>
<tr>
<th>Default</th>
<th>Standard-conforming</th>
</tr>
</thead>
<tbody>
<tr>
<td>isalpha()</td>
<td>isalpha()</td>
</tr>
<tr>
<td>tests for any character for which isupper() or islower() is true.</td>
<td></td>
</tr>
<tr>
<td>tests for any character for which isupper() or islower() is true, or any character that is one of the current locale-defined set of characters for which none of iscntrl(), isdigit(), ispunct(), or isspace() is true. In C locale, isalpha() returns true only for the characters for which isupper() or islower() is true.</td>
<td></td>
</tr>
</tbody>
</table>

modified 26 Feb 1997
SunOS 5.6
3C-243
isupper() tests for any character that is an upper-case letter or is one of the current locale-defined set of characters for which none of iscntrl(), isdigit(), ispunct(), isspace(), or islower() is true. In the C locale, isupper() returns true only for the characters defined as upper-case ASCII characters.

islower() tests for any character that is a lower-case letter or is one of the current locale-defined set of characters for which none of iscntrl(), isdigit(), ispunct(), isspace(), or isupper() is true. In the C locale, islower() returns true only for the characters defined as lower-case ASCII characters.

isdigit() tests for any decimal-digit character.

Default isxdigit() tests for any hexadecimal-digit character ([0–9], [A–F], or [a–f]).

Standard-conforming isxdigit() tests for any hexadecimal-digit character ([0–9], [A–F], or [a–f] or the current locale-defined sets of characters representing the hexadecimal digits 10 to 15 inclusive). In the C locale, only

\[
0 1 2 3 4 5 6 7 8 9 A B C D E F a b c d e f
\]

are included.

isalnum() tests for any character for which isalpha() or isdigit() is true (letter or digit).

isspace() tests for any space, tab, carriage-return, newline, vertical-tab or form-feed (standard white-space characters) or for one of the current locale-defined set of characters for which isalnum() is false. In the C locale, isspace() returns true only for the standard white-space characters.

ispunct() tests for any printing character which is neither a space (" ") nor a character for which isalnum() or iscntrl() is true.

Default isprint() tests for any character for which ispunct(), isupper(), islower(), isdigit(), and the space character (" ") is true.

Standard-conforming isprint() tests for any character for which iscntrl() is false, and isalnum(), isgraph(), ispunct(), the space character (" "), and the characters in the current locale-defined "print" class are true.

Default isgraph() tests for any character for which ispunct(), isupper(), islower(), and isdigit() is true.

Standard-conforming isgraph() tests for any character for which isalnum() and ispunct() are true, or any character in the current locale-defined "graph" class which is neither a space (" ") nor a character for which iscntrl() is true.

iscntrl() tests for any “control character” as defined by the character set.

isascii() tests for any ASCII character, code between 0 and 0177 inclusive.
RETURN VALUES

If the argument to any of the character handling macros is not in the domain of the function, the result is undefined. Otherwise, the macro/function will return non-zero if the classification is TRUE, and 0 for FALSE.

FILES

/usr/lib/locale/locale/LC_CTYPE

ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe with exceptions</td>
</tr>
<tr>
<td>CSI</td>
<td>Enabled</td>
</tr>
</tbody>
</table>

SEE ALSO

setlocale(3C), stdio(3S), ascii(5), environ(5), standards(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.

modified 26 Feb 1997

SunOS 5.6
NAME curses – CRT screen handling and optimization package

SYNOPSIS cc [ flag ... ] file ... -lcurses [ library ... ]
#include <curses.h>

DESCRIPTION The curses library routines give the user a terminal-independent method of updating character screens with reasonable optimization.

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.

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:

    initscr,cbreak,noecho;

Most programs would additionally use the sequence:

    nonl,intrflush(stdscr,FALSE),keypad(stdscr,TRUE);

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.)

The curses library permits manipulation of data structures, called windows, 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).

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.

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.

Special windows called pads may also be manipulated. These are windows which are not constrained to the size of the screen and whose contents need not be completely displayed. See curs_pad(3X) for more information.
In addition to drawing characters on the screen, video attributes and colors may be included, causing the characters to show up in such modes as underlined, in reverse video, or in color on terminals that support such display enhancements. Line drawing characters may be specified to be output. On input, curses is also able to translate arrow and function keys that transmit escape sequences into single values. The video attributes, line drawing characters, and input values use names, defined in <curses.h>, such as A_REVERSE, ACS_HLINE, and KEY_LEFT.

If the environment variables LINES and COLUMNS are set, or if the program is executing in a window environment, line and column information in the environment will override information read by terminfo. This would effect a program running in an AT&T 630 layer, for example, where the size of a screen is changeable.

If the environment variable TERMINFO is defined, any program using curses checks for a local terminal definition before checking in the standard place. For example, if TERM is set to att4424, then the compiled terminal definition is found in

```
/usr/share/lib/terminfo/a/att4424.
```

(The ‘a’ is copied from the first letter of att4424 to avoid creation of huge directories.) However, if TERMINFO is set to $HOME/myterms, curses first checks

```
$HOME/myterms/a/att4424,
```

and if that fails, it then checks

```
/usr/share/lib/terminfo/a/att4424.
```

This is useful for developing experimental definitions or when write permission in /usr/share/lib/terminfo is not available.

The integer variables LINES and COLS are defined in <curses.h> and will be filled in by initscr with the size of the screen. The constants TRUE and FALSE have the values 1 and 0, respectively.

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.
The following table lists each `curses` routine and the name of the manual page on which it is described.

<table>
<thead>
<tr>
<th>curses Routine Name</th>
<th>Manual Page Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>addch</td>
<td>curs_addch(3X)</td>
</tr>
<tr>
<td>addchnstr</td>
<td>curs_addchstr(3X)</td>
</tr>
<tr>
<td>addchstr</td>
<td>curs_addchstr(3X)</td>
</tr>
<tr>
<td>addnstr</td>
<td>curs_addstr(3X)</td>
</tr>
<tr>
<td>addnwstr</td>
<td>curs_addwstr(3X)</td>
</tr>
<tr>
<td>addstr</td>
<td>curs_addstr(3X)</td>
</tr>
<tr>
<td>addwch</td>
<td>curs_addwch(3X)</td>
</tr>
<tr>
<td>addwchnstr</td>
<td>curs_addwchstr(3X)</td>
</tr>
<tr>
<td>addwchstr</td>
<td>curs_addwchstr(3X)</td>
</tr>
<tr>
<td>addwstr</td>
<td>curs_addwstr(3X)</td>
</tr>
<tr>
<td>adjcurspos</td>
<td>curs_alecompat(3X)</td>
</tr>
<tr>
<td>attroff</td>
<td>curs_attr(3X)</td>
</tr>
<tr>
<td>attron</td>
<td>curs_attr(3X)</td>
</tr>
<tr>
<td>attrset</td>
<td>curs_attr(3X)</td>
</tr>
<tr>
<td>baudrate</td>
<td>curs_termattrs(3X)</td>
</tr>
<tr>
<td>beep</td>
<td>curs_beep(3X)</td>
</tr>
<tr>
<td>bkgd</td>
<td>curs_bkgd(3X)</td>
</tr>
<tr>
<td>bkgdset</td>
<td>curs_bkgd(3X)</td>
</tr>
<tr>
<td>border</td>
<td>curs_border(3X)</td>
</tr>
<tr>
<td>box</td>
<td>curs_border(3X)</td>
</tr>
<tr>
<td>can_change_color</td>
<td>curs_color(3X)</td>
</tr>
<tr>
<td>cbreak</td>
<td>curs_inopts(3X)</td>
</tr>
<tr>
<td>clear</td>
<td>curs_clear(3X)</td>
</tr>
<tr>
<td>clearok</td>
<td>curs_outopts(3X)</td>
</tr>
<tr>
<td>clrtoeol</td>
<td>curs_clear(3X)</td>
</tr>
<tr>
<td>color_content</td>
<td>curs_color(3X)</td>
</tr>
<tr>
<td>copywin</td>
<td>curs_overlay(3X)</td>
</tr>
<tr>
<td>curs_set</td>
<td>curs_kernel(3X)</td>
</tr>
<tr>
<td>def_prog_mode</td>
<td>curs_kernel(3X)</td>
</tr>
<tr>
<td>def_shell_mode</td>
<td>curs_kernel(3X)</td>
</tr>
<tr>
<td>del_curterm</td>
<td>curs_terminfo(3X)</td>
</tr>
<tr>
<td>delay_output</td>
<td>curs_util(3X)</td>
</tr>
<tr>
<td>delch</td>
<td>curs_delch(3X)</td>
</tr>
<tr>
<td>deleteln</td>
<td>curs_deleteln(3X)</td>
</tr>
<tr>
<td>delscreen</td>
<td>curs_initscr(3X)</td>
</tr>
<tr>
<td>delwin</td>
<td>curs_window(3X)</td>
</tr>
<tr>
<td>derwin</td>
<td>curs_window(3X)</td>
</tr>
<tr>
<td>dupodate</td>
<td>curs_refresh(3X)</td>
</tr>
<tr>
<td>dupwin</td>
<td>curs_window(3X)</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>echo</td>
<td>curs_inopts(3X)</td>
</tr>
<tr>
<td>echochar</td>
<td>curs_addch(3X)</td>
</tr>
<tr>
<td>echowchar</td>
<td>curs_addwch(3X)</td>
</tr>
<tr>
<td>endwin</td>
<td>curs_initscr(3X)</td>
</tr>
<tr>
<td>erase</td>
<td>curs_clear(3X)</td>
</tr>
<tr>
<td>erasechar</td>
<td>curs_termattrs(3X)</td>
</tr>
<tr>
<td>filter</td>
<td>curs_util(3X)</td>
</tr>
<tr>
<td>flash</td>
<td>curs_bEEP(3X)</td>
</tr>
<tr>
<td>flushinp</td>
<td>curs_util(3X)</td>
</tr>
<tr>
<td>getbegyx</td>
<td>curs_getyx(3X)</td>
</tr>
<tr>
<td>getch</td>
<td>curs_getch(3X)</td>
</tr>
<tr>
<td>getmaxyx</td>
<td>curs_getyx(3X)</td>
</tr>
<tr>
<td>getnwstr</td>
<td>curs_getwstr(3X)</td>
</tr>
<tr>
<td>getparyx</td>
<td>curs_getyx(3X)</td>
</tr>
<tr>
<td>getstr</td>
<td>curs_getstr(3X)</td>
</tr>
<tr>
<td>getsyx</td>
<td>curs_kernel(3X)</td>
</tr>
<tr>
<td>getch</td>
<td>curs_getwch(3X)</td>
</tr>
<tr>
<td>getwin</td>
<td>curs_util(3X)</td>
</tr>
<tr>
<td>getwstr</td>
<td>curs_getwstr(3X)</td>
</tr>
<tr>
<td>getyx</td>
<td>curs_getyx(3X)</td>
</tr>
<tr>
<td>halfdelay</td>
<td>curs_inopts(3X)</td>
</tr>
<tr>
<td>has_colors</td>
<td>curs_color(3X)</td>
</tr>
<tr>
<td>has_ic</td>
<td>curs_termattrs(3X)</td>
</tr>
<tr>
<td>has_il</td>
<td>curs_termattrs(3X)</td>
</tr>
<tr>
<td>idcok</td>
<td>curs_outopts(3X)</td>
</tr>
<tr>
<td>idlok</td>
<td>curs_outopts(3X)</td>
</tr>
<tr>
<td>immedok</td>
<td>curs_outopts(3X)</td>
</tr>
<tr>
<td>inch</td>
<td>curs_inch(3X)</td>
</tr>
<tr>
<td>inchstr</td>
<td>curs_inchstr(3X)</td>
</tr>
<tr>
<td>inchstr</td>
<td>curs_inchstr(3X)</td>
</tr>
<tr>
<td>init_color</td>
<td>curs_color(3X)</td>
</tr>
<tr>
<td>init_pair</td>
<td>curs_color(3X)</td>
</tr>
<tr>
<td>initscr</td>
<td>curs_initscr(3X)</td>
</tr>
<tr>
<td>innstr</td>
<td>curs_instr(3X)</td>
</tr>
<tr>
<td>innwstr</td>
<td>curs_inwstr(3X)</td>
</tr>
<tr>
<td>insch</td>
<td>curs_insch(3X)</td>
</tr>
<tr>
<td>insdelln</td>
<td>curs_deleteln(3X)</td>
</tr>
<tr>
<td>insertln</td>
<td>curs_deleteln(3X)</td>
</tr>
<tr>
<td>insnstr</td>
<td>curs_insstr(3X)</td>
</tr>
<tr>
<td>insnwstr</td>
<td>curs_inswstr(3X)</td>
</tr>
<tr>
<td>insstr</td>
<td>curs_insstr(3X)</td>
</tr>
<tr>
<td>instr</td>
<td>curs_instr(3X)</td>
</tr>
<tr>
<td>inswch</td>
<td>curs_inswch(3X)</td>
</tr>
<tr>
<td>inswstr</td>
<td>curs_inswstr(3X)</td>
</tr>
<tr>
<td>intrflush</td>
<td>curs_inopts(3X)</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>inwch</td>
<td>curs_inwch(3X)</td>
</tr>
<tr>
<td>inwchnstr</td>
<td>curs_inwchnstr(3X)</td>
</tr>
<tr>
<td>inwchstr</td>
<td>curs_inwchstr(3X)</td>
</tr>
<tr>
<td>inwstr</td>
<td>curs_inwstr(3X)</td>
</tr>
<tr>
<td>is_linetouched</td>
<td>curs_touch(3X)</td>
</tr>
<tr>
<td>is_wintouched</td>
<td>curs_touch(3X)</td>
</tr>
<tr>
<td>isendwin</td>
<td>curs_initscr(3X)</td>
</tr>
<tr>
<td>keyname</td>
<td>curs_util(3X)</td>
</tr>
<tr>
<td>keypad</td>
<td>curs_inopts(3X)</td>
</tr>
<tr>
<td>killchar</td>
<td>curs_termattrs(3X)</td>
</tr>
<tr>
<td>leaveok</td>
<td>curs_outtermattr(3X)</td>
</tr>
<tr>
<td>longname</td>
<td>curs_termattr(3X)</td>
</tr>
<tr>
<td>meta</td>
<td>curs_inopts(3X)</td>
</tr>
<tr>
<td>move</td>
<td>curs_move(3X)</td>
</tr>
<tr>
<td>movenextch</td>
<td>curs_alecompat(3X)</td>
</tr>
<tr>
<td>moveprevch</td>
<td>curs_alecompat(3X)</td>
</tr>
<tr>
<td>mvaddch</td>
<td>curs_addch(3X)</td>
</tr>
<tr>
<td>mvaddchstr</td>
<td>curs_addchstr(3X)</td>
</tr>
<tr>
<td>mvaddchstr</td>
<td>curs_addchstr(3X)</td>
</tr>
<tr>
<td>mvaddnstr</td>
<td>curs_addstr(3X)</td>
</tr>
<tr>
<td>mvaddnwstr</td>
<td>curs_addnwstr(3X)</td>
</tr>
<tr>
<td>mvaddstr</td>
<td>curs_addstr(3X)</td>
</tr>
<tr>
<td>mvaddwch</td>
<td>curs_addwch(3X)</td>
</tr>
<tr>
<td>mvaddwchstr</td>
<td>curs_addwchstr(3X)</td>
</tr>
<tr>
<td>mvaddwchstr</td>
<td>curs_addwchstr(3X)</td>
</tr>
<tr>
<td>mvaddwstr</td>
<td>curs_addwstr(3X)</td>
</tr>
<tr>
<td>mvcur</td>
<td>curs_terminfo(3X)</td>
</tr>
<tr>
<td>mvdelch</td>
<td>curs_delch(3X)</td>
</tr>
<tr>
<td>mvderwin</td>
<td>curs_window(3X)</td>
</tr>
<tr>
<td>mvgetch</td>
<td>curs_getch(3X)</td>
</tr>
<tr>
<td>mvgetnwstr</td>
<td>curs_getnwstr(3X)</td>
</tr>
<tr>
<td>mvgetstr</td>
<td>curs_getstr(3X)</td>
</tr>
<tr>
<td>mvgetwch</td>
<td>curs_getwch(3X)</td>
</tr>
<tr>
<td>mvgetwstr</td>
<td>curs_getwstr(3X)</td>
</tr>
<tr>
<td>mvinch</td>
<td>curs_inch(3X)</td>
</tr>
<tr>
<td>mvinchnstr</td>
<td>curs_inchnstr(3X)</td>
</tr>
<tr>
<td>mvinchstr</td>
<td>curs_inchstr(3X)</td>
</tr>
<tr>
<td>mvinnstr</td>
<td>curs_instr(3X)</td>
</tr>
<tr>
<td>mvinnwstr</td>
<td>curs_innwstr(3X)</td>
</tr>
<tr>
<td>mvinsch</td>
<td>curs_insch(3X)</td>
</tr>
<tr>
<td>mvinsnstr</td>
<td>curs_insstr(3X)</td>
</tr>
<tr>
<td>mvinsnwstr</td>
<td>curs_insnwstr(3X)</td>
</tr>
<tr>
<td>mvinsstr</td>
<td>curs_instr(3X)</td>
</tr>
<tr>
<td>mvinswch</td>
<td>curs_inswwch(3X)</td>
</tr>
</tbody>
</table>

modified 31 Dec 1996 SunOS 5.6 3X-251
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mvinswstr</td>
<td>curs_inswstr(3X)</td>
</tr>
<tr>
<td>mvinwch</td>
<td>curs_inwch(3X)</td>
</tr>
<tr>
<td>mvinwchnstr</td>
<td>curs_inwchstr(3X)</td>
</tr>
<tr>
<td>mvinwchstr</td>
<td>curs_inwchstr(3X)</td>
</tr>
<tr>
<td>mvinsstr</td>
<td>curs_inwstr(3X)</td>
</tr>
<tr>
<td>mvprintw</td>
<td>curs_printw(3X)</td>
</tr>
<tr>
<td>mvscanw</td>
<td>curs_scanw(3X)</td>
</tr>
<tr>
<td>mvwaddch</td>
<td>curs_addch(3X)</td>
</tr>
<tr>
<td>mvwaddchnstr</td>
<td>curs_addchstr(3X)</td>
</tr>
<tr>
<td>mvwaddchstr</td>
<td>curs_addchstr(3X)</td>
</tr>
<tr>
<td>mvwaddstr</td>
<td>curs_addstr(3X)</td>
</tr>
<tr>
<td>mvwaddnwstr</td>
<td>curs_addwstr(3X)</td>
</tr>
<tr>
<td>mvwaddstr</td>
<td>curs_addstr(3X)</td>
</tr>
<tr>
<td>mvwaddwch</td>
<td>curs_addwch(3X)</td>
</tr>
<tr>
<td>mvwaddwchnstr</td>
<td>curs_addwchstr(3X)</td>
</tr>
<tr>
<td>mvwaddwchstr</td>
<td>curs_addwchstr(3X)</td>
</tr>
<tr>
<td>mvwaddwstr</td>
<td>curs_addwstr(3X)</td>
</tr>
<tr>
<td>mvwdelch</td>
<td>curs_delch(3X)</td>
</tr>
<tr>
<td>mvwgetch</td>
<td>curs_getch(3X)</td>
</tr>
<tr>
<td>mvwgetnwstr</td>
<td>curs_getwstr(3X)</td>
</tr>
<tr>
<td>mvwgetstr</td>
<td>curs_getstr(3X)</td>
</tr>
<tr>
<td>mvwgetwch</td>
<td>curs_getwch(3X)</td>
</tr>
<tr>
<td>mvwgetwstr</td>
<td>curs_getwstr(3X)</td>
</tr>
<tr>
<td>mvwin</td>
<td>curs_window(3X)</td>
</tr>
<tr>
<td>mvwinch</td>
<td>curs_inch(3X)</td>
</tr>
<tr>
<td>mvwinchnstr</td>
<td>curs_inchstr(3X)</td>
</tr>
<tr>
<td>mvwinchstr</td>
<td>curs_inchstr(3X)</td>
</tr>
<tr>
<td>mvwinstr</td>
<td>curs_instr(3X)</td>
</tr>
<tr>
<td>mvwinnstr</td>
<td>curs_inwstr(3X)</td>
</tr>
<tr>
<td>mvwinnwstr</td>
<td>curs_inwstr(3X)</td>
</tr>
<tr>
<td>mvwinsch</td>
<td>curs_insch(3X)</td>
</tr>
<tr>
<td>mvwinsnstr</td>
<td>curs_insstr(3X)</td>
</tr>
<tr>
<td>mvwinsstr</td>
<td>curs_insstr(3X)</td>
</tr>
<tr>
<td>mvwinstr</td>
<td>curs_instr(3X)</td>
</tr>
<tr>
<td>mvwinswch</td>
<td>curs_inswch(3X)</td>
</tr>
<tr>
<td>mvwinswstr</td>
<td>curs_inswstr(3X)</td>
</tr>
<tr>
<td>mvwinwch</td>
<td>curs_inwch(3X)</td>
</tr>
<tr>
<td>mvwinwchnstr</td>
<td>curs_inwchstr(3X)</td>
</tr>
<tr>
<td>mvwinwchstr</td>
<td>curs_inwchstr(3X)</td>
</tr>
<tr>
<td>mvwinwstr</td>
<td>curs_inwstr(3X)</td>
</tr>
<tr>
<td>mvprintw</td>
<td>curs_printw(3X)</td>
</tr>
<tr>
<td>mvwscanw</td>
<td>curs_scanw(3X)</td>
</tr>
<tr>
<td>napms</td>
<td>curs_kernel(3X)</td>
</tr>
<tr>
<td>newpad</td>
<td>curs_pad(3X)</td>
</tr>
<tr>
<td>newterm</td>
<td>curs_initscr(3X)</td>
</tr>
<tr>
<td>newwin</td>
<td>curs_window(3X)</td>
</tr>
</tbody>
</table>
nl  
cnocbreak  
nodelay  
noecho  
nonl  
nqiflush  
noraw  
notimeout  
overlay  
overwrite  
pair_content  
pchochar  
pchowchar  
 pnoutrefresh  
prefresh  
printw  
pputp  
pputwin  
qiflush  
raw  
redrawwin  
refresh  
reset_prog_mode  
reset_shell_mode  
resetty  
restartterm  
ripoftime  
savetty  
scanw  
scr_dump  
scr_init  
scr_restore  
scr_set  
scroll  
scrollok  
set_curterm  
set_term  
setscreen  
setsyx  
setterm  
setupterm  
silk_attroff  
silk_attron  
silk_attrset  
silk_clear

modified 31 Dec 1996  SunOS 5.6  3X-253
<table>
<thead>
<tr>
<th>Function</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>slk_init</td>
<td>curs_slk(3X)</td>
</tr>
<tr>
<td>slk_label</td>
<td>curs_slk(3X)</td>
</tr>
<tr>
<td>slk_noutrefresh</td>
<td>curs_slk(3X)</td>
</tr>
<tr>
<td>slk_refresh</td>
<td>curs_slk(3X)</td>
</tr>
<tr>
<td>slk_restore</td>
<td>curs_slk(3X)</td>
</tr>
<tr>
<td>slk_set</td>
<td>curs_slk(3X)</td>
</tr>
<tr>
<td>slk_touch</td>
<td>curs_slk(3X)</td>
</tr>
<tr>
<td>srcl</td>
<td>curs_scroll(3X)</td>
</tr>
<tr>
<td>standend</td>
<td>curs_attr(3X)</td>
</tr>
<tr>
<td>standout</td>
<td>curs_attr(3X)</td>
</tr>
<tr>
<td>start_color</td>
<td>curs_color(3X)</td>
</tr>
<tr>
<td>subpad</td>
<td>curs_pad(3X)</td>
</tr>
<tr>
<td>subwin</td>
<td>curs_window(3X)</td>
</tr>
<tr>
<td>syncok</td>
<td>curs_window(3X)</td>
</tr>
<tr>
<td>termattrss</td>
<td>curs_termattrss(3X)</td>
</tr>
<tr>
<td>termname</td>
<td>curs_termattrss(3X)</td>
</tr>
<tr>
<td>tgetent</td>
<td>curs_termcap(3X)</td>
</tr>
<tr>
<td>tgetflag</td>
<td>curs_termcap(3X)</td>
</tr>
<tr>
<td>tgetnum</td>
<td>curs_termcap(3X)</td>
</tr>
<tr>
<td>tgetstr</td>
<td>curs_termcap(3X)</td>
</tr>
<tr>
<td>tgoto</td>
<td>curs_termcap(3X)</td>
</tr>
<tr>
<td>tigetflag</td>
<td>curs_terminfo(3X)</td>
</tr>
<tr>
<td>tigetnum</td>
<td>curs_terminfo(3X)</td>
</tr>
<tr>
<td>tigetstr</td>
<td>curs_terminfo(3X)</td>
</tr>
<tr>
<td>timeout</td>
<td>curs_inopts(3X)</td>
</tr>
<tr>
<td>touchline</td>
<td>curs_touch(3X)</td>
</tr>
<tr>
<td>touchwin</td>
<td>curs_touch(3X)</td>
</tr>
<tr>
<td>tparm</td>
<td>curs_terminfo(3X)</td>
</tr>
<tr>
<td>tputs</td>
<td>curs_terminfo(3X)</td>
</tr>
<tr>
<td>typeahead</td>
<td>curs_inopts(3X)</td>
</tr>
<tr>
<td>unctrl</td>
<td>curs_util(3X)</td>
</tr>
<tr>
<td>ungetch</td>
<td>curs_getch(3X)</td>
</tr>
<tr>
<td>ungetwch</td>
<td>curs_getwch(3X)</td>
</tr>
<tr>
<td>untouchwin</td>
<td>curs_touch(3X)</td>
</tr>
<tr>
<td>use_env</td>
<td>curs_util(3X)</td>
</tr>
<tr>
<td>vidattr</td>
<td>curs_terminfo(3X)</td>
</tr>
<tr>
<td>vidputs</td>
<td>curs_terminfo(3X)</td>
</tr>
<tr>
<td>vwprintw</td>
<td>curs_printw(3X)</td>
</tr>
<tr>
<td>vwscanw</td>
<td>curs_scanw(3X)</td>
</tr>
<tr>
<td>waddch</td>
<td>curs_addch(3X)</td>
</tr>
<tr>
<td>waddchnstr</td>
<td>curs_addchnstr(3X)</td>
</tr>
<tr>
<td>waddchstr</td>
<td>curs_addchnstr(3X)</td>
</tr>
<tr>
<td>waddnstr</td>
<td>curs_addnstr(3X)</td>
</tr>
<tr>
<td>waddnwstr</td>
<td>curs_addnwstr(3X)</td>
</tr>
<tr>
<td>waddstr</td>
<td>curs_addstr(3X)</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-------------------------------------------</td>
</tr>
<tr>
<td>waddwch</td>
<td>curses_addwch(3X)</td>
</tr>
<tr>
<td>waddwchnstr</td>
<td>curses_addwchstr(3X)</td>
</tr>
<tr>
<td>waddwchstr</td>
<td>curses_addwchstr(3X)</td>
</tr>
<tr>
<td>waddwstr</td>
<td>curses_addwstr(3X)</td>
</tr>
<tr>
<td>wadjcurspos</td>
<td>curses_addwchstr(3X)</td>
</tr>
<tr>
<td>wattroff</td>
<td>curses_attr(3X)</td>
</tr>
<tr>
<td>wattron</td>
<td>curses_attr(3X)</td>
</tr>
<tr>
<td>wattrset</td>
<td>curses_attr(3X)</td>
</tr>
<tr>
<td>wbkgd</td>
<td>curses_bkgd(3X)</td>
</tr>
<tr>
<td>wbkgdset</td>
<td>curses_bkgd(3X)</td>
</tr>
<tr>
<td>wborder</td>
<td>curses_border(3X)</td>
</tr>
<tr>
<td>wclear</td>
<td>curs_alecompat(3X)</td>
</tr>
<tr>
<td>wclrtobot</td>
<td>curs_clear(3X)</td>
</tr>
<tr>
<td>wclrtoeol</td>
<td>curs_clear(3X)</td>
</tr>
<tr>
<td>wcursyncup</td>
<td>curs_window(3X)</td>
</tr>
<tr>
<td>wdelch</td>
<td>curs_delch(3X)</td>
</tr>
<tr>
<td>wdeleteLN</td>
<td>curs_deleteLN(3X)</td>
</tr>
<tr>
<td>wechochar</td>
<td>curs_addch(3X)</td>
</tr>
<tr>
<td>wechowchar</td>
<td>curs_addwch(3X)</td>
</tr>
<tr>
<td>werase</td>
<td>curs_clear(3X)</td>
</tr>
<tr>
<td>wgetch</td>
<td>curs_getch(3X)</td>
</tr>
<tr>
<td>wgetnstr</td>
<td>curs_getstr(3X)</td>
</tr>
<tr>
<td>wgetnwstr</td>
<td>curs_getwstr(3X)</td>
</tr>
<tr>
<td>wgetstr</td>
<td>curs_getstr(3X)</td>
</tr>
<tr>
<td>wgetwch</td>
<td>curs_getwch(3X)</td>
</tr>
<tr>
<td>wgetwstr</td>
<td>curs_getwstr(3X)</td>
</tr>
<tr>
<td>whline</td>
<td>curs_border(3X)</td>
</tr>
<tr>
<td>winch</td>
<td>curs_inch(3X)</td>
</tr>
<tr>
<td>winchnstr</td>
<td>curs_inchstr(3X)</td>
</tr>
<tr>
<td>winchstr</td>
<td>curs_inchstr(3X)</td>
</tr>
<tr>
<td>winnstr</td>
<td>curs_instr(3X)</td>
</tr>
<tr>
<td>winnwsr</td>
<td>curs_inwstr(3X)</td>
</tr>
<tr>
<td>winsch</td>
<td>curs_insch(3X)</td>
</tr>
<tr>
<td>winsdellln</td>
<td>curs_deleteLN(3X)</td>
</tr>
<tr>
<td>wininsertln</td>
<td>curs_deleteLN(3X)</td>
</tr>
<tr>
<td>winsnsr</td>
<td>curs_insstr(3X)</td>
</tr>
<tr>
<td>winsnwsr</td>
<td>curs_insstr(3X)</td>
</tr>
<tr>
<td>winssr</td>
<td>curs_insstr(3X)</td>
</tr>
<tr>
<td>winstr</td>
<td>curs_instr(3X)</td>
</tr>
<tr>
<td>winswch</td>
<td>curs_inswwch(3X)</td>
</tr>
<tr>
<td>winswstr</td>
<td>curs_inswwstr(3X)</td>
</tr>
<tr>
<td>winwch</td>
<td>curs_inwch(3X)</td>
</tr>
<tr>
<td>winwchnstr</td>
<td>curs_inwchstr(3X)</td>
</tr>
<tr>
<td>winwchstr</td>
<td>curs_inwchstr(3X)</td>
</tr>
<tr>
<td>winwstr</td>
<td>curs_inwstr(3X)</td>
</tr>
</tbody>
</table>

modified 31 Dec 1996
SunOS 5.6
3X-255
curses (3X) Miscellaneous Library Functions

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)
wsetscrreg    curs_outopts(3X)
wstandend    curs_attr(3X)
wstandout    curs_attr(3X)
wsynccdown    curs_window(3X)
wsyncup    curs_window(3X)
wtimout    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 setscrreg(), wsetscrreg(), getyx(), getbegyx(), and getmaxyx(). The return values of setscrreg(), wsetscrreg(), getyx(), getbegyx(), and getmaxyx() are undefined (that is, these should not be used as the right-hand side of assignment statements).

Routines that return pointers return NULL on error.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
terminfo(4), attributes(5) and 3X pages whose names begin with “curs_” for detailed routine descriptions.

NOTES
The header <curses.h> automatically includes the headers <stdio.h> and <unctrl.h>.

3X-256 SunOS 5.6 modified 31 Dec 1996
NAME
curses – introduction and overview of X/Open Curses

DESCRIPTION
The X/Open Curses screen management package conforms fully with Issue 4 of the
X/Open Extended Curses specification. It provides a set of internationalized functions
and macros for creating and modifying input and output to a terminal screen. This
includes functions for creating windows, highlighting text, writing to the screen, reading
from user input, and moving the cursor. X/Open Curses is designed to optimize screen
update activities.

X/Open Curses is a terminal-independent package, providing a common user interface
to a variety of terminal types. Its portability is facilitated by the Terminfo database which
contains a compiled definition of each terminal type. By referring to the database infor-
mation X/Open Curses gains access to low-level details about individual terminals.

X/Open Curses tailors its activities to the terminal type specified by the TERM envi-
ronment variable. The TERM environment variable may be set in the Korn Shell (see ksh(1))
by typing:

    export TERM=terminal_name

To set environment variables using other command line interfaces or shells, see the
environ(5) manual page.

Three additional environment variables are useful, and can be set in the Korn Shell:

1. If you have an alternate Terminfo database containing terminal types that are not
   available in the system default database /usr/lib/terminfo, you can specify the
   TERMINFO environment variable to point to this alternate database:

       export TERMINFO=path

   This path specifies the location of the alternate compiled Terminfo database whose
   structure consists of directory names 0 to 9 and a to z (which represent the first
   letter of the compiled terminal definition file name).

   The alternate database specified by TERMINFO is examined before the system
default database. If the terminal type specified by TERM cannot be found in either
database, the default terminal type dumb is assumed.

2. To specify a window width smaller than your screen width (for example, in situa-
tions where your communications line is slow), set the COLUMNS
   environment variable to the number of vertical columns you want between the
   left and right margins:

       export COLUMNS=number

   The number of columns may be set to a number smaller than the screen size; how-
ever, if set larger than the screen or window width, the results are undefined.

   The value set using this environment variable takes precedence over the value nor-
mally used for the terminal.

3. To specify a window height smaller than your current screen height (for example,
in situations where your communications line is slow), override the LINES

modified 1 Jun 1996

SunOS 5.6

3XC-257
environment variable by setting it to a smaller number of horizontal lines:

```
export LINES=number
```

The `number` of lines may be set to a number smaller than the screen height; however, if set larger than the screen or window height, the results are undefined.

The value set using this environment variable takes precedence over the value normally used for the terminal.

**Data Types**

X/Open Curses defines the following data types:

- **attr_t** an integral type that holds an OR-ed set of attributes. The attributes acceptable are those which begin with the `WA_` prefix (see Attributes, Color Pairs, and Renditions).
- **bool** Boolean data type.
- **cchar_t** a type that refers to a string consisting of a spacing wide character, up to 5 non-spacing wide characters, and zero or more attributes of any type (see Attributes, Color Pairs, and Renditions). A null `cchar_t` object terminates arrays of `cchar_t` objects.
- **chtype** an integral type whose values are formed by OR-ing an "unsigned char" with a color pair (see Attributes, Color Pairs, and Renditions) and with zero or more attributes. The attributes acceptable are those which begin with the `A_` prefix and `COLOR_PAIR(3XC)` (see Attributes, Color Pairs, and Renditions).
- **SCREEN** an opaque data type associated with a terminal’s display screen.
- **TERMINAL** an opaque data type associated with a terminal. It contains information about the terminal’s capabilities (as defined by `terminfo`), the terminal modes, and current state of input/output operations.
- **wchar_t** an integral data type whose values represent wide characters.
- **WINDOW** an opaque data type associated with a window.

**Screens, Windows, and Terminals**

The X/Open Curses documentation refers at various points to screens, windows (also subwindows, derived windows, and pads), and terminals. The following list defines each of these terms.

- **Screen** A screen is a terminal’s physical output device. The `SCREEN` data type is associated with a terminal.
- **Window** Window objects are two-dimensional arrays of characters and their renditions. X/Open Curses provides `stdscr`, a default window which is the size of the terminal screen. You can use the `newwin(3XC)` function to create others.

To refer to a window, use a variable declared as `WINDOW *`. X/Open Curses includes both functions that modify `stdscr`, and more general versions that let you specify a window.
There are three sub-types of windows:

- **Subwindow**: a window which has been created within another window (the parent window) and whose position has been specified with absolute screen coordinates. The `derwin(3XC)` and `subwin(3XC)` functions can be used to create subwindows.

- **Derived Window**: a subwindow whose position is defined relative to the parent window’s coordinates rather than in absolute terms.

- **Pad**: a special type of window that can be larger than the screen. For more information, see the `newpad(3XC)` man page.

- **Terminal**: A terminal is the input and output device which character-based applications use to interact with the user. The `TERMINAL` data type is associated with such a device.

**Attributes, Color Pairs, and Renditions**

A character’s rendition consists of its attributes (such as underlining or reverse video) and its color pair (the foreground and background colors). When using `waddstr(3XC)`, `waddchstr(3XC)`, `wprintw(3XC)`, `winsch(3XC)`, and so on, the window’s rendition is combined with that character’s renditions. The window rendition is the attributes and color set using the `attroff(3XC)` and `attr_off(3XC)` sets of functions. The window’s background character and rendition are set with the `bkgdset(3XC)` and `bkgrndset(3XC)` sets of functions.

When spaces are written to the screen, the background character and window rendition replace the space. For example, if the background rendition and character is `A_UNDERLINE|'∗'`, text written to the window appears underlined and the spaces appear as underlined asterisks.

Each character written retains the rendition that it has obtained. This allows the character to be copied "as is" to or from a window with the `addchstr(3XC)` or `inch(3XC)` functions.
You can specify attributes using the constants listed in the tables provided in this man page. The following constants modify objects of type `chttype`:

**A_ Constant Values for Highlighting Attributes**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A_ALTCHARSET</td>
<td>Alternate character set</td>
</tr>
<tr>
<td>A_ATTRIBUTES</td>
<td>Attribute mask</td>
</tr>
<tr>
<td>A_BLINK</td>
<td>Blinking</td>
</tr>
<tr>
<td>A_BOLD</td>
<td>Bold</td>
</tr>
<tr>
<td>A_CHARTEXT</td>
<td>Character mask</td>
</tr>
<tr>
<td>A_COLOR</td>
<td>Color mask</td>
</tr>
<tr>
<td>A_DIM</td>
<td>Dim</td>
</tr>
<tr>
<td>A_INVIS</td>
<td>Invisible</td>
</tr>
<tr>
<td>A_NORMAL</td>
<td>Disable attributes</td>
</tr>
<tr>
<td>A_PROTECT</td>
<td>No display</td>
</tr>
<tr>
<td>A_REVERSE</td>
<td>Reverse video</td>
</tr>
<tr>
<td>A_STANDOUT</td>
<td>Highlights specific to terminal</td>
</tr>
<tr>
<td>A_UNDERLINE</td>
<td>Underline</td>
</tr>
</tbody>
</table>

The following constants modify objects of type `attr_t`:

**WA_ Constant Values for Highlighting Attributes**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WA_ALTCHARSET</td>
<td>Alternate character set</td>
</tr>
<tr>
<td>WA_ATTRIBUTES</td>
<td>Attribute mask</td>
</tr>
<tr>
<td>WA_BLINK</td>
<td>Blinking</td>
</tr>
<tr>
<td>WA_BOLD</td>
<td>Bold</td>
</tr>
<tr>
<td>WA_DIM</td>
<td>Dim</td>
</tr>
<tr>
<td>WA_HORIZONTAL</td>
<td>Horizontal highlight</td>
</tr>
<tr>
<td>WA_INVIS</td>
<td>Invisible</td>
</tr>
<tr>
<td>WA_LEFT</td>
<td>Left highlist</td>
</tr>
<tr>
<td>WA_LOW</td>
<td>Low highlist</td>
</tr>
<tr>
<td>WA_PROTECT</td>
<td>No display</td>
</tr>
<tr>
<td>WA_REVERSE</td>
<td>Reverse video</td>
</tr>
<tr>
<td>WA_RIGHT</td>
<td>Right highlight</td>
</tr>
<tr>
<td>WA_STANDOUT</td>
<td>Highlights specific to terminal</td>
</tr>
<tr>
<td>WA_TOP</td>
<td>Top highlist</td>
</tr>
<tr>
<td>WA_UNDERLINE</td>
<td>Underline</td>
</tr>
<tr>
<td>WA_VERTICAL</td>
<td>Vertical highlight</td>
</tr>
</tbody>
</table>
Colors always appear in pairs; the foreground color of the character itself and the background color of the field on which it is displayed. The following color macros are defined:

<table>
<thead>
<tr>
<th>Color Macros</th>
</tr>
</thead>
<tbody>
<tr>
<td>Macro</td>
</tr>
<tr>
<td>--------------</td>
</tr>
<tr>
<td>COLOR_BLACK</td>
</tr>
<tr>
<td>COLOR_BLUE</td>
</tr>
<tr>
<td>COLOR_GREEN</td>
</tr>
<tr>
<td>COLOR_CYAN</td>
</tr>
<tr>
<td>COLOR_RED</td>
</tr>
<tr>
<td>COLOR_MAGENTA</td>
</tr>
<tr>
<td>COLOR_YELLOW</td>
</tr>
<tr>
<td>COLOR_WHITE</td>
</tr>
</tbody>
</table>

Together, a character’s attributes and its color pair form the character’s rendition. A character’s rendition moves with the character during any scrolling or insert/delete operations. If your terminal lacks support for the specified rendition, X/Open Curses may substitute a different rendition.

The COLOR_PAIR(3XC) function modifies a chtype object. The PAIR_NUMBER(3XC) function extracts the color pair from a chtype object.

The following functions modify a window’s color:

<table>
<thead>
<tr>
<th>Functions for Modifying a Window’s Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function</td>
</tr>
<tr>
<td>attr_set(), wattr_set()</td>
</tr>
<tr>
<td>color_set(), wcolor_set()</td>
</tr>
</tbody>
</table>

Non-Spacing Characters

When the wcwidth(3C) function returns a width of zero for a character, that character is called a non-spacing character. Non-spacing characters can be written to a window.

Each non-spacing character is associated with a spacing character (that is, one which does not have a width of zero) and modifies that character. You cannot address a non-spacing character directly. Whenever you perform an X/Open Curses operation on the associated character, you are implicitly addressing the non-spacing character.

Non-spacing characters do not have a rendition. For functions that use wide characters and a rendition, X/Open Curses ignores any rendition specified for non-spacing characters. Multicolumn characters have one rendition that applies to all columns spanned.

Complex Characters

The cchar_t date type represents a complex character. A complex character may contain a spacing character, its associated non-spacing characters, and its rendition. This implementation of complex characters supports up to 5 non-spacing characters for each spacing character.

When a cchar_t object representing a non-spacing complex character is written to the screen, its rendition is not used, but rather it becomes associated with the rendition of the existing character at that location. The setcchar(3XC) function initializes an object of type.
cchar_t. The `getchar(3XC)` function extracts the contents of a `cchar_t` object.

**Display Operations**

In adding internationalization support to X/Open Curses, every attempt was made to minimize the number of changes to the historical CURSES package. This enables programs written to use the historical implementation of CURSES to use the internationalized version with little or no modification. The following rules apply to the internationalized X/Open Curses package:

- The cursor can be placed anywhere in the window. Window and screen origins are (0,0).
- A multicolumn character cannot be displayed in the last column, because the character would appear truncated. Instead, the background character is displayed in the last column and the multicolumn character appears at the beginning of the next line. This is called wrapping.

If the original line is the last line in the scroll region and scrolling is enabled, X/Open Curses moves the contents of each line in the region to the previous line. The first line of the region is lost. The last line of the scrolling region contains any wrapped characters. The remainder of that line is filled with the background character. If scrolling is disabled, X/Open Curses truncates any character that would extend past the last column of the screen.

- Overwrites operate on screen columns. If displaying a single-column or multicolumn character results in overwriting only a portion of a multicolumn character or characters, background characters are displayed in place of the non-overwritten portions.
- Insertions and deletions operate on whole characters. The cursor is moved to the first column of the character prior to performing the operation.

**Overlapping Windows**

When windows overlap, it may be necessary to overwrite only part of a multicolumn character. As mentioned earlier, the non-overwritten portions are replaced with the background character. This results in issues concerning the `overwrite(3XC)`, `overlay(3XC)`, `copywin(3XC)`, `wnoutrefresh(3XC)`, and `wrefresh(3XC)` functions.

In the upcoming examples, some characters have special meanings:

- {, [, and ( represent the left halves of multicolumn characters. }, ], and ) represent the corresponding right halves of the same multicolumn characters.
- Alphanumeric characters and periods (.) represent single-column characters.
- The number sign (#) represents the background character.
The following examples show how X/Open Curses deals with a number of issues:

1. Copying single-column characters over single-column characters.
   \[\text{copywin}\left(s, t, 0, 1, 0, 1, 1, 3, 0\right)\]
   \[
   \begin{array}{c}
   s \\
   \text{abcdef} \\
   \text{ghijk}l
   \end{array}
   \rightarrow
   \begin{array}{c}
   t \\
   \text{.......} \\
   \text{.bcd..} \\
   \text{.hij..}
   \end{array}
   \]

   There are no special problems with this situation.

2. Copying multicolumn characters over single-column characters.
   \[\text{copywin}\left(s, t, 0, 1, 0, 1, 1, 3, 0\right)\]
   \[
   \begin{array}{c}
   s \\
   \text{a[ ]def} \\
   \text{gh( )kl}
   \end{array}
   \rightarrow
   \begin{array}{c}
   t \\
   \text{.......} \\
   \text{.[ ]d..} \\
   \text{.h( )..}
   \end{array}
   \]

   There are no special problems with this situation.

3. Copying single-column characters from source overlaps multicolumn characters in target.
   \[\text{copywin}\left(s, t, 0, 1, 0, 1, 1, 3, 0\right)\]
   \[
   \begin{array}{c}
   s \\
   \text{abcdef} \\
   \text{ghijk}l
   \end{array}
   \rightarrow
   \begin{array}{c}
   t \\
   \text{[ ]....} \\
   \text{#bcd..} \\
   \text{...().} \\
   \text{.hij#.}
   \end{array}
   \]

   Overwriting multicolumn characters in \(t\) has resulted in the # background characters being required to erase the remaining halves of the target’s multicolumn characters.

4. Copy incomplete multicolumn characters from source to target.
   \[\text{copywin}\left(s, t, 0, 1, 0, 1, 1, 3, 0\right)\]
   \[
   \begin{array}{c}
   s \\
   \text{[ ]cdef} \\
   \text{ghi()}l
   \end{array}
   \rightarrow
   \begin{array}{c}
   t \\
   \text{123456} \\
   \text{789012}
   \end{array}
   \]

   \[
   \begin{array}{c}
   \text{[ ]cd56}
   \text{7hi()}2
   \end{array}
   \]

   The ] and ( halves of the multicolumn characters have been copied from the source and expanded in the target outside of the specified target region.
Consider a pop-up dialog box that contains single-column characters and a base window that contains multicolumn characters and you do the following:

```
save=dupwin(dialog); /* create backing store */
overwrite(cursor, save); /* save region to be overlayed */
wrefresh(dialog); /* display dialog */
wrefresh(save); /* restore screen image */
delwin(save); /* release backing store */
```

You can use code similar to this to implement generic `popup()` and `popdown()` routines in a variety of CURSES implementations (including BSD UNIX, and UNIX System V). In the simple case where the base window contains single-column characters only, it would correctly restore the image that appeared on the screen before the dialog box was displayed.

However, with multicolumn characters, the `overwrite()` function might save a region with incomplete multicolumn characters. The `wrefresh(dialog)` statement results in the behavior described in example 3. The behavior described in this example (that is, example 4) allows the `wrefresh(save)` statement to restore the window correctly.

5. Copying an incomplete multicolumn character to region next to screen margin (not a window edge).

**Case (a)**

```
copywin(s, t, 0, 1, 0, 0, 1, 2, 0)

s  t  →  t
[]cdef 123456 #cd456
ghi jkl 789012 hij012
```

The background character (#) replaces the j character that would have been copied from the source, because it is not possible to expand the multicolumn character to its complete form.

**Case (b)**

```
copywin(s, t, 0, 1, 0, 3, 1, 5, 0)

s  t  →  t
abcdef 123456 123bcd
ghi{}l 789012 789hi#
```

This is the same as Case (a) but with the right margin.

**Special Characters**

Some functions assign special meanings to certain special characters:

- **Backspace** moves the cursor one column towards the beginning of the line. If the cursor was already at the beginning of the line, it remains there. All subsequent characters are added or inserted at this point.

3XC-264 SunOS 5.6 modified 1 Jun 1996
Carriage Return moves the cursor to the beginning of the current line. If the cursor was already at the beginning of the line, it remains there. All subsequent characters are added or inserted at this point.

Newline When adding characters, X/Open Curses fills the remainder of the line with the background character (effectively truncating the newline) and scrolls the window as described earlier. All subsequent characters are inserted at the start of the new line.

When inserting characters, X/Open Curses fills the remainder of the line with the background character (effectively truncating the line), moves the cursor to the beginning of a new line, and scrolls the window as described earlier. All subsequent characters are placed at the start of the new line.

Tab moves subsequent characters to next horizontal tab stop. Default tab stops are set at 0, 8, 16, and so on.

When adding or inserting characters, X/Open Curses inserts or adds the background character into each column until the next tab stop is reached. If there are no remaining tab stops on the current line, wrapping and scrolling occur as described earlier.

Control Characters

When X/Open Curses functions perform special character processing, they convert control characters to the `X notation, where X is a single-column character (uppercase, if it is a letter) and writes that notation to the window. Functions that retrieve text from the window will retrieve the converted notation not the original.

X/Open Curses displays non-printable bytes, that have their high bit set, using the M-X meta notation where X is the non-printable byte with its high bit turned off.

There are four input modes possible with X/Open Curses that affect the behavior of input functions like `getch(3XC) and `getnstr(3XC).

Line Canonical (Cooked)

In line input mode, the terminal driver handles the input of line units as well as SIGERASE and SIGKILL character processing. See `termio(7I) for more information.

In this mode, the `getch() and `getnstr() functions will not return until a complete line has been read by the terminal driver, at which point only the requested number of bytes/characters are returned. The rest of the line unit remains unread until subsequent call to the `getch() or `getnstr() functions.

The functions `nocbreak(3XC) and `noraw(3XC) are used to enter this mode. These functions are described on the `cbreak(3XC) man page which also details which `termios flags are enabled.

modified 1 Jun 1996

SunOS 5.6

3XC-265
Of the modes available, this one gives applications the least amount of control over input. However, it is the only input mode possible on a block mode terminal.

**cbreak Mode**  
Byte/character input provides a finer degree of control. The terminal driver passes each byte read to the application without interpreting erase and kill characters. It is the application’s responsibility to handle line editing. It is unknown whether the signal characters (SIGINT, SIGQUIT, SIGSUSP) and flow control characters (SIGSTART, SIGSTOP) are enabled. To ensure that they are, call the noraw() function first, then call the cbreak() function.

**halfdelay Mode**  
This is the same as the cbreak() mode with a timeout. The terminal driver waits for a byte to be received or for a timer to expire, in which case the getch() function either returns a byte or ERR respectively. This mode overrides timeouts set for an individual window with the wtimeout() function.

**raw Mode**  
This mode provides byte/character input with the most control for an application. It is similar to cbreak() mode, but also disables signal character processing (SIGINT, SIGSUSP, SIGQUIT) and flow control processing (SIGSTART, SIGSTOP) so that the application can process them as it wants.

These modes affect all X/Open Curses input. The default input mode is inherited from the parent process when the application starts up.

A timeout similar to halfdelay(3XC) can be applied to individual windows (see timeout(3XC)). The nodelay(3XC) function is equivalent to setting wtimeout(3XC) for a window with a zero timeout (non-blocking) or infinite delay (blocking).

To handle function keys, keypad(3XC) must be enabled. When it is enabled, the getch() function returns a KEY_ constant for a uniquely encoded key defined for that terminal. When keypad() is disabled, the getch() function returns the individual bytes composing the function key (see getch(3XC) and wget_wch(3XC)). By default, keypad() is disabled.

When processing function keys, once the first byte is recognized, a timer is set for each subsequent byte in the sequence. If any byte in the function key sequence is not received before the timer expires, the bytes already received are pushed into a buffer and the original first byte is returned. Subsequent X/Open Curses input would take bytes from the buffer until exhausted, after which new input from the terminal will be requested. Enabling and disabling of the function key interbyte timer is handled by the notimeout(3XC) function. By default, notimeout() is disabled (that is, the timer is used).

X/Open Curses always disables the terminal driver’s echo processing. The echo(3XC) and noecho(3XC) functions control X/Open Curses software echoing. When software echoing is enabled, X/Open Curses input functions echo printable characters, control keys, and meta keys in the input window at the last cursor position. Functions keys are never echoed. When software echoing is disabled, it is the application’s responsibility to handle echoing.
SEE ALSO ksh(1), COLOR_PAIR(3XC), PAIR_NUMBER(3XC), addchstr(3XC), attr_off(3XC), attroff(3XC), bkgdset(3XC), bkgrndset(3XC), cbreak(3XC), copywin(3XC), derwin(3XC), echo(3XC), getcchar(3XC), getch(3XC), getnstr(3XC), halfdelay(3XC), inch(3XC), keypad(3XC), newpad(3XC), newwin(3XC), nocbreak(3XC), nodelay(3XC), noecho(3XC), noraw(3XC), ntimeout(3XC), overlay(3XC), overwrite(3XC), setcchar(3XC), subwin(3XC), timeout(3XC), waddchstr(3XC), waddstr(3XC), wcwidth(3C), wget_wch(3XC), winsch(3XC), wnoutrefresh(3XC), wprintw(3XC), wrefresh(3XC), wtimeout(3XC), termio(7I), environ(5)

modified 1 Jun 1996 SunOS 5.6 3XC-267
NAME
curs_addch, addch, waddch, mvaddch, mvwaddch, echochar, wechochar – add a character (with attributes) to a curses window and advance cursor

SYNOPSIS
cc [ flag ... ] file ... -lcurses [ library ... ]
#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);

DESCRIPTION
With the addch(), waddch(), mvaddch(), and mvwaddch() routines, the character ch 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 scrolllok() is enabled, the scrolling region is scrolled up one line.

If ch 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 ch 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).

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 stdout(), predefined video attribute constants, on the curs_attr(3X) page).

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.

Line Graphics
The following variables may be used to add line drawing characters to the screen with routines of the addch() family. When variables are defined for the terminal, the A_ALTCHARSET bit is turned on (see curs_attr(3X)). Otherwise, the default character listed below is stored in the variable. The names chosen are consistent with the VT100 nomenclature.
<table>
<thead>
<tr>
<th>Name</th>
<th>Default</th>
<th>Glyph Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACS_ULCORNER</td>
<td>+</td>
<td>upper left-hand corner</td>
</tr>
<tr>
<td>ACS_LLCORNER</td>
<td>+</td>
<td>lower left-hand corner</td>
</tr>
<tr>
<td>ACS_URCORNER</td>
<td>+</td>
<td>upper right-hand corner</td>
</tr>
<tr>
<td>ACS_LRCORNER</td>
<td>+</td>
<td>lower right-hand corner</td>
</tr>
<tr>
<td>ACS_RTEE</td>
<td>+</td>
<td>right tee (−)</td>
</tr>
<tr>
<td>ACS_LTEE</td>
<td>+</td>
<td>left tee (−)</td>
</tr>
<tr>
<td>ACS_BTEE</td>
<td>+</td>
<td>bottom tee (↓)</td>
</tr>
<tr>
<td>ACS_TTEE</td>
<td>+</td>
<td>top tee (↑)</td>
</tr>
<tr>
<td>ACS_HLINE</td>
<td>−</td>
<td>horizontal line</td>
</tr>
<tr>
<td>ACS_VLINE</td>
<td></td>
<td>vertical line</td>
</tr>
<tr>
<td>ACS_PLUS</td>
<td>+</td>
<td>plus</td>
</tr>
<tr>
<td>ACS_S1</td>
<td>−</td>
<td>scan line 1</td>
</tr>
<tr>
<td>ACS_S9</td>
<td>−</td>
<td>scan line 9</td>
</tr>
<tr>
<td>ACS_DIAMOND</td>
<td>+</td>
<td>diamond</td>
</tr>
<tr>
<td>ACS_CKBOARD</td>
<td>:</td>
<td>checker board (stipple)</td>
</tr>
<tr>
<td>ACS_DEGREE</td>
<td>'</td>
<td>degree symbol</td>
</tr>
<tr>
<td>ACS_PLMINUS</td>
<td>#</td>
<td>plus/minus</td>
</tr>
<tr>
<td>ACS_BULLET</td>
<td>o</td>
<td>bullet</td>
</tr>
<tr>
<td>ACS_LARROW</td>
<td>&lt;</td>
<td>arrow pointing left</td>
</tr>
<tr>
<td>ACS_RARROW</td>
<td>&gt;</td>
<td>arrow pointing right</td>
</tr>
<tr>
<td>ACS_DARROW</td>
<td>v</td>
<td>arrow pointing down</td>
</tr>
<tr>
<td>ACS_UARROW</td>
<td>-</td>
<td>arrow pointing up</td>
</tr>
<tr>
<td>ACS_BOARD</td>
<td>#</td>
<td>board of squares</td>
</tr>
<tr>
<td>ACS_LANTERN</td>
<td>#</td>
<td>lantern symbol</td>
</tr>
<tr>
<td>ACS_BLOCK</td>
<td>#</td>
<td>solid square block</td>
</tr>
</tbody>
</table>

**RETURN VALUES**

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

**ATTRIBUTES**

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

**SEE ALSO**

`curs_attr(3X), curs_clear(3X), curs_inch(3X), curs_outopts(3X), curs_refresh(3X), curses(3X), putc(3S), attributes(5)`

**NOTES**

The header `<curses.h>` automatically includes the headers `<stdio.h>` and `<unctrl.h>`. Note that `addch()`, `mvaddch()`, `mvwaddch()`, and `echochar()` may be macros.

modified 31 Dec 1996

SunOS 5.6

3X-269
NAME
curs_addchstr, addchstr, addchnstr, waddchstr, waddchnstr, mvaddchstr, mvaddchnstr,
mvwaddchstr, mvwaddchnstr – add string of characters (and attributes) to a curses window

SYNOPSIS
c [ flag . . . ] file . . . - Icurses [ library . . ]
#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);

DESCRIPTION
All of these routines copy chstr directly into the window image structure starting at the current cursor position. The four routines with n as the last argument copy at most n elements, but no more than will fit on the line. If n=-1 then the whole string is copied, to the maximum number that fit on the line.

The position of the window cursor is not advanced. These routines works faster than waddnstr() (see curs_addstr(3X)) because they merely copy chstr 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 then wrapping it around to the next line.

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

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

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
curs_addstr(3X), curses(3X), attributes(5)

NOTES
The header <curses.h> automatically includes the headers <stdio.h> and <unctrl.h>.
Note that all routines except waddchnstr() and waddchstr() may be macros.
NAME

curs_addstr, addstr, addnstr, waddstr, waddnstr, mvaddstr, mvaddnstr, mvwaddstr, 
mvwaddnstr – add a string of characters to a curses window and advance cursor

SYNOPSIS

cc [ flag ... ] file ... -lcurses [ library ... ]
#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);

DESCRIPTION

All of these routines write all the characters of the null terminated character string str on 
the given window. It is similar to calling waddch() once for each character in the string. 
The four routines with n as the last argument write at most n characters. If n 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.

ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO

curs_addch(3X), curses(3X), attributes(5)

NOTES

The header <curses.h> automatically includes the headers <stdio.h> and <unctrl.h>. 
Note that all routines except waddstr() and waddnstr() may not be macros.
NAME
curs_addwch, addwch, waddwch, mvaddwch, mvwaddwch, echowchar, wechowchar –
add a wchar_t character (with attributes) to a curses window and advance cursor

SYNOPSIS
cc [ flag ... ] file ... -lcurses [ library ... ]
#include <curses.h>

int addwch(chtype wch);
int waddwch(WINDOW *win, chtype wch);
int mvaddwch(int y, int x, chtype wch);
int mvwaddwch(WINDOW *win, int y, int x, chtype wch);
int echowchar(chtype wch);
int wechowchar(WINDOW *win, chtype wch);

DESCRIPTION
The addwch(), waddwch(), mvaddwch(), and mvwaddwch() routines put the character
wch, holding a wchar_t character, into the window at the current cursor position of the
window and advance the position of the window cursor. Their function is similar to that
of putwchar(3S) in the C multibyte library. At the right margin, an automatic newline is
performed. At the bottom of the scrolling region, if scrollok is enabled, the scrolling
region is scrolled up one line.

If wch is a tab, newline, or backspace, the cursor is moved appropriately within the window.
A newline also does a clrtoeol(3X) before moving. Tabs are considered to be at
every eighth column. If wch is another control character, it is drawn in the "X notation.
Calling winwch(3X) after adding a control character does not return the control character,
but instead returns the representation of the control character.

Video attributes can be combined with a wchar_t character by OR-ing them into the
parameter. This results in these attributes also being set. (The intent here is that text,
including attributes, can be copied from one place to another using inwch() and
addwch().) See standout(3X), predefined video attribute constants.

The echowchar() and wechowchar() routines are functionally equivalent to a call to
addwch() followed by a call to refresh(3X), or a call to waddwch() followed by a call to
wrefresh(3X). The knowledge that only a single character is being output is taken into
consideration and, for non-control characters, a considerable performance gain might be
seen by using these routines instead of their equivalents.

Line Graphics
The following variables may be used to add line drawing characters to the screen with
routines of the addwch() family. When variables are defined for the terminal, the
A_ALTCCHARSET 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.
<table>
<thead>
<tr>
<th>Name</th>
<th>Default</th>
<th>Glyph Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACS_ULCORNER</td>
<td>+</td>
<td>upper left-hand corner</td>
</tr>
<tr>
<td>ACS_LLCORNER</td>
<td>+</td>
<td>lower left-hand corner</td>
</tr>
<tr>
<td>ACS_URCORNER</td>
<td>+</td>
<td>upper right-hand corner</td>
</tr>
<tr>
<td>ACS_LRCORNER</td>
<td>+</td>
<td>lower right-hand corner</td>
</tr>
<tr>
<td>ACS_RTEE</td>
<td>+</td>
<td>right tee (−)</td>
</tr>
<tr>
<td>ACS_LTEE</td>
<td>+</td>
<td>left tee (−)</td>
</tr>
<tr>
<td>ACS_BTEE</td>
<td>+</td>
<td>bottom tee ()</td>
</tr>
<tr>
<td>ACS_TTEE</td>
<td>+</td>
<td>top tee ()</td>
</tr>
<tr>
<td>ACS_HLINE</td>
<td>−</td>
<td>horizontal line</td>
</tr>
<tr>
<td>ACS_VLINE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACS_PLUS</td>
<td>+</td>
<td>plus</td>
</tr>
<tr>
<td>ACS_S1</td>
<td>−</td>
<td>scan line 1</td>
</tr>
<tr>
<td>ACS_S9</td>
<td>−</td>
<td>scan line 9</td>
</tr>
<tr>
<td>ACS_DIAMOND</td>
<td>+</td>
<td>diamond</td>
</tr>
<tr>
<td>ACS_CKBOARD</td>
<td>:</td>
<td>checker board (stipple)</td>
</tr>
<tr>
<td>ACS_DEGREE</td>
<td>'</td>
<td>degree symbol</td>
</tr>
<tr>
<td>ACS_PLMINUS</td>
<td>#</td>
<td>plus/minus</td>
</tr>
<tr>
<td>ACS_BULLET</td>
<td>o</td>
<td>bullet</td>
</tr>
<tr>
<td>ACS_LARROW</td>
<td>&lt;</td>
<td>arrow pointing left</td>
</tr>
<tr>
<td>ACS_RARROW</td>
<td>&gt;</td>
<td>arrow pointing right</td>
</tr>
<tr>
<td>ACS_DARROW</td>
<td>v</td>
<td>arrow pointing down</td>
</tr>
<tr>
<td>ACS_UARROW</td>
<td>-</td>
<td>arrow pointing up</td>
</tr>
<tr>
<td>ACS_BOARD</td>
<td>#</td>
<td>board of squares</td>
</tr>
<tr>
<td>ACS_LANTERN</td>
<td>#</td>
<td>lantern symbol</td>
</tr>
<tr>
<td>ACS_BLOCK</td>
<td>#</td>
<td>solid square block</td>
</tr>
</tbody>
</table>

**RETURN VALUE**

All routines return the integer ERR upon failure and an integer value other than ERR upon successful completion, unless otherwise noted in the preceding routine descriptions.

**ATTRIBUTES**

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

**SEE ALSO**

putwchar(3S), clrtoeol(3X), curses(3X), curs_attr(3X), curs_inwch(3X), curs_outopts(3X), refresh(3X), standout(3X), winwch(3X), wrefresh(3X), attributes(5)

**NOTES**

The header file <curses.h> automatically includes the header files <stdio.h>, <unctrl.h> and <widec.h>.

Note that addwch(), mvaddwch(), mvwaddwch(), and echowchar() may be macros.

modified 31 Dec 1996 SunOS 5.6 3X-273
None of these routines can use the color attribute in `chtype`. 
NAME
curs_addwchstr, addwchstr, addwchnstr, waddwchstr, waddwchnstr, mvaddwchstr,
mvaddwchnstr, mvwaddwchstr, mvwaddwchnstr – add string of wchar_t characters
(and attributes) to a curses window

SYNOPSIS
cc [ flag ... ] file ... -lcurses [ library ... ]
#include <curses.h>

int addwchstr(chtype *wchstr);
int addwchnstr(chtype *wchstr, int n);
int waddwchstr(WINDOW *win, chtype *wchstr);
int waddwchnstr(WINDOW *win, chtype *wchstr, int n);
int mvaddwchstr(int y, int x, chtype *wchstr);
int mvaddwchnstr(int y, int x, chtype *wchstr, int n);
int mvwaddwchstr(WINDOW *win, int y, int x, chtype *wchstr);
int mvwaddwchnstr(WINDOW *win, int y, int x, chtype *wchstr, int n);

DESCRIPTION
All of these routines copy wchstr, which points to a string of wchar_t characters, directly
into the window image structure starting at the current cursor position. The four rou-
tines with n as the last argument copy at most n elements, but no more than will fit on
the line. If n=-1 then the whole string is copied, to the maximum number that fit on the
line.

The position of the window cursor is not advanced. These routines work faster than
waddnwstr(3X) because they merely copy wchstr into the window image structure. On
the other hand, care must be taken when using these functions because they don’t per-
form any kind of checking (such as for the newline character), they do not advance the
current cursor position, and they truncate the string, rather than wrapping it around to
the new line.

RETURN VALUE
All routines return the integer ERR upon failure and an integer value other than ERR
upon successful completion, unless otherwise noted in the preceding routine descrip-
tions.
ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO

curses(3X), waddnwstr(3X), attributes(5)

NOTES

The header file <curses.h> automatically includes the header files <stdio.h>, <unctrl.h> and <widec.h>.

Note that all routines except waddwchnstr() may be macros.

None of these routines can use the color attribute in chtype.
NAME
curs_addwstr, addwstr, addnstr, waddwstr, waddnwstr, mvaddwstr, mvaddnstr, mvwaddwstr, mvwaddnwstr – add a string of wchar_t characters to a curses window and advance cursor

SYNOPSIS
cc [ flag ... ] file ... -lcurses [ library ... ]
#include <curses.h>

int addwstr(wchar_t *wstr);
int addnstr(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 mvaddnstr(int y, int x, wchar_t *wstr, int n);
int mvwaddwstr(WINDOW *win, int y, int x, wchar_t *wstr);
int mvwaddnwstr(WINDOW *win, int y, int x, wchar_t *wstr, int n);

DESCRIPTION
All of these routines write all the characters of the null-terminated wchar_t character string wstr on the given window. The effect is similar to calling waddwch(3X) once for each wchar_t character in the string. The four routines with n as the last argument write at most n wchar_t characters. If n is negative, then the entire string will be added.

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

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-LEVEL</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
curses(3X), waddwch(3X), attributes(5)

NOTES
The header file <curses.h> automatically includes the header files <stdio.h>, <unctrl.h> and <widec.h>

Note that all of these routines except waddwstr() and waddnwstr() may be macros.

modified 31 Dec 1996

SunOS 5.6

3X-277
NAME
curs_alecompat, movenextch, wmovenextch, moveprevch, wmoveprevch, adjcurspos, wadjcurspos – these functions are added to ALE curses library for moving the cursor by character.

SYNOPSIS
c c [ flag ... ] file ... -lcurses [ library ... ]
#include <curses.h>

int movenextch(void);
int wmovenextch(WINDOW *win);
int moveprevch(void);
int wmoveprevch(WINDOW *win);
int adjcurspos(void);
int wadjcurspos(WINDOW *win);

DESCRIPTION
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.

moveprevch() and wmoveprevch() routines are the opposite of movenextch() and wmovenextch(), moving the cursor to the left-most column of the previous character.

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.

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

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
curses(3X), getyx(3X), attributes(5)

NOTES
The header file <curses.h> automatically includes the header files <stdio.h>, <unctrl.h> and <widec.h>.

3X-278 SunOS 5.6 modified 31 Dec 1996
Note that `movenextch()`, `moveprevch()`, and `adjcurspos()` may be macros.
NAME  curs_attr, attroff, wattroff, attron, wattroff, attrset, wattrset, standend, wstandend, standout, wstandout — curses character and window attribute control routines

SYNOPSIS cc [ flag ... ] file ... -lcurses [ library ... ]
#include <curses.h>
int attroff(int attrs);
int wattroff(WINDOW *win, int attrs);
int attron(int attrs);
int wattron(WINDOW *win, int attrs);
int attset(int attrs);
int wattrset(WINDOW *win, int attrs);
int standend(void);
int wstandend(WINDOW *win);
int standout (void);
int wstandout(WINDOW *win);

DESCRIPTION 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.

The routine attrset() sets the current attributes of the given window to attrs. 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 attrset(), that is, it turns off all attributes.

Attributes The following video attributes, defined in <curses.h>, can be passed to the routines attron(), attroff(), and attrset(), or OR-ed with the characters passed to addch().

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>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_ALTCOLORSET</td>
<td>Alternate character set</td>
</tr>
<tr>
<td>A_CHARSETTEXT</td>
<td>Bit-mask to extract a character</td>
</tr>
<tr>
<td>COLOR_PAIR(n)</td>
<td>Color-pair number n</td>
</tr>
</tbody>
</table>

3X-280 SunOS 5.6 modified 31 Dec 1996
The following macro is the reverse of \texttt{COLOR\_PAIR(n)}:

\begin{verbatim}
PAIR\_NUMBER(attrs) \quad Returns the pair number associated 
\quad with the \texttt{COLOR\_PAIR(n)} attribute.
\end{verbatim}

### RETURN VALUES

These routines always return 1.

### ATTRIBUTES

See \texttt{attributes(5)} for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

### SEE ALSO

curs\_addch(3X), curs\_addstr(3X), curs\_printw(3X), curses(3X), attributes(5)

### NOTES

The header \texttt{<curses.h>} automatically includes the headers \texttt{<stdio.h>} and \texttt{<unctrl.h>}. Note that \texttt{attroff()}, \texttt{wattroff()}, \texttt{attron()}, \texttt{wattron()}, \texttt{wattrset()}, \texttt{standend()}, and \texttt{standout()} may be macros.
NAME
curs_beep, beep, flash – curses bell and screen flash routines

SYNOPSIS
cc [ flag . . . ] file . . . -lcurses [ library . . ]
#include <curses.h>
int beep(void);
int flash(void);

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.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
curses(3X), attributes(5)

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  
cc [ flag ... ] file ... -lcurses [ library ... ]
#include <curses.h>
int bkgd(chtype ch);
void bkgdset(chtype ch);
void wbkgdset(WINDOW *win, chtype ch);
int wbkgd(WINDOW *win, chtype ch);

DESCRIPTION  
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.

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.

RETURN VALUES  
bkgd() and wbkgd() return the integer OK, or a non-negative integer, if immedok() is set. See curs_outopts(3X).

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO  
curs_addch(3X), curs_outopts(3X), curses(3X), attributes(5)

NOTES  
The header <curses.h> automatically includes the headers <stdio.h> and <unctrl.h>. Note that bkgdset() and bkgd() may be macros.

modified 31 Dec 1996
SunOS 5.6
3X-283
NAME
curs_border, border, wborder, box, whline, wvline – create curses borders, horizontal and vertical lines

SYNOPSIS
cc [ flag ... ] file ... -lcurses [ library ... ]
#include <curses.h>
int border(chtype ls, chtype rs, chtype ts, chtype bs, chtype tl, chtype tr, chtype bl, chtype br);
int wborder(WINDOW *win, chtype ls, chtype rs, chtype ts, chtype bs, chtype tl, chtype tr, chtype bl, chtype br);
int box(WINDOW *win, chtype verch, chtype horch);
int hline(chtype ch, int n);
int whline(WINDOW *win, chtype ch, int n);
int vline(chtype ch, int n);
int wvline(WINDOW *win, chtype ch, int n);

DESCRIPTION
With the border(), wborder(), and box() routines, a border is drawn around the edges of the window. The arguments and attributes are:

ls left side of the border
rs right side of the border
ts top side of the border
bs bottom side of the border
tl top left-hand corner
tr top right-hand corner
bl bottom left-hand corner
br bottom right-hand corner

If any of these arguments is zero, then the following default values (defined in <curses.h>) are used respectively instead: ACS_VLINE, ACS_VLINE, ACS_HLINE, ACS_HLINE, ACS_ULCORNER, ACS_URCORNER, ACS_BLCORNER, ACS_BRCORNER.

box(win, verch, horch) is a shorthand for the following call:

wborder(win, verch, verch, horch, horch, 0, 0, 0, 0)

hline() and whline() draw a horizontal (left to right) line using ch starting at the current cursor position in the window. The current cursor position is not changed. The line is at most n characters long, or as many as fit into the window.

vline() and wvline() draw a vertical (top to bottom) line using ch starting at the current cursor position in the window. The current cursor position is not changed. The line is at most n characters long, or as many as fit into the window.

RETURN VALUES
All routines return the integer OK, or a non-negative integer if immedok() is set. See curs_outopts(3X).
ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
curs_outopts(3X), curses(3X), attributes(5)

NOTES
The header <curses.h> automatically includes the headers <stdio.h> and <unctrl.h>. Note that border() and box() may be macros.
NAME

curs_clear, erase, werase, clear, wclear, clrtobot, wclrtobot, clrtoeol, wclrtoeol – clear all or part of a curses window

SYNOPSIS

cc [flag ...] file ... -lcurses [ library ...]
#include <curses.h>
int erase(void);
int werase(WINDOW *win);
int clear(void);
int wclear(WINDOW *win);
int clrtobot(void);
int wclrtobot (WINDOW *win);
int clrtoeol(void);
int wclrtoeol(WINDOW *win);

DESCRIPTION

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.

The clrtobot() and wclrtobot() routines erase all lines below the cursor in the window. Also, the current line to the right of the cursor, inclusive, is erased.

The clrtoeol() and wclrtoeol() routines erase the current line to the right of the cursor, inclusive.

RETURN VALUES

All routines return the integer OK, or a non-negative integer if immedok() is set. See curs_outopts(3X).

ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO

curs_outopts(3X), curs_refresh(3X), curses(3X), attributes(5)

NOTES

The header <curses.h> automatically includes the headers <stdio.h> and <unctrl.h>. Note that erase(), werase(), clear(), wclear(), clrtobot(), and clrtoeol() may be macros.

3X-286
SunOS 5.6
modified 31 Dec 1996
NAME

curs_color, start_color, init_pair, init_color, has_colors, can_change_color, color_content,
pair_content – curses color manipulation routines

SYNOPSIS

cc [ flag ... ] file ... -lcurses [ library ... ]
#include <curses.h>
int start_color(void);
int init_pair(short pair, short fg, short bg);
int init_color(short color, short red, short green, short blue);
bool has_colors(void);
bool can_change_color(void);
int color_content(short color, short *redp, short *greenp, short *bluep);
int pair_content(short pair, short *fgp, short *bgp);

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 termi-
nal 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 back-
ground 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.

SunOS 5.6

3X-287
The `init_color()` routine changes the definition of a color. It takes four arguments: the number of the color to be changed followed by three RGB values (for the amounts of red, green, and blue components). The value of the first argument must be between 0 and `COLORS`. (See the section Colors for the default color index.) Each of the last three arguments must be a value between 0 and 1000. When `init_color()` is used, all occurrences of that color on the screen immediately change to the new definition.

The `has_colors()` routine requires no arguments. It returns `TRUE` if the terminal can manipulate colors; otherwise, it returns `FALSE`. This routine facilitates writing terminal-independent programs. For example, a programmer can use it to decide whether to use color or some other video attribute.

The `can_change_color()` routine requires no arguments. It returns `TRUE` if the terminal supports colors and can change their definitions; other, it returns `FALSE`. This routine facilitates writing terminal-independent programs.

The `color_content()` routine gives users a way to find the intensity of the red, green, and blue (RGB) components in a color. It requires four arguments: the color number, and three addresses of `short`s for storing the information about the amounts of red, green, and blue components in the given color. The value of the first argument must be between 0 and `COLORS`. The values that are stored at the addresses pointed to by the last three arguments are between 0 (no component) and 1000 (maximum amount of component).

The `pair_content()` routine allows users to find out what colors a given color-pair consists of. It requires three arguments: the color-pair number, and two addresses of `short`s for storing the foreground and the background color numbers. The value of the first argument must be between 1 and `COLOR_PAIRS`—1. The values that are stored at the addresses pointed to by the second and third arguments are between 0 and `COLORS`.

Colors In `<curses.h>` the following macros are defined. These are the default colors. `curses` also assumes that `COLOR_BLACK` is the default background color for all terminals.

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

RETURN VALUES All routines that return an integer return ERR upon failure and OK upon successful completion.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>
SEE ALSO  curs_attr(3X), curs_initscr(3X), curses(3X), attributes(5)

NOTES  The header <curses.h> automatically includes the headers <stdio.h> and <unctrl.h>.
NAME
curs_delch, delch, wdelch, mvdelch, mvwdelch – delete character under cursor in a
curses window

SYNOPSIS
cc [ flag ... ] file ... -lcurses [ library ... ]
#include <curses.h>
int delch(void);
int wdelch(WINDOW *win);
int mvdelch(int y, int x);
int mvwdelch(WINDOW *win, int y, int x);

DESCRIPTION
With these routines the character under the cursor in the window is deleted; all charac-
ters 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 y, x, 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.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
curses(3X), attributes(5)

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  cc [flag ...] file ... -lcurses [ library ... ]
           #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);

DESCRIPTION  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.

With the insdelln() and winsdelln() routines, for positive n, insert n lines into the specified window above the current line. The n bottom lines are lost. For negative n, delete n lines (starting with the one under the cursor), and move the remaining lines up. The bottom n lines are cleared. The current cursor position remains the same.

With the insertln() and insertln() 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.

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

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO  curses(3X), attributes(5)

NOTES  The header <curses.h> automatically includes the headers <stdio.h> and <unctrl.h>. Note that all but winsdelln() may be macros.
NAME  curs_getch, getch, wgetch, mvgetch, mvwgetch, ungetch – get (or push back) characters from curses terminal keyboard

SYNOPSIS  cc [ flag ... ] file ... -lcurses [ library ... ]
#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);

DESCRIPTION  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.

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.

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.

The ungetch() routine places ch back onto the input queue to be returned by the next call to wgetch().

Function Keys  The following function keys, defined in <curses.h>, might be returned by getch() if keypad() has been enabled. Note that not all of these may be supported on a particular terminal if the terminal does not transmit a unique code when the key is pressed or if the definition for the key is not present in the terminfo database.
<table>
<thead>
<tr>
<th>Name</th>
<th>Key name</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEY_BREAK</td>
<td>Break key</td>
</tr>
<tr>
<td>KEY_DOWN</td>
<td>The four arrow keys . . .</td>
</tr>
<tr>
<td>KEY_UP</td>
<td></td>
</tr>
<tr>
<td>KEY_LEFT</td>
<td></td>
</tr>
<tr>
<td>KEY_RIGHT</td>
<td></td>
</tr>
<tr>
<td>KEY_HOME</td>
<td>Home key (upward+left arrow)</td>
</tr>
<tr>
<td>KEY_BACKSPACE</td>
<td>Backspace</td>
</tr>
<tr>
<td>KEY_F0</td>
<td>Function keys; space for 64 keys is reserved.</td>
</tr>
<tr>
<td>KEY_F(n)</td>
<td>For $0 \leq n \leq 63$</td>
</tr>
<tr>
<td>KEY_DL</td>
<td>Delete line</td>
</tr>
<tr>
<td>KEY_IL</td>
<td>Insert line</td>
</tr>
<tr>
<td>KEY_DC</td>
<td>Delete character</td>
</tr>
<tr>
<td>KEY_IC</td>
<td>Insert char or enter insert mode</td>
</tr>
<tr>
<td>KEY_EIC</td>
<td>Exit insert char mode</td>
</tr>
<tr>
<td>KEY_CLEAR</td>
<td>Clear screen</td>
</tr>
<tr>
<td>KEY_EOS</td>
<td>Clear to end of screen</td>
</tr>
<tr>
<td>KEY_EOL</td>
<td>Clear to end of line</td>
</tr>
<tr>
<td>KEY_SF</td>
<td>Scroll 1 line forward</td>
</tr>
<tr>
<td>KEY_SR</td>
<td>Scroll 1 line backward (reverse)</td>
</tr>
<tr>
<td>KEY_NPAGE</td>
<td>Next page</td>
</tr>
<tr>
<td>KEY_PPAGE</td>
<td>Previous page</td>
</tr>
<tr>
<td>KEY_STAB</td>
<td>Set tab</td>
</tr>
<tr>
<td>KEY_CTAB</td>
<td>Clear tab</td>
</tr>
<tr>
<td>KEY_CATAB</td>
<td>Clear all tabs</td>
</tr>
<tr>
<td>KEY_ENTER</td>
<td>Enter or send</td>
</tr>
<tr>
<td>KEY_SRESET</td>
<td>Soft (partial) reset</td>
</tr>
<tr>
<td>KEY_RESET</td>
<td>Reset or hard reset</td>
</tr>
<tr>
<td>KEY_PRINT</td>
<td>Print or copy</td>
</tr>
<tr>
<td>KEY_LH</td>
<td>Home down or bottom (lower left). Keypad is arranged like this:</td>
</tr>
<tr>
<td></td>
<td>A1 up A3</td>
</tr>
<tr>
<td></td>
<td>left B2 right</td>
</tr>
<tr>
<td></td>
<td>C1 down C3</td>
</tr>
<tr>
<td>KEY_A1</td>
<td>Upper left of keypad</td>
</tr>
<tr>
<td>KEY_A3</td>
<td>Upper right of keypad</td>
</tr>
<tr>
<td>KEY_B2</td>
<td>Center of keypad</td>
</tr>
<tr>
<td>KEY_C1</td>
<td>Lower left of keypad</td>
</tr>
<tr>
<td>KEY_C3</td>
<td>Lower right of keypad</td>
</tr>
<tr>
<td>KEY_BTAB</td>
<td>Back tab key</td>
</tr>
<tr>
<td>KEY_BEG</td>
<td>Beg(inning) key</td>
</tr>
<tr>
<td>KEY_CANCEL</td>
<td>Cancel key</td>
</tr>
<tr>
<td>KEY_CLOSE</td>
<td>Close key</td>
</tr>
<tr>
<td>KEY_COMMAND</td>
<td>Cmd (command) key</td>
</tr>
<tr>
<td>KEY_COPY</td>
<td>Copy key</td>
</tr>
<tr>
<td>KEY_CREATE</td>
<td>Create key</td>
</tr>
<tr>
<td>KEY_END</td>
<td>End key</td>
</tr>
<tr>
<td>KEY_EXIT</td>
<td>Exit key</td>
</tr>
<tr>
<td>KEY_FIND</td>
<td>Find key</td>
</tr>
</tbody>
</table>

modified 31 Dec 1996

SunOS 5.6
<table>
<thead>
<tr>
<th>Key Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEY_HELP</td>
<td>Help key</td>
</tr>
<tr>
<td>KEY_MARK</td>
<td>Mark key</td>
</tr>
<tr>
<td>KEY_MESSAGE</td>
<td>Message key</td>
</tr>
<tr>
<td>KEY_MOVE</td>
<td>Move key</td>
</tr>
<tr>
<td>KEY_NEXT</td>
<td>Next object key</td>
</tr>
<tr>
<td>KEY_OPEN</td>
<td>Open key</td>
</tr>
<tr>
<td>KEY_OPTIONS</td>
<td>Options key</td>
</tr>
<tr>
<td>KEY_PREVIOUS</td>
<td>Previous object key</td>
</tr>
<tr>
<td>KEY_REDO</td>
<td>Redo key</td>
</tr>
<tr>
<td>KEY_REFERENCE</td>
<td>Reference key</td>
</tr>
<tr>
<td>KEY_REFRESH</td>
<td>Refresh key</td>
</tr>
<tr>
<td>KEY_REPLACE</td>
<td>Replace key</td>
</tr>
<tr>
<td>KEY_RESTART</td>
<td>Restart key</td>
</tr>
<tr>
<td>KEY_RESUME</td>
<td>Resume key</td>
</tr>
<tr>
<td>KEY_SAVE</td>
<td>Save key</td>
</tr>
<tr>
<td>KEY_SBEG</td>
<td>Shifted beginning key</td>
</tr>
<tr>
<td>KEY_SCANCEL</td>
<td>Shifted cancel key</td>
</tr>
<tr>
<td>KEY_SCOMMAND</td>
<td>Shifted command key</td>
</tr>
<tr>
<td>KEY_SCOPY</td>
<td>Shifted copy key</td>
</tr>
<tr>
<td>KEY_SCREATE</td>
<td>Shifted create key</td>
</tr>
<tr>
<td>KEY_SDC</td>
<td>Shifted delete char key</td>
</tr>
<tr>
<td>KEY(SDL)</td>
<td>Shifted delete line key</td>
</tr>
<tr>
<td>KEY_SELECT</td>
<td>Select key</td>
</tr>
<tr>
<td>KEY_SEND</td>
<td>Shifted end key</td>
</tr>
<tr>
<td>KEY_SEOL</td>
<td>Shifted clear line key</td>
</tr>
<tr>
<td>KEY_SEXIT</td>
<td>Shifted exit key</td>
</tr>
<tr>
<td>KEY_SFIND</td>
<td>Shifted find key</td>
</tr>
<tr>
<td>KEY_SHELP</td>
<td>Shifted help key</td>
</tr>
<tr>
<td>KEY_SHOME</td>
<td>Shifted home key</td>
</tr>
<tr>
<td>KEY_SIC</td>
<td>Shifted input key</td>
</tr>
<tr>
<td>KEY_SLEFT</td>
<td>Shifted left arrow key</td>
</tr>
<tr>
<td>KEY_SMESSAGE</td>
<td>Shifted message key</td>
</tr>
<tr>
<td>KEY_SMOVE</td>
<td>Shifted move key</td>
</tr>
<tr>
<td>KEY_SNEXT</td>
<td>Shifted next key</td>
</tr>
<tr>
<td>KEY_SOPTIONS</td>
<td>Shifted options key</td>
</tr>
<tr>
<td>KEY_SPREVIOUS</td>
<td>Shifted prev key</td>
</tr>
<tr>
<td>KEY_SPRINT</td>
<td>Shifted print key</td>
</tr>
<tr>
<td>KEY_SREDO</td>
<td>Shifted redo key</td>
</tr>
<tr>
<td>KEY_SREPLACE</td>
<td>Shifted replace key</td>
</tr>
<tr>
<td>KEY_SRIGHT</td>
<td>Shifted right arrow</td>
</tr>
<tr>
<td>KEY_SRSUME</td>
<td>Shifted resume key</td>
</tr>
<tr>
<td>KEY_SSAVE</td>
<td>Shifted save key</td>
</tr>
<tr>
<td>KEY_SSUSPEND</td>
<td>Shifted suspend key</td>
</tr>
<tr>
<td>KEY_SUNDO</td>
<td>Shifted undo key</td>
</tr>
<tr>
<td>KEY_SUSPEND</td>
<td>Suspen key</td>
</tr>
<tr>
<td>KEY_UNDO</td>
<td>Undo key</td>
</tr>
</tbody>
</table>
RETURN VALUES
All routines return the integer ERR upon failure. The ungetch() routine returns an integer value other than ERR upon successful completion. The other routines return the next input character or function key code upon successful completion.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
curs_inopts(3X), curs_move(3X), curs_refresh(3X), curses(3X), attributes(5)

NOTES
The header <curses.h> automatically includes the headers <stdio.h> and <unctrl.h>. Use of the escape key for a single character function is discouraged.

When using getch(), wgetch(), mvgetch(), or mvwgetch(), nocbreak mode (nocbreak()) and echo mode (echo()) should not be used at the same time. Depending on the state of the tty driver when each character is typed, the program may produce undesirable results.

Note that getch(), mvgetch(), and mvwgetch() may be macros.
NAME  
curs_getstr, getstr, wgetstr, mvgetstr, mvwgetstr, wgetnstr – get character strings from 
curses terminal keyboard

SYNOPSIS  
cc [ flag ... ] file ... -lcurses [ library ... ]
#include <curses.h>
int getstr(char *str);
int wgetstr(WINDOW *win, char *str);
int 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);

DESCRIPTION  
The effect of getstr() is as though a series of calls to getch() were made, until a newline or 
carriage return is received. The resulting value is placed in the area pointed to by the 
character pointer str. wgetstr() reads at most n characters, thus preventing a possible 
overflow of the input buffer. The user’s erase and kill characters are interpreted, as well 
as any special keys (such as function keys, HOME key, CLEAR key, etc.).

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

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO  
curs_getch(3X), curses(3X), attributes(5)

NOTES  
The header <curses.h> automatically includes the headers <stdio.h> and <unctrl.h>. 
Note that getstr(), mvgetstr(), and mvwgetstr() may be macros.
NAME
curs_getwch, getwch, wgetwch, mvgetwch, mvwgetwch, ungetwch – get (or push back)
wchar_t characters from curses terminal keyboard

SYNOPSIS
cc [flag ...] file ... -lcurses [ library ...]
#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);

DESCRIPTION
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.

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.

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 rea-
son, many terminals experience a delay between the time a user presses the escape key
and the escape is returned to the program.

The ungetwch() routine places wch back onto the input queue to be returned by the next
call to wgetwch().

Function Keys
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 ter-
minal 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.

modified 31 Dec 1996 SunOS 5.6 3X-297
<table>
<thead>
<tr>
<th>Name</th>
<th>Key name</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEY_BREAK</td>
<td>Break key</td>
</tr>
<tr>
<td>KEY_DOWN</td>
<td>The four arrow keys ...</td>
</tr>
<tr>
<td>KEY_UP</td>
<td></td>
</tr>
<tr>
<td>KEY_LEFT</td>
<td></td>
</tr>
<tr>
<td>KEY_RIGHT</td>
<td></td>
</tr>
<tr>
<td>KEY_HOME</td>
<td>Home key (upward+left arrow)</td>
</tr>
<tr>
<td>KEY_BACKSPACE</td>
<td>Backspace</td>
</tr>
<tr>
<td>KEY_F0</td>
<td>Function keys; space for 64 keys is reserved.</td>
</tr>
<tr>
<td>KEY_F(n)</td>
<td>For 0 ≤ n ≤ 63</td>
</tr>
<tr>
<td>KEY_DL</td>
<td>Delete line</td>
</tr>
<tr>
<td>KEY_IL</td>
<td>Insert line</td>
</tr>
<tr>
<td>KEY_DC</td>
<td>Delete character</td>
</tr>
<tr>
<td>KEY_IC</td>
<td>Insert char or enter insert mode</td>
</tr>
<tr>
<td>KEY_EIC</td>
<td>Exit insert char mode</td>
</tr>
<tr>
<td>KEY_CLEAR</td>
<td>Clear screen</td>
</tr>
<tr>
<td>KEY_EOS</td>
<td>Clear to end of screen</td>
</tr>
<tr>
<td>KEY_EOL</td>
<td>Clear to end of line</td>
</tr>
<tr>
<td>KEY_SF</td>
<td>Scroll 1 line forward</td>
</tr>
<tr>
<td>KEY_SR</td>
<td>Scroll 1 line backward (reverse)</td>
</tr>
<tr>
<td>KEY_NPAGE</td>
<td>Next page</td>
</tr>
<tr>
<td>KEY_PPAGE</td>
<td>Previous page</td>
</tr>
<tr>
<td>KEY_STAB</td>
<td>Set tab</td>
</tr>
<tr>
<td>KEY_CTAB</td>
<td>Clear tab</td>
</tr>
<tr>
<td>KEY_CATAB</td>
<td>Clear all tabs</td>
</tr>
<tr>
<td>KEY_ENTER</td>
<td>Enter or send</td>
</tr>
<tr>
<td>KEY_SRESET</td>
<td>Soft (partial) reset</td>
</tr>
<tr>
<td>KEY_RESET</td>
<td>Reset or hard reset</td>
</tr>
<tr>
<td>KEY_PRINT</td>
<td>Print or copy</td>
</tr>
<tr>
<td>KEY_LL</td>
<td>Home down or bottom (lower left). Keypad is arranged like this:</td>
</tr>
<tr>
<td></td>
<td>A1 up A3</td>
</tr>
<tr>
<td></td>
<td>left B2 right</td>
</tr>
<tr>
<td></td>
<td>C1 down C3</td>
</tr>
<tr>
<td>KEY_A1</td>
<td>Upper left of keypad</td>
</tr>
<tr>
<td>KEY_A3</td>
<td>Upper right of keypad</td>
</tr>
<tr>
<td>KEY_B2</td>
<td>Center of keypad</td>
</tr>
<tr>
<td>KEY_C1</td>
<td>Lower left of keypad</td>
</tr>
<tr>
<td>KEY_C3</td>
<td>Lower right of keypad</td>
</tr>
<tr>
<td>KEY_BTAB</td>
<td>Back tab key</td>
</tr>
<tr>
<td>KEY_BEG</td>
<td>Beg(inning) key</td>
</tr>
<tr>
<td>KEY_CANCEL</td>
<td>Cancel key</td>
</tr>
<tr>
<td>KEY_CLOSE</td>
<td>Close key</td>
</tr>
<tr>
<td>KEY_COMMAND</td>
<td>Cmd (command) key</td>
</tr>
<tr>
<td>KEY_COPY</td>
<td>Copy key</td>
</tr>
<tr>
<td>KEY_CREATE</td>
<td>Create key</td>
</tr>
<tr>
<td>KEY_END</td>
<td>End key</td>
</tr>
<tr>
<td>KEY_EXIT</td>
<td>Exit key</td>
</tr>
<tr>
<td>KEY_FIND</td>
<td>Find key</td>
</tr>
</tbody>
</table>

3X-298

SunOS 5.6

modified 31 Dec 1996
<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEY_HELP</td>
<td>Help key</td>
</tr>
<tr>
<td>KEY_MARK</td>
<td>Mark key</td>
</tr>
<tr>
<td>KEY_MESSAGE</td>
<td>Message key</td>
</tr>
<tr>
<td>KEY_MOVE</td>
<td>Move key</td>
</tr>
<tr>
<td>KEY_NEXT</td>
<td>Next object key</td>
</tr>
<tr>
<td>KEY_OPEN</td>
<td>Open key</td>
</tr>
<tr>
<td>KEY_OPTIONS</td>
<td>Options key</td>
</tr>
<tr>
<td>KEY_PREVIOUS</td>
<td>Previous object key</td>
</tr>
<tr>
<td>KEY_REDO</td>
<td>Redo key</td>
</tr>
<tr>
<td>KEY_REFERENCE</td>
<td>Reference key</td>
</tr>
<tr>
<td>KEY_REFRESH</td>
<td>Refresh key</td>
</tr>
<tr>
<td>KEY_REPLACE</td>
<td>Replace key</td>
</tr>
<tr>
<td>KEY_RESTART</td>
<td>Restart key</td>
</tr>
<tr>
<td>KEY_RESUME</td>
<td>Resume key</td>
</tr>
<tr>
<td>KEY_SAVE</td>
<td>Save key</td>
</tr>
<tr>
<td>KEY_SBEG</td>
<td>Shifted beginning key</td>
</tr>
<tr>
<td>KEY_SCANCEL</td>
<td>Shifted cancel key</td>
</tr>
<tr>
<td>KEY_SCOMMAND</td>
<td>Shifted command key</td>
</tr>
<tr>
<td>KEY_SCOPY</td>
<td>Shifted copy key</td>
</tr>
<tr>
<td>KEY_SCREATE</td>
<td>Shifted create key</td>
</tr>
<tr>
<td>KEY_SDC</td>
<td>Shifted delete char key</td>
</tr>
<tr>
<td>KEY(SDL)</td>
<td>Shifted delete line key</td>
</tr>
<tr>
<td>KEY_SELECT</td>
<td>Select key</td>
</tr>
<tr>
<td>KEY_SEND</td>
<td>Shifted end key</td>
</tr>
<tr>
<td>KEY_SEOL</td>
<td>Shifted clear line key</td>
</tr>
<tr>
<td>KEY_SEXIT</td>
<td>Shifted exit key</td>
</tr>
<tr>
<td>KEY_SFIND</td>
<td>Shifted find key</td>
</tr>
<tr>
<td>KEY_SHELP</td>
<td>Shifted help key</td>
</tr>
<tr>
<td>KEY_SHOME</td>
<td>Shifted home key</td>
</tr>
<tr>
<td>KEY_SIC</td>
<td>Shifted input key</td>
</tr>
<tr>
<td>KEY_SLEFT</td>
<td>Shifted left arrow key</td>
</tr>
<tr>
<td>KEY_SMESSAGE</td>
<td>Shifted message key</td>
</tr>
<tr>
<td>KEY_SMOVE</td>
<td>Shifted move key</td>
</tr>
<tr>
<td>KEY_SNEXT</td>
<td>Shifted next key</td>
</tr>
<tr>
<td>KEY_SOPTIONS</td>
<td>Shifted options key</td>
</tr>
<tr>
<td>KEY_SPREVIOUS</td>
<td>Shifted prev key</td>
</tr>
<tr>
<td>KEY_SPRINT</td>
<td>Shifted print key</td>
</tr>
<tr>
<td>KEY_SREDO</td>
<td>Shifted redo key</td>
</tr>
<tr>
<td>KEY_SREPLACE</td>
<td>Shifted replace key</td>
</tr>
<tr>
<td>KEY_SRIGHT</td>
<td>Shifted right arrow</td>
</tr>
<tr>
<td>KEY_SRESUME</td>
<td>Shifted resume key</td>
</tr>
<tr>
<td>KEY_SSSAVE</td>
<td>Shifted save key</td>
</tr>
<tr>
<td>KEY_SSUSPEND</td>
<td>Shifted suspend key</td>
</tr>
<tr>
<td>KEY_SUNDO</td>
<td>Shifted undo key</td>
</tr>
<tr>
<td>KEY_SUSPEND</td>
<td>Suspend key</td>
</tr>
<tr>
<td>KEY_UNDO</td>
<td>Undo key</td>
</tr>
</tbody>
</table>

modified 31 Dec 1996

SunOS 5.6

3X-299
RETURN VALUE
All routines return the integer **ERR** upon failure and an integer value other than **ERR** upon successful completion.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
`curses`(3X), `curs_inopts`(3X), `curs_move`(3X), `wrefresh`(3X), `terminfo`(4), `attributes`(5)

NOTES
The header file `<curses.h>` automatically includes the header files `<stdio.h>`, `<unctrl.h>`, and `<widec.h>.

Use of the escape key by a programmer for a single character function is discouraged. When using `getwch()`, `wgetwch()`, `mvgetwch()`, or `mvwgetwch()`, `nocbreak` mode and echo mode should not be used at the same time. Depending on the state of the tty driver when each character is typed, the program may produce undesirable results.

Note that `getwch()`, `mvgetwch()`, and `mvwgetwch()` may be macros.
NAME
curs_getwstr, getwstr, getnwstr, wgetwstr, wgetnwstr, mvgetwstr, mvgetnwstr,
mvwgetwstr, mvwgetnwstr – get wchar_t character strings from curses terminal keyboard

SYNOPSIS
cc [ flag ... ] file ... -lcurses [ library ... ]
#include <curses.h>

int getwstr(wchar_t *wstr);
int getnwstr(wchar_t *wstr, int n);
int wgetwstr(WINDOW *win, wchar_t *wstr);
int wgetnwstr(WINDOW *win, wchar_t *wstr, int n);
int mvgetwstr(int y, int x, wchar_t *wstr);
int mvgetnwstr(int y, int x, wchar_t *wstr, int n);
int mvwgetwstr(WINDOW *win, int y, int x, wchar_t *wstr);
int mvwgetnwstr(WINDOW *win, int y, int x, wchar_t *wstr, int n);

DESCRIPTION
The effect of getwstr() is as though a series of calls to getwch(3X) were made, until a newline and carriage return is received. The resulting value is placed in the area pointed to by the wchar_t pointer wstr. getnwstr() reads at most n wchar_t characters, thus preventing a possible overflow of the input buffer. The user’s erase and kill characters are interpreted, as well as any special keys (such as function keys, HOME key, CLEAR key, etc.).

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

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-LEVEL</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
curses(3X), getwch(3X), attributes(5)

NOTES
The header file <curses.h> automatically includes the header files <stdio.h>, <unctrl.h>, and <widec.h>.
Note that all routines except wgetnwstr() may be macros.
NAME
curs_getyx, getyx, getparyx, getbegyx, getmaxyx – get curses cursor and window coordinates

SYNOPSIS
c { [ flag ... ] file ... -lcurses [ library ... ]
#include <curses.h>
void getyx(WINDOW *win, int y, int x);
void getparyx(WINDOW *win, int y, int x);
void getbegyx(WINDOW *win, int y, int x);
void getmaxyx(WINDOW *win, int y, int x);

DESCRIPTION
With the getyx() macro, the cursor position of the window is placed in the two integer
variables y and x.

With the getparyx() macro, if win is a subwindow, the beginning coordinates of the
subwindow relative to the parent window are placed into two integer variables, y and x.
Otherwise, −1 is placed into y and x.

Like getyx() , the getbegyx() and getmaxyx() macros store the current beginning coordi-
nates and size of the specified window.

RETURN VALUES
The return values of these macros are undefined (that is, they should not be used as the
right-hand side of assignment statements).

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
 curses(3X), attributes(5)

NOTES
The header <curses.h> automatically includes the headers <stdio.h> and <unctrl.h>.
Note that all of these interfaces are macros and that “&” is not necessary before the vari-
ables y and x.
NAME
curs_inch, inch, winch, mvinch, mvwinch – get a character and its attributes from a
curses window

SYNOPSIS
cc [ flag . . ] file . . -lcurses [ library . . ]
#include <curses.h>
chtype inch(void);
chtype winch(WINDOW *win);
chtype mvinch(int y, int x);
chtype mvwinch(WINDOW *win, int y, int x);

DESCRIPTION
With these routines, the character, of type chtype(), at the current position in the named
window is returned. If any attributes are set for that position, their values are OR-ed into
the value returned. Constants defined in <curses.h> can be used with the logical AND
(&) operator to extract the character or attributes alone.

Attributes
The following bit-masks may be AND-ed with characters returned by winch().

- A_CHARTEXT Bit-mask to extract character
- A_ATTRIBUTES Bit-mask to extract attributes
- A_COLOR Bit-mask to extract color-pair field information

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
curses(3X), attributes(5)

NOTES
The header <curses.h> automatically includes the headers <stdio.h> and <unctrl.h>.
Note that all of these routines may be macros.
NAME
curs_inchstr, inchstr, inchnstr, winchstr, winchnstr, mvinchstr, mvinchnstr, mvwinchstr, mvwinchnstr – get a string of characters (and attributes) from a curses window

SYNOPSIS
cc [ flag ...] file ... -lcurses [ library ... ]
#include <curses.h>

int inchstr(chtype *chstr);
int inchnstr(chtype *chstr, int n);
int winchstr(WINDOW *win, chtype *chstr);
int winchnstr(WINDOW *win, chtype *chstr, int n);
int mvinchstr(int y, int x, chtype *chstr);
int mvinchnstr(int y, int x, chtype *chstr, int n);
int mvwinchstr(WINDOW *win, int y, int x, chtype *chstr);
int mvwinchnstr(WINDOW *win, int y, int x, chtype *chstr, int n);

DESCRIPTION
With these routines, a string of type chtype(), starting at the current cursor position in the named window and ending at the right margin of the window, is returned. The four functions with n as the last argument, return the string at most n characters long. Constants defined in <curses.h> can be used with the & (logical AND) operator to extract the character or the attribute alone from any position in the chstr (see curs_inch(3X)).

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

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
curs_inch(3X), curses(3X), attributes(5)

NOTES
The header <curses.h> automatically includes the headers <stdio.h> and <unctrl.h>. Note that all routines except winchnstr() may be macros.
NAME
curs_initscr, initscr, newterm, endwin, isendwin, set_term, delscreen – curses screen initialization and manipulation routines

SYNOPSIS
cc [flag ...] file ... -lcurses [library ...]
#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);

DESCRIPTION
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.

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 type 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 type 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.

A program should always call endwin() before exiting or escaping from curses mode temporarily. This routine restores tty modes, moves the cursor to the lower left-hand corner of the screen and resets the terminal into the proper non-visual mode. Calling refresh() or doupdate() after a temporary escape causes the program to resume visual mode.

The isendwin() routine returns TRUE if endwin() has been called without any subsequent calls to wrefresh(), and FALSE otherwise.

The set_term() routine is used to switch between different terminals. The screen reference new becomes the new current terminal. The previous terminal is returned by the routine. This is the only routine which manipulates SCREEN pointers; all other routines affect only the current terminal.

modified 31 Dec 1996
SunOS 5.6
3X-305
The `delscreen()` routine frees storage associated with the `SCREEN` data structure. The `endwin()` routine does not do this, so `delscreen()` should be called after `endwin()` if a particular `SCREEN` is no longer needed.

**RETURN VALUES**

`endwin()` returns the integer `ERR` upon failure and `OK` upon successful completion. Routines that return pointers always return `NULL` on error.

**ATTRIBUTES**

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

**SEE ALSO**

`curs_kernel(3X)`, `curs_refresh(3X)`, `curs_slk(3X)`, `curs_util(3X)`, `curses(3X)`, `attributes(5)`

**NOTES**

The header `<curses.h>` automatically includes the headers `<stdio.h>` and `<unctrl.h>`. Note that `initscr()` and `newterm()` may be macros.
NAME
curs_inopts, cbreak, nocbreak, echo, noecho, halfdelay, intrflush, keypad, meta, nodelay,
notimeout, raw, noraw, noqiflush, qiflush, timeout, wtimeout, typeahead – curses terminal
input option control routines

SYNOPSIS
cc [ flag . . . ] file . . . -lcurses [ library . . ]
#include <curses.h>
int cbreak(void);
in int nocbreak(void);
in int echo(void);
in int noecho(void);
in int halfdelay(int tenths);
in int intrflush(WINDOW *win, bool bf);
in int keypad(WINDOW *win, bool bf);
in int meta(WINDOW *win, bool bf);
in int nodelay(WINDOW *win, bool bf);
in int notimeout(WINDOW *win, bool bf);
in int raw(void);
in int noraw(void);
void noqiflush(void);
void qiflush(void);
void timeout(int delay);
void wtimeout(WINDOW *win, int delay);
in int typeahead(int fildes);

DESCRIPTION
The cbreak() and nocbreak() routines put the terminal into and out of cbreak() mode,
respectively. In this mode, characters typed by the user are immediately available to the
program, and erase/kill character-processing is not performed. When out of this mode,
the tty driver buffers the typed characters until a newline or carriage return is typed.
Interrupt and flow control characters are unaffected by this mode. Initially the terminal
may or may not be in cbreak() mode, as the mode is inherited; therefore, a program
should call cbreak() or nocbreak() explicitly. Most interactive programs using curses set
the cbreak() mode.

Note that cbreak() overrides raw(). (See curs_getch(3X) for a discussion of how these
routines interact with echo() and noecho().)

The echo() and noecho() routines control whether characters typed by the user are
echoed by getch() as they are typed. Echoing by the tty driver is always disabled, but initial-
ly getch() is in echo mode, so characters typed are echoed. Authors of most interac-
tive programs prefer to do their own echoing in a controlled area of the screen, or not to
echo at all, so they disable echoing by calling noecho(). (See curs_getch(3X) for a

modified 31 Dec 1996

SunOS 5.6

3X-307
discussion of how these routines interact with \texttt{cbreak()} and \texttt{nocbreak()}.

The \texttt{halfdelay()} routine is used for half-delay mode, which is similar to \texttt{cbreak()} mode in that characters typed by the user are immediately available to the program. However, after blocking for \textit{tenths} tenths of seconds, \texttt{ERR} is returned if nothing has been typed. The value of \textit{tenths} must be a number between 1 and 255. Use \texttt{nocbreak()} to leave half-delay mode.

If the \texttt{intrflush()} option is enabled, (\texttt{bf} is \texttt{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 \texttt{curses} to have the wrong idea of what is on the screen. Disabling (\texttt{bf} is \texttt{FALSE}), the option prevents the flush. The default for the option is inherited from the tty driver settings. The window argument is ignored.

The \texttt{keypad()} option enables the keypad of the user's terminal. If enabled (\texttt{bf} is \texttt{TRUE}), the user can press a function key (such as an arrow key) and \texttt{wgetch()} returns a single value representing the function key, as in \texttt{KEY_LEFT}. If disabled (\texttt{bf} is \texttt{FALSE}), \texttt{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 \texttt{wgetch()} is called. The default value for keypad is \texttt{false}.

Initially, whether the terminal returns 7 or 8 significant bits on input depends on the control mode of the tty driver (see \texttt{termio(7I)}). To force 8 bits to be returned, invoke \texttt{meta(win, TRUE)}. To force 7 bits to be returned, invoke \texttt{meta(win, FALSE)}. The window argument, \texttt{win}, is always ignored. If the terminfo capabilities \texttt{smm} (\texttt{meta_on}) and \texttt{rmm} (\texttt{meta_off}) are defined for the terminal, \texttt{smm} is sent to the terminal when \texttt{meta(win, TRUE)} is called and \texttt{rmm} is sent when \texttt{meta(win, FALSE)} is called.

The \texttt{nodelay()} option causes \texttt{getch()} to be a non-blocking call. If no input is ready, \texttt{getch()} returns \texttt{ERR}. If disabled (\texttt{bf} is \texttt{FALSE}), \texttt{getch()} waits until a key is pressed.

While interpreting an input escape sequence, \texttt{wgetch()} sets a timer while waiting for the next character. If \texttt{notimeout(win, TRUE)} is called, then \texttt{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 \texttt{raw()} and \texttt{noraw()} routines, the terminal is placed into or out of raw mode. Raw mode is similar to \texttt{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 \texttt{curses}.

When the \texttt{noqiflush()} routine is used, normal flush of input and output queues associated with the \texttt{INTR}, \texttt{QUIT} and \texttt{SUSP} characters will not be done (see \texttt{termio(7I)}). When \texttt{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 `fdes` is to be used to check for typeahead instead. If `fdes` is −1, then no typeahead checking is done.

**RETURN VALUES**

All routines that return an integer return `ERR` upon failure and an integer value other than `ERR` upon successful completion, unless otherwise noted in the preceding routine descriptions.

**ATTRIBUTES**

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

**SEE ALSO**

`curs_getch(3X)`, `curs_initscr(3X)`, `curses(3X)`, `attributes(5)`, `termio(7I)`

**NOTES**

The header `<curses.h>` automatically includes the headers `<stdio.h>` and `<unctrl.h>`. Note that `echo()`, `noecho()`, `halfdelay()`, `intrflush()`, `meta()`, `nodelay()`, `notimeout()`, `noqiflush()`, `qiflush()`, `timeout()`, and `wtimeout()` may be macros.
NAME
curs_insch, insch, winsch, mvinsch, mvwinsch – insert a character before the character under the cursor in a curses window

SYNOPSIS
c c [ flag ... ] file ... -lcurses [ library ... ]
#include <curses.h>
int insch(chtype ch);
int winsch(WINDOW *win, chtype ch);
int mvinsch(int y, int x, chtype ch);
int mvwinsch(WINDOW *win, int y, int x, chtype ch);

DESCRIPTION
With these routines, the character ch 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 y, x, 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.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
 curses(3X), attributes(5)

NOTES
The header <curses.h> automatically includes the headers <stdio.h> and <unctrl.h>. Note that insch(), mvinsch(), and mvwinsch() 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
cc [ flag ... ] file ... -lcurses [ library ... ]
#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);

DESCRIPTION
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 y, x, if specified). (This does
not imply use of the hardware insert character feature.) The four routines with
n as the
last argument insert at most n characters. If n<=0, then the entire string is inserted.

If a character in str 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 str is another control char-
acter, it is drawn in the \X notation. Calling winch() after adding a control character
(and moving to it, if necessary) does not return the control character, but instead returns
the representation of the control character.

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

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
curs_clear(3X), curs_inch(3X), curses(3X), attributes(5)

NOTES
The header <curses.h> automatically includes the headers <stdio.h> and <unctrl.h>.
Note that all but winsnstr() may be macros.
NAME
curs_instr, instr, innstr, winstr, winnstr, mvinstr, mvinnstr, mvwinstr, mvwinnstr – get a string of characters from a curses window

SYNOPSIS
cc [ flag ... ] file ... -lcurses [ library ... ]
#include <curses.h>
int instr(char *str);
int innstr(char *str, int n);
int winstr(WINDOW *win, char *str);
int winnstr(WINDOW *win, char *str, int n);
int mvinstr(int y, int x, char *str);
int mvinnstr(int y, int x, char *str, int n);
int mvwinstr(WINDOW *win, int y, int x, char *str);
int mvwinnstr(WINDOW *win, int y, int x, char *str, int n);

DESCRIPTION
These routines return a string of characters in str, starting at the current cursor position in the named window and ending at the right margin of the window. Attributes are stripped from the characters. The four functions with n as the last argument return the string at most n characters long.

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

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
curses(3X), attributes(5)

NOTES
The header <curses.h> automatically includes the headers <stdio.h> and <unctrl.h>. Note that all routines except winnstr() may be macros.
NAME
curs_inswch, inswch, winswch, mvinswch, mvwinswch – insert a wchar_t character before the character under the cursor in a curses window

SYNOPSIS
cc [ flag ... ] file ... -lcurses [ library ... ]
#include <curses.h>

int inswch(chtype wch);
int winswch(WINDOW *win, chtype wch);
int mvinswch(int y, int x, chtype wch);
int mvwinswch(WINDOW *win, int y, int x, chtype wch);

DESCRIPTION
These routines insert the character wch, holding a wchar_t 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 y, x, if specified). (This does not imply use of the hardware insert character feature.)

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

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
curses(3X), attributes(5)

NOTES
The header file <curses.h> automatically includes the header files <stdio.h>, <unctrl.h> and <widec.h>.
Note that inswch(), mvinswch(), and mvwinswch() may be macros.
None of these routines can use the color attribute in chtype.
NAME
curs_inswstr, inswstr, insnwstr, winswstr, winsnwstr, mvinswstr, mvinsnwstr,
mvwinswstr, mvwinsnwstr — insert wchar_t string before character under the cursor in a
curses window

SYNOPSIS
cc [ flag … ] file … |lcurses [ library … ]
#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);

DESCRIPTION
These routines insert a wchar_t character string (as many wchar_t 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 y, x, if specified). (This does
not imply use of the hardware insert character feature.) The four routines with n as the
last argument insert at most n wchar_t characters. If n<=0, then the entire string is
inserted.

If a character in wstr is a tab, newline, carriage return, or backspace, the cursor is moved
appropriately within the window. A newline also does a clrtoeol(3X) before moving.
Tabs are considered to be at every eighth column. If a character in wstr is another control
character, it is drawn in the "X notation. Calling winwch(3X) after adding a control char-
acter (and moving to it, if necessary) does not return the control character, but instead
returns the representation of the control character.

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

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
clrtoeol(3X), curses(3X), winwch(3X), attributes(5)

NOTES
The header file <curses.h> automatically includes the header files <stdio.h>, <unctrl.h> and <widec.h>.
Note that all but winsnwstr() may be macros.
NAME
curs_inwch, inwch, winwch, mvinwch, mvwinwch – get a wchar_t character and its attributes from a curses window

SYNOPSIS
cc [ flag ... ] file ... -lcurses [ library ... ]
#include <curses.h>

chtype inwch(void);
chtype winwch(WINDOW *win);
chtype mvinwch(int y, int x);
chtype mvwinwch(WINDOW *win, int y, int x);

DESCRIPTION
These routines return the wchar_t character, of type ctype, at the current position in the named window. If any attributes are set for that position, their values are OR-ed into the value returned. Constants defined in <curses.h> can be used with the logical AND (&) operator to extract the character or attributes alone.

Attributes
The following bit-masks may be AND-ed with characters returned by winwch().

A_WCHARTEXT  Bit-mask to extract character
A_WATTRIBUTES Bit-mask to extract attributes

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
curses(3X), attributes(5)

NOTES
The header file <curses.h> automatically includes the header files <stdio.h>, <unctrl.h>, and <widec.h>.
Note that all of these routines may be macros.
None of these routines can use the color attribute in ctype.
NAME  
curs_inwchstr, inwchstr, inwchnstr, winwchstr, winwchnstr, mvinwchstr, 
mvinwchnstr, mvwinwchstr, mvwinwchnstr – get a string of wchar_t characters (and attributes) 
from a curses window

SYNOPSIS  
cc [ flag ... ] file ... -lcurses [ library ... ]
#include <curses.h>

int inwchstr(chtype *wchstr);
int inwchnstr(chtype *wchstr, int n);
int winwchstr(WINDOW *win, chtype *wchstr);
int winwchnstr(WINDOW *win, chtype *wchstr, int n);
int mvinwchstr(int y, int x, chtype *wchstr);
int mvinwchnstr(int y, int x, chtype *wchstr, int n);
int mvwinwchstr(WINDOW *win, int y, int x, chtype *wchstr);
int mvwinwchnstr(WINDOW *win, int y, int x, chtype *wchstr, int n);

DESCRIPTION  
These routines return a string of type chtype, holding wchar_t characters, starting at 
the current cursor position in the named window and ending at the right margin of the win-
dow. The four functions with n as the last argument, return the string at most n wchar_t 
characters long. Constants defined in <curses.h> can be used with the logical AND (&) 
operator to extract the wchar_t character or the attribute alone from any position in the 
wchstr (see curs_inwch(3X)).

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

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-LEVEL</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO  
curses(3X), curs_inwch(3X), attributes(5)

NOTES  
The header file <curses.h> automatically includes the header files <stdio.h>, <unctrl.h>,
and <widec.h>.
Note that all routines except winwchnstr() may be macros.
None of these routines can use the color attribute in chtype.

modified 31 Dec 1996  SunOS 5.6  3X-317
NAME
curs_inwstr, inwstr, innwstr, winwstr, winnwwstr, mvinwstr, mvinnwstr, mvwinwstr,
mvinnwstr – get a string of wchar_t characters from a curses window

SYNOPSIS
c c [ flag . . . ] file . . . -lcurses [ library . . ]
#include <curses.h>

```c
int inwstr(wchar_t *wstr);
int innwstr(wchar_t *wstr, int n);
int winwstr(WINDOW *win, wchar_t *wstr);
int winnwwstr(WINDOW *win, wchar_t *wstr, int n);
int mvinwstr(int y, int x, wchar_t *wstr);
int mvinnwstr(int y, int x, wchar_t *wstr, int n);
int mvwinwstr(WINDOW *win, int y, int x, wchar_t *wstr);
int mvwinnwstr(WINDOW *win, int y, int x, wchar_t *wstr, int n);
```

DESCRIPTION
These routines return the string of wchar_t characters in wstr starting at the current cursor position in the named window and ending at the right margin of the window. Attributes are stripped from the characters. The four functions with n as the last argument return the string at most n wchar_t characters long.

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

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
curses(3X), attributes(5)

NOTES
The header file <curses.h> automatically includes the header files <stdio.h>, <unctrl.h> and <widec.h>.

Note that all routines except winnwwstr() 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
cc [flag ...] file ... -lcurses [library ...]
#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);

DESCRIPTION
The following routines give low-level access to various curses functionality. Theses rou-
tines typically are used inside library routines.
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().
The reset_prog_mode() and reset_shell_mode() routines restore the terminal to “pro-
gram” (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.
The resetty() and savetty() routines save and restore the state of the terminal modes.
savetty() saves the current state in a buffer and resetty() restores the state to what it was
at the last call to savetty().

With the getsyx() routine, the current coordinates of the virtual screen cursor are
returned in y and x. If leaveok() is currently TRUE, then −1,−1 is returned. If lines have
been removed from the top of the screen, using ripoffline(), y and x include these lines;
therefore, y and x should be used only as arguments for setsyx().

With the setsyx() routine, the virtual screen cursor is set to y, x. If y and x are both −1,
then leaveok() is set. The two routines getsyx() and setsyx() are designed to be used by
a library routine, which manipulates curses windows but does not want to change the
current position of the program’s cursor. The library routine would call getsyx() at
the beginning, do its manipulation of its own windows, do a wnoutrefresh() on its windows,
call setsyx(), and then call doupdate().

modified 31 Dec 1996
SunOS 5.6
3X-319
The `ripofline()` routine provides access to the same facility that `slk_init()` (see `curs_slk(3X)`) uses to reduce the size of the screen. `ripofline()` 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.

`ripofline()` can be called up to five times before calling `initscr()` or `newterm()`.

With the `curs_set()` routine, the cursor state is set to invisible, normal, or very visible for `visibility` equal to 0, 1, or 2 respectively. If the terminal supports the `visibility` requested, the previous cursor state is returned; otherwise, `ERR` is returned.

The `napms()` routine is used to sleep for `ms` milliseconds.

### RETURN VALUES

Except for `curs_set()`, these routines always return `OK`. `curs_set()` returns the previous cursor state, or `ERR` if the requested `visibility` is not supported.

### ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

### SEE ALSO

`curs_initscr(3X), curs_outopts(3X), curs_refresh(3X), curs_scr_dump(3X), curs_slk(3X), curses(3X), attributes(5)`

### NOTES

The header `<curses.h>` automatically includes the headers `<stdio.h>` and `<unctrl.h>.

Note that `getsyx()` is a macro, so an ampersand (`&`) is not necessary before the variables `y` and `x`. 

3X-320

SunOS 5.6

modified 31 Dec 1996
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);

DESCRIPTION
With these routines, the cursor associated with the window is moved to line y and
column x. This routine does not move the physical cursor of the terminal until refresh() is
called. The position specified is relative to the upper left-hand corner of the window,
which is (0,0).

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

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
curs_refresh(3X), curses(3X), attributes(5)

NOTES
The header <curses.h> automatically includes the headers <stdio.h> and <unctrl.h>.
Note that move() may be a macro.
NAME
curs_outopts, clearok, idlok, idcok, immedok, leaveok, setscrreg, wsetscrreg, scrollok, nl, nonl – curses terminal output option control routines

SYNOPSIS
cc [flag...] file ... -lcurses [library...]
#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 setscrreg(int top, int bot);
int wsetscrreg(WINDOW *win, int top, int bot);
int scrollok(WINDOW *win, bool bf);
int nl(void);
int nonl(void);

DESCRIPTION
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().

With the clearok() routine, if enabled (bf 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 win 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.

With the idlok() routine, if enabled (bf is TRUE), curses considers using the hardware insert/delete line feature of terminals so equipped. If disabled (bf 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.

With the idcok() routine, if enabled (bf is TRUE), curses considers using the hardware insert/delete character feature of terminals so equipped. This is enabled by default.

With the immedok() routine, if enabled (bf is TRUE), any change in the window image, such as the ones caused by waddch(), wclrtobot(), wscrl(), etc., automatically cause a call to wrefresh(). However, it may degrade the performance considerably, due to repeated calls to wrefresh(). It is disabled by default. Normally, the hardware cursor is left at the location of the window cursor being refreshed. The leaveok() option allows the cursor to be left wherever the update happens to leave it. It is useful for applications...
where the cursor is not used, since it reduces the need for cursor motions. If possible, the cursor is made invisible when this option is enabled.

The `setscreg()` and `wsetscreg()` 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**

`setscreg()` and `wsetscreg()` return `OK` upon success and `ERR` upon failure. All other routines that return an integer always return `OK`.

**ATTRIBUTES**

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

**SEE ALSO**

curs_addch(3X), curs_clear(3X), curs_initscr(3X), curs_refresh(3X), curs_scroll(3X), curses(3X), attributes(5)

**NOTES**

The header `<curses.h>` automatically includes the headers `<stdio.h>` and `<unctrl.h>.

Note that `clearok()`, `leaveok()`, `scrollok()`, `idcok()`, `nl()`, `nonl()`, and `setscreg()` may be macros.

The `immedok()` routine is useful for windows that are used as terminal emulators.

modified 31 Dec 1996

SunOS 5.6

3X-323
NAME
curs_overlay, overlay, overwrite, copywin – overlap and manipulate overlapped curses windows

SYNOPSIS
c c [ flag ... ] file ... -lcurses [ library ... ]
#include <curses.h>
int overlay(WINDOW *srcwin, WINDOW *dstwin);
int overwrite(WINDOW *srcwin, WINDOW *dstwin);
int copywin(WINDOW *srcwin, WINDOW *dstwin, int sminrow, int smincol, int dminrow,
int dmincol, int dmaxrow, int dmaxcol, int overlay);

DESCRIPTION
The overlay() and overwrite() routines overlay srcwin on top of dstwin. srcwin and dstwin are not required to be the same size; only text where the two windows overlap is copied. The difference is that overlay() is non-destructive (blanks are not copied) whereas overwrite() is destructive.

The copywin() routine provides a finer granularity of control over the overlay() and overwrite() routines. Like in the refresh() routine, a rectangle is specified in the destination window, (dminrow, dmincol) and (dmaxrow, dmaxcol), and the upper-left-corner coordinates of the source window, (sminrow, smincol). If the argument overlay is true, then copying is non-destructive, as in overlay().

RETURN VALUES
Routines that return an integer return ERR upon failure and an integer value other than ERR upon successful completion.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
curs_pad(3X), curs_refresh(3X), curses(3X), attributes(5)

NOTES
The header <curses.h> automatically includes the headers <stdio.h> and <unctrl.h>. Note that overlay() and overwrite may be macros.
NAME
curs_pad, newpad, subpad, prefresh, pnoutrefresh, pechochar, pechowchar – create and display curses pads

SYNOPSIS
c ( [ flag . . . ] file . . . ] - ]lcurses [ library . . ]
#include <curses.h>

WINDOW *newpad(int nlines, int ncols);
WINDOW *subpad(WINDOW *orig, int nlines, int ncols, int begin_y, int begin_x);
int prefresh(WINDOW *pad, int pminrow, int pmincol, int sminrow, int smincol,
            int smaxrow, int smaxcol);
int pnoutrefresh(WINDOW *pad, int pminrow, int pmincol, int sminrow, int smincol,
                int smaxrow, int smaxcol);
int pechochar(WINDOW *pad, chtype ch);
int pechowchar(WINDOW *pad, chtype wch);

DESCRIPTION
The newpad() routine creates and returns a pointer to a new pad data structure with the given number of lines, nlines, and columns, ncols. 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 pad 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.

The subpad() routine creates and returns a pointer to a subwindow within a pad with the given number of lines, nlines, and columns, ncols. Unlike subwin(3X), which uses screen coordinates, the window is at position (begin_x, begin_y) on the pad. The window is made in the middle of the window orig, 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 orig before calling prefresh().

The prefresh() and pnoutrefresh() routines are analogous to wrefresh(3X) and wnoutrefresh(3X) except that they relate to pads instead of windows. The additional parameters are needed to indicate what part of the pad and screen are involved. pminrow and pmincol specify the upper left-hand corner of the rectangle to be displayed in the pad. sminrow, smincol, smaxrow, and smaxcol specify the edges of the rectangle to be displayed on the screen. The lower right-hand corner of the rectangle to be displayed in the pad is calculated from the screen coordinates, since the rectangles must be the same size. Both rectangles must be entirely contained within their respective structures. Negative values of pminrow, pmincol, sminrow, or smincol are treated as if they were zero.

The pechochar() routine is functionally equivalent to a call to addch(3X) followed by a call to refresh(3X), a call to waddch(3X) followed by a call to wrefresh(3X), or a call to waddch(3X) followed by a call to prefresh(). The knowledge that only a single character is being output is taken into consideration and, for non-control characters, a considerable
performance gain might be seen by using these routines instead of their equivalents. In the case of `pechochar()`, the last location of the pad on the screen is reused for the arguments to `prefresh()`.

**RETURN VALUES**
Routines that return an integer return `ERR` upon failure and an integer value other than `ERR` upon successful completion.
Routines that return pointers return `NULL` on error.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

**SEE ALSO**
`addch(3X), curses(3X), refresh(3X), subwin(3X), touchline(3X), touchwin(3X), waddch(3X), wnoutrefresh(3X), wrefresh(3X), attributes(5)`

**NOTES**
The header `<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
cc [ flag ... ] file ... -lcurses [ library ... ]
#include <curses.h>
int printw(char *fmt, /* arg */ ...);
int wprintw(WINDOW *win, char *fmt, /* arg */ ...);
int mvprintw(int y, int x, char *fmt, /* arg */ ...);
int mvwprintw(WINDOW *win, int y, int x, char *fmt, /* arg */ ...);
#include <varargs.h>
int vwprintw(WINDOW *win, char *fmt, /* varglist */ ...);

DESCRIPTION
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.

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>.

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

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
curses(3X), printf(3S), vprintf(3S), attributes(5)

NOTES
The header <curses.h> automatically includes the headers <stdio.h> and <unctrl.h>.
NAME
curs_refresh, refresh, wrefresh, wnoutrefresh, doupdate, redrawwin, wredrawln – refresh curses windows and lines

SYNOPSIS
c [ flag ... ] file ... -lcurses [ library ... ]
#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);

DESCRIPTION
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.

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.

The wrefresh() routine 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 win argument to wrefresh() is the global variable curscr, the screen is immediately cleared and repainted from scratch.

The redrawwin() routine indicates to curses that some screen lines are corrupted and should be thrown away before anything is written over them. These routines could be used for programs such as editors, which want a command to redraw some part of the screen or the entire screen. The routine redrawln() is preferred over redrawwin() where a noisy communication line exists and redrawing the entire window could be subject to even more communication noise. Just redrawing several lines offers the possibility that they would show up unblemished.
RETURN VALUES: All routines return the integer ERR upon failure and an integer value other than ERR upon successful completion.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO: curs_outopts(3X), curses(3X), attributes(5)

NOTES: The header <curses.h> automatically includes the headers <stdio.h> and <unctrl.h>. Note that refresh() and redrawwin() may be macros.
NAME
curs_scanw, scanw, wscanw, mvscanw, mvwscanw, vwscanw – convert formatted input from a curses widow

SYNOPSIS
cc [ flag ... ] file ... -lcurses [ library ... ]
#include <curses.h>
int scanw(char *fmt, /* arg */ ...);
int wscanw(WINDOW *win, char *fmt, /* arg */ ...);
int mvscanw(int y, int x, char *fmt, /* arg */ ...);
int mvwscanw(WINDOW *win, int y, int x, char *fmt, /* arg */ ...);
int vwscanw(WINDOW *win, char *fmt, va_list varglist);

DESCRIPTION
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 fmt
field are lost.
The vwscanw() routine is similar to vwprintw() in that it performs a wscanw() using a
variable argument list. The third argument is a va_list, a pointer to a list of arguments, as
declared in <varargs.h>.

RETURN VALUES
vwscanw() returns ERR on failure and an integer equal to the number of fields scanned
on success.
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.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
curs_getstr(3X), curs_printw(3X), curses(3X), scanf(3S), attributes(5)

NOTES
The header <curses.h> automatically includes the headers <stdio.h> and <unctrl.h>.

3X-330 SunOS 5.6 modified 31 Dec 1996
NAME    curs_scr_dump, scr_dump, scr_restore, scr_init, scr_set – read (write) a curses screen from (to) a file

SYNOPSIS    cc [ flag ... ] file ... -lcurses [ library ... ]
#include <curses.h>
int scr_dump(char *filename);
int scr_restore(char *filename);
int scr_init(char *filename);
int scr_set(char *filename);

DESCRIPTION    With the scr_dump() routine, the current contents of the virtual screen are written to the file filename.

With the scr_restore() routine, the virtual screen is set to the contents of filename, 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.

With the scr_init() routine, the contents of filename 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.

The scr_set() routine is a combination of scr_restore() and scr_init(). It tells the program
that the information in filename is what is currently on the screen, and also what the pro-
gram wants on the screen. This can be thought of as a screen inheritance function.
To read (write) a window from (to) a file, use the getwin() and putwin() routines (see
curs_util(3X)).

RETURN VALUES    All routines return the integer ERR upon failure and OK upon success.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO    curs_initscr(3X), curs_refresh(3X), curs_util(3X), curses(3X), system(3S), attributes(5)

NOTES    The header <curses.h> automatically includes the headers <stdio.h> and <unctrl.h>.
Note that scr_init(), scr_set(), and scr_restore() may be macros.
cursscroll (3X)  Miscellaneous Library Functions

NAME     cursscroll, scroll, scrl, wscrl – scroll a curses window

SYNOPSIS  cc [ flag ... ] file ... -lcurses [ library ... ]
#include <curses.h>
int scroll(WINDOW *win);
int scrl(int n);
int wscrl(WINDOW *win, int n);

DESCRIPTION  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.

With the scrl() and wscrl() routines, for positive n scroll the window up n lines (line i+n
becomes i); otherwise scroll the window down n lines. This involves moving the lines in
the window character image structure. The current cursor position is not changed.

For these functions to work, scrolling must be enabled via scrollok().

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

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO  curssoutopts(3X), curses(3X), attributes(5)

NOTES  The header <curses.h> automatically includes the headers <stdio.h> and <unctrl.h>.
Note that scrl() and scroll() may be macros.
### NAME

curs_set – set visibility of cursor

### SYNOPSIS

```c
#include <curses.h>

int curs_set(int visibility);
```

### ARGUMENTS

- `visibility` Is a value of 0 (invisible), 1 (normal), or 2 (very visible).

### DESCRIPTION

The `curs_set()` function sets the visibility of the cursor to invisible (0), normal (1), or very visible (2). The exact appearance of normal and very visible cursors is terminal dependent.

### RETURN VALUES

If the terminal supports the mode specified by the `visibility` parameter, the `curs_set()` function returns the previous cursor state. Otherwise, it returns `ERR`.

### ERRORS

None.
NAME
curs_slk, slk_init, slk_set, slk_refresh, slk_noutrefresh, slk_label, slk_clear, slk_restore,
slk_touch, slk_attron, slk_attrset, slk_attroff – curses soft label routines

SYNOPSIS
cc [ flag . . . ] file . . . -lcurses [ library . . . ]
#include <curses.h>
int slk_init(int fmt);
int slk_set(int labnum, char *label, int fmt);
int slk_refresh(void);
int slk_noutrefresh(void);
char *slk_label(int labnum);
int slk_clear(void);
int slk_restore(void);
int slk_touch(void);
int slk_attron(chtype attrs);
int slk_attrset(chtype attrs);
int slk_attroff(chtype attrs);

DESCRIPTION
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.

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 fmt
determines how the labels are arranged on the screen. Setting fmt to 0 indicates a 3-2-3
arrangement of the labels; 1 indicates a 4-4 arrangement.

With the slk_set() routine, labnum is the label number, from 1 to 8. label 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. fmt is either 0, 1, or 2, indicating whether the label is to be left-justified, cen-
tered, or right-justified, respectively, within the label.

The slk_refresh() and slk_noutrefresh() routines correspond to the wrefresh() and
wnoutrefresh() routines.

With the slk_label() routine, the current label for label number labnum is returned with
leading and trailing blanks stripped.

With the slk_clear() routine, the soft labels are cleared from the screen.

With the slk_restore() routine, the soft labels are restored to the screen after a slk_clear()
is performed.

With the slk_touch() routine, all the soft labels are forced to be output the next time a
slk_noutrefresh() is performed.
The slk_attron(), slk_attrset(), and slk_attroff() routines correspond to attron(),
attrset(), and attroff(). They have an effect only if soft labels are simulated on the bottom line of the screen.

RETURN VALUES
Routines that return an integer return ERR upon failure and an integer value other than
ERR upon successful completion.
slk_label() returns NULL on error.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO	curs_attr(3X), curs_initscr(3X), curs_refresh(3X), curses(3X), attributes(5)

NOTES
The header <curses.h> automatically includes the headers <stdio.h> and <unctrl.h>.
Most applications would use slk_noutrefresh() because a wrefresh() is likely to follow soon.

modified 31 Dec 1996	SunOS 5.6	3X-335
NAME  
curs_termattrs, baudrate, erasechar, has_ic, has_il, killchar, longname, termattrs, term-
name – curses environment query routines

SYNOPSIS  
cc [ flag ... ] file ... -lcurses [ library ... ]
#include <curses.h>
int baudrate(void);
char erasechar(void);
int has_ic(void);
int has_il(void);
char killchar(void);
char *longname(void);
ctype termattrs(void);
char *termname(void);

DESCRIPTION  
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.
With the erasechar() routine, the user’s current erase character is returned.
The has_ic() routine is true if the terminal has insert- and delete-character capabilities.
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().
With the killchar() routine, the user’s current line kill character is returned.
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.
If a given terminal doesn’t support a video attribute that an application program is trying
to use, curses may substitute a different video attribute for it. The termattrs() function
returns a logical OR of all video attributes supported by the terminal. This information is
useful when a curses program needs complete control over the appearance of the screen.
The termname() routine returns the value of the environment variable TERM (truncated
to 14 characters).

RETURN VALUES  
longname() and termname() return NULL on error.
Routines that return an integer return ERR upon failure and an integer value other than
ERR upon successful completion.
ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO `curs_initscr(3X)`, `curs_outopts(3X)`, `curses(3X)`, `attributes(5)`

NOTES

The header `<curses.h>` automatically includes the headers `<stdio.h>` and `<unctrl.h>.
Note that `termattr()` may be a macro.
NAME
curs_termcap, tgetent, tgetflag, tgetnum, tgetstr, tgoto, tputs – curses interfaces (emulated) to the termcap library

SYNOPSIS
cc [ flag ...] file ... -lcurses [ library ... ]
#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));

DESCRIPTION
These routines are included as a conversion aid for programs that use the termcap library. Their parameters are the same and the routines are emulated using the terminfo database. These routines are supported at Level 2 and should not be used in new applications.

The tgetent() routine looks up the termcap entry for name. The emulation ignores the buffer pointer bp.

The tgetflag() routine gets the boolean entry for id.

The tgetnum() routine gets the numeric entry for id.

The tgetstr() routine returns the string entry for id. Use tputs() to output the returned string.

The tgoto() routine instantiates the parameters into the given capability. The output from this routine is to be passed to tputs().

The tputs() routine is described on the curs_terminfo(3X) manual page.

RETURN VALUES
Routines that return an integer return ERR upon failure and an integer value other than ERR upon successful completion.
Routines that return pointers return NULL on error.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
curs_terminfo(3X), curses(3X), putc(3S), attributes(5)

NOTES
The header <curses.h> automatically includes the headers <stdio.h> and <unctrl.h>.
NAME
curs_terminfo, setupterm, setterm, set_curterm, del_curterm, restartterm, tparm, tputs,
putp, vidputs, vidattr, mvcur, tigetflag, tigetnum, tigetstr – curses interfaces to terminfo
database

SYNOPSIS
c c [ flag ... ] file ... -lcurses [ library ... ]
#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));
in
nt putp(char *str);
int vidputs(chtype attrs, int (*putc)(char));
in
nt vidattr(chtype attrs);
int mvcur(int oldrow, int oldcol, int newrow, int newcol);
int tigetflag(char *capname);
in
nt tigetnum(char *capname);
char *tigetstr(char *capname);

DESCRIPTION
These low-level routines must be called by programs that have to deal directly with the
terminfo database to handle certain terminal capabilities, such as programming function
keys. For all other functionality, curses routines are more suitable and their use is recom-
mended.

Initially, setupterm() should be called. Note that setupterm() is automatically called by
initscr() and newterm(). This defines the set of terminal-dependent variables (listed in
terminfo(4)). The terminfo variables lines and columns are initialized by setupterm() as
follows: If use_env(FALSE) has been called, values for lines and columns specified in ter-
minfo are used. Otherwise, if the environment variables LINES and COLUMNS exist, their
values are used. If these environment variables do not exist and the program is running
in a window, the current window size is used. Otherwise, if the environment variables
do not exist, the values for lines and columns specified in the terminfo database are used.

The headers <curses.h> and <term.h> should be included (in this order) to get the
definitions for these strings, numbers, and flags. Parameterized strings should be passed
through tparm() to instantiate them. All terminfo strings (including the output of
tparm()) should be printed with tputs() or putp(). Call the reset_shell_mode() routine
to restore the tty modes before exiting (see curs_kernel(3X)). Programs which use cursor
modified 31 Dec 1996
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 `fd` which is initialized for output. If `err` is not null, then `setupterm()` returns `OK` or `ERR` and stores a status value in the integer pointed to by `err`. A status of 1 in `err` is normal, 0 means that the terminal could not be found, and -1 means that the `terminfo` database could not be found. If `err` 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()`. `aff` 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 `files` 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.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

**SEE ALSO**
curs_initscr(3X), curs_kernel(3X), curs_termcap(3X), curses(3X), putc(3S), terminfo(4), attributes(5)

**NOTES**
The header `<curses.h>` automatically includes the headers `<stdio.h>` and `<unctrl.h>`.
The `setupterm()` routine should be used in place of `setterm()`.

Note that `vidattr()` and `vidputs()` may be macros.
NAME  
curs_touch, touchwin, touchline, untouchwin, wtouchln, is_linetouched, is_wintouched –
curses refresh control routines

SYNOPSIS  
cc [ flag … ] file … -lcurses [ library … ]
#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);

DESCRIPTION  
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 count lines have been changed, beginning with line start.
The untouchwin() routine marks all lines in the window as unchanged since the last call
to wrefresh().
The wtouchln() routine makes n lines in the window, starting at line y, look as if they
have (changed=1) or have not (changed=0) been changed since the last call to wrefresh().
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 line is not valid for the given window.

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 descrip-
tions.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO  
curs_refresh(3X), curses(3X), attributes(5)

NOTES  
The header <curses.h> automatically includes the headers <stdio.h> and <unctrl.h>.
Note that all routines except wtouchln() may be macros.

3X-342  
SunOS 5.6  
modified 31 Dec 1996
NAME  
curs_util, unctrl, keyname, filter, use_env, putwin, getwin, delay_output, flushinp –
curses miscellaneous utility routines

SYNOPSIS  
cc [ flag ... ] file ... -lcurses [ library ... ]

#include <curses.h>

char *unctrl(chtype c);
char *keyname(int c);
int filter(void);
void use_env(char bool);
int putwin(WINDOW *win, FILE *lep);
WINDOW *getwin(FILE *lep);
int delay_output(int ms);
int flushinp(void);

DESCRIPTION  
The unctrl() macro expands to a character string which is a printable representation of
the character c. Control characters are displayed in the ^X notation. Printing characters
are displayed as is.

With the keyname() routine, a character string corresponding to the key c is returned.
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.

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 termi-
info 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).

With the putwin() routine, all data associated with window win is written into the file to
which lep points. This information can be later retrieved using the getwin() function.
The getwin() routine reads window related data stored in the file by putwin(). The rou-
tine then creates and initializes a new window using that data. It returns a pointer to the
new window.

The delay_output() routine inserts an ms millisecond pause in output. This routine
should not be used extensively because padding characters are used rather than a CPU
pause.

The flushinp() routine throws away any typeahead that has been typed by the user and
has not yet been read by the program.

modified 31 Dec 1996  SunOS 5.6  3X-343
RETURN VALUES

Except for `flushinp()`, routines that return an integer return `ERR` upon failure and an integer value other than `ERR` upon successful completion.

`flushinp()` always returns `OK`.

Routines that return pointers return `NULL` on error.

ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO  `curs_initscr(3X), curs_scr_dump(3X), curses(3X), attributes(5)`

NOTES

The header `<curses.h>` automatically includes the headers `<stdio.h>` and `<unctrl.h>.

Note that `unctrl()` is a macro, which is defined in `<unctrl.h>`.
NAME  
curs_window, newwin, delwin, mvwin, subwin, derwin, mvderwin, dupwin, wsyncup, 
syncok, wcursyncup, wsyncdown – create curses windows

SYNOPSIS  
cc [ flag ... ] file ... -lcurses [ library ... ]
#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);

DESCRIPTION  
The newwin() routine creates and returns a pointer to a new window with the given
number of lines, nlines, and columns, ncols. The upper left-hand corner of the window is
at line begin_y, column begin_x. If either nlines or ncols is zero, they default to
LINES — begin_y and COLS — begin_x. A new full-screen window is created by calling
newwin(0,0,0,0).

The delwin() routine deletes the named window, freeing all memory associated with it.
Subwindows must be deleted before the main window can be deleted.

The mvwin() routine moves the window so that the upper left-hand corner is at position
(x, y). 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.

The subwin() routine creates and returns a pointer to a new window with the given
number of lines, nlines, and columns, ncols. The window is at position (begin_y, begin_x)
on the screen. (This position is relative to the screen, and not to the window orig.) The
window is made in the middle of the window orig, so that changes made to one window
will affect both windows. The subwindow shares memory with the window orig. When
using this routine, it is necessary to call touchwin() or touchline() on orig before calling
wrefresh() on the subwindow.

The derwin() routine is the same as subwin(), except that begin_y and begin_x are relative
to the origin of the window orig rather than the screen. There is no difference
between the subwindows and the derived windows.

The mvderwin() routine moves a derived window (or subwindow) inside its parent win-
dow. 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

modified 31 Dec 1996

SunOS 5.6

3X-345
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 `wsynccdown()` updates the status structure of the window to reflect the changes in the status structures of its ancestors. Applications seldom call this routine because it is called automatically by `wrefresh()`.

**RETURN VALUES**

Routines that return an integer return the integer `ERR` upon failure and an integer value other than `ERR` upon successful completion.

`delwin()` returns the integer `ERR` upon failure and `OK` upon successful completion.

Routines that return pointers return `NULL` on error.

**ATTRIBUTES**

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

**SEE ALSO**

`curs_refresh(3X), curs_touch(3X), curses(3X), attributes(5)`

**NOTES**

The header `<curses.h>` automatically includes the headers `<stdio.h>` and `<unctrl.h>`.

If many small changes are made to the window, the `wsyncup()` option could degrade performance.

Note that `syncok()` may be a macro.
NAME  cuserid – get character login name of the user

SYNOPSIS  
```
#include <stdio.h>

char *cuserid(char *s);
```

DESCRIPTION  
cuserid( ) generates a character-string representation of the login name that the owner of
the current process is logged in under. If s is a NULL pointer, this representation is gen-
erated in an internal static area, the address of which is returned. Otherwise, s 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 s is not a NULL
pointer, a null character \0 will be placed at s[0].

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  
getlogin(3C), getpwnam(3C), attributes(5)

NOTES  
In multi-thread applications, the caller must always supply an array s for the return
value.
NAME
dbm, dbminit, dbmclose, fetch, store, delete, firstkey, nextkey – data base subroutines

SYNOPSIS
/usr/ucb/cc [flag ...] file ... -ldbmc
#include <dbm.h>

typedef struct {
   char *dptr;
   int dsize;
} datum;

int dbminit(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;

DESCRIPTION
The dbm() library has been superseded by ndbm (see dbm_clearerr(3)).
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 sys-
tem accesses.

key/dat and their content are described by the datum typedef. A datum specifies a string
of dsize bytes pointed to by dptr. 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.

Before a database can be accessed, it must be opened by dbminit(). At the time of this
call, the files file.dir and file.pag must exist. An empty database is created by creating
zero-length .dir and .pag files.

A database may be closed by calling dbmclose(). You must close a database before
opening a new one.

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:
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), dbm_clearerr(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  
dbm_clearerr, dbm_close, dbm_delete, dbm_error, dbm_fetch, dbm_firstkey,  
dbm_nextkey, dbm_open, dbm_store – database functions

SYNOPSIS  
#include <ndbm.h>
int dbm_clearerr(DBM *db);
void dbm_close(DBM *db);
int dbm_delete(DBM *db, datum key);
int dbm_error(DBM *db);
datum dbm_fetch(DBM *db, datum key);
datum dbm_firstkey(DBM *db);
datum dbm_nextkey(DBM *db);
DBM *dbm_open(const char *file, int open_flags, mode_t file_mode);
int dbm_store(DBM *db, datum key, datum content, int store_mode);

DESCRIPTION  
These functions create, access and modify a database. They maintain key/content pairs in  
a database. The functions will handle large databases (up to a billion blocks) and will  
access a keyed item in one or two file system accesses. This package replaces the earlier  
dbm(3B) library, which managed only a single database.

keys and contents are described by the datum typedef. A datum consists of at least two  
members, dptr and dsize. The dptr member points to an object that is dsize bytes in  
length. Arbitrary binary data, as well as ASCII character strings, may be stored in the  
object pointed to by dptr.

The database is stored in two files. One file is a directory containing a bit map of keys  
and has .dir as its suffix. The second file contains all data and has .pag as its suffix.

The dbm_open() function opens a database. The file argument to the function is the  
pathname of the database. The function opens two files named file.dir and file.pag. The  
open_flags argument has the same meaning as the flags argument of open(2) except that a  
database opened for write-only access opens the files for read and write access. The  
file_mode argument has the same meaning as the third argument of open(2).

The dbm_close() function closes a database. The argument db must be a pointer to a  
dbm structure that has been returned from a call to dbm_open().

The dbm_fetch() function reads a record from a database. The argument db is a pointer  
to a database structure that has been returned from a call to dbm_open(). The argument  
key is a datum that has been initialized by the application program to the value of the key  
that matches the key of the record the program is fetching.

The dbm_store() function writes a record to a database. The argument db is a pointer to  
a database structure that has been returned from a call to dbm_open(). The argument  
key is a datum that has been initialized by the application program to the value of the key  
that identifies (for subsequent reading, writing or deleting) the record the program is  
writing. The argument content is a datum that has been initialized by the application
program to the value of the record the program is writing. The argument store_mode controls whether \texttt{dbm_store()} replaces any pre-existing record that has the same key that is specified by the key argument. The application program must set store_mode to either \texttt{DBM_INSERT} or \texttt{DBM_REPLACE}. If the database contains a record that matches the key argument and store_mode is \texttt{DBM_REPLACE}, the existing record is replaced with the new record. If the database contains a record that matches the key argument and store_mode is \texttt{DBM_INSERT}, the existing record is not replaced with the new record. If the database does not contain a record that matches the key argument and store_mode is either \texttt{DBM_INSERT} or \texttt{DBM_REPLACE}, the new record is inserted in the database.

The \texttt{dbm_delete()} function deletes a record and its key from the database. The argument \texttt{db} is a pointer to a database structure that has been returned from a call to \texttt{dbm_open()}. The argument \texttt{key} is a \texttt{datum} that has been initialized by the application program to the value of the key that identifies the record the program is deleting.

The \texttt{dbm_firstkey()} function returns the first key in the database. The argument \texttt{db} is a pointer to a database structure that has been returned from a call to \texttt{dbm_open()}. The \texttt{dbm_nextkey()} function returns the next key in the database. The argument \texttt{db} is a pointer to a database structure that has been returned from a call to \texttt{dbm_open()}. The \texttt{dbm_firstkey()} function must be called before calling \texttt{dbm_nextkey()}. Subsequent calls to \texttt{dbm_nextkey()} return the next key until all of the keys in the database have been returned.

The \texttt{dbm_error()} function returns the error condition of the database. The argument \texttt{db} is a pointer to a database structure that has been returned from a call to \texttt{dbm_open()}. The \texttt{dbm_clearerr()} function clears the error condition of the database. The argument \texttt{db} is a pointer to a database structure that has been returned from a call to \texttt{dbm_open()}. These database functions support key/content pairs of at least 1024 bytes.

**RETURN VALUES**

The \texttt{dbm_store()} and \texttt{dbm_delete()} functions return 0 when they succeed and a negative value when they fail.

The \texttt{dbm_store()} function returns 1 if it is called with a flags value of \texttt{DBM_INSERT} and the function finds an existing record with the same key.

The \texttt{dbm_error()} function returns 0 if the error condition is not set and returns a non-zero value if the error condition is set.

The return value of \texttt{dbm_clearerr()} is unspecified.

The \texttt{dbm_firstkey()} and \texttt{dbm_nextkey()} functions return a key \texttt{datum}. When the end of the database is reached, the \texttt{dptr} member of the key is a null pointer. If an error is detected, the \texttt{dptr} member of the key is a null pointer and the error condition of the database is set.

The \texttt{dbm_fetch()} function returns a content \texttt{datum}. If no record in the database matches the key or if an error condition has been detected in the database, the \texttt{dptr} member of the content is a null pointer.
The `dbm_open()` function returns a pointer to a database structure. If an error is detected during the operation, `dbm_open()` returns a `(DBM *)0`.

**ERRORS**
No errors are defined.

**USAGE**
The following code can be used to traverse the database:

```c
for(key = dbm_firstkey(db); key.dptr != NULL; key = dbm_nextkey(db))
```

The `dbm_` functions provided in this library should not be confused in any way with those of a general-purpose database management system. These functions do not provide for multiple search keys per entry, they do not protect against multi-user access (in other words they do not lock records or files), and they do not provide the many other useful database functions that are found in more robust database management systems. Creating and updating databases by use of these functions is relatively slow because of data copies that occur upon hash collisions. These functions are useful for applications requiring fast lookup of relatively static information that is to be indexed by a single key.

The `dptr` pointers returned by these functions may point into static storage that may be changed by subsequent calls.

The `dbm_delete()` function does not physically reclaim file space, although it does make it available for reuse.

After calling `dbm_store()` or `dbm_delete()` during a pass through the keys by `dbm_firstkey()` and `dbm_nextkey()`, the application should reset the database by calling `dbm_firstkey()` before again calling `dbm_nextkey()`.

**EXAMPLES**
The following example stores and retrieves a phone number, using the name as the key. Note that this example does not include error checking.

```c
#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);
}
```

3-352 SunOS 5.6 modified 29 Dec 1996
(void) printf("Name: %s, Phone Number: %s\n", name.dptr, get_phone_no.dptr);

/* Close the database */
dbm_close(db);
return (0);

ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO ar(1), cat(1), cp(1), tar(1), open(2), dbm(3B), netconfig(4), attributes(5)

NOTES

The .pag file will contain holes so that its apparent size may be larger than its actual content. Older versions of the UNIX operating system may create real file blocks for these holes when touched. These files cannot be copied by normal means (cp(1), cat(1), tar(1), ar(1)) without filling in the holes.

The sum of the sizes of a key/content pair must not exceed the internal block size (currently 1024 bytes). Moreover all key/content pairs that hash together must fit on a single block. dbm_store() will return an error in the event that a disk block fills with inseparable data.

The order of keys presented by dbm_firstkey() and dbm_nextkey() depends on a hashing function.

There are no interlocks and no reliable cache flushing; thus concurrent updating and reading is risky.

The database files (file.dir and file.pag) are binary and are architecture-specific (for example, they depend on the architecture’s byte order.) These files are not guaranteed to be portable across architectures.
NAME
decimal_to_floating, decimal_to_single, decimal_to_double, decimal_to_extended,
decimal_to_quadruple – convert decimal record to floating-point value

SYNOPSIS
#include <floatingpoint.h>

void decimal_to_single(single *px, decimal_mode *pm,
fp_exception_field_type *ps,
decimal_record *pd);

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);

DESCRIPTION
The decimal_to_floating() functions convert the decimal record at *pd into a floating-point value at *px, observing the modes specified in *pm and setting exceptions in *ps. If there are no IEEE exceptions, *ps will be zero.

*px is correctly rounded according to the IEEE rounding modes in pm->rd. *ps is set to contain fp_inexact, fp_underflow, or fp_overflow if any of these arise.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
fscanf(3S), scanf(3S), sscanf(3S), and strtod(3C) all use decimal_to_double().

3-354 SunOS 5.6 modified 29 Dec 1996
### NAME
def_prog_mode, def_shell_mode, reset_prog_mode, reset_shell_mode – save/restore terminal modes

### SYNOPSIS
```c
#include <curses.h>

int def_prog_mode(void);
int def_shell_mode(void);
int reset_prog_mode(void);
int reset_shell_mode(void);
```

### DESCRIPTION
The `def_prog_mode()` and `def_shell_mode()` functions save the current terminal modes as "program" (within X/Open Curses) or "shell" (outside X/Open Curses). The modes are saved automatically by `initscr(3XC)`, `newterm(3XC)`, and `setupterm(3XC)`.

The `reset_prog_mode()` and `reset_shell_mode()` functions reset the current terminal modes to "program" (within X/Open Curses) or "shell" (outside X/Open Curses). The `endwin(3XC)` function automatically calls the `reset_shell_mode()` function and the `doupdate(3XC)` function calls the `reset_prog_mode()` function after calling `endwin()`.

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

### ERRORS
None.

### SEE ALSO
`endwin(3XC)`, `initscr(3XC)`, `newterm(3XC)`, `setupterm(3XC)`

---

modified 1 Jun 1996

SunOS 5.6

3XC-355
### NAME
delay_output – delays output

### SYNOPSIS
```c
#include <curses.h>

int delay_output(int ms);
```

### ARGUMENTS
- `ms` Is the number of milliseconds to delay the output.

### DESCRIPTION
The `delay_output()` function delays output for `ms` milliseconds by inserting pad characters in the output stream.

### RETURN VALUES
- On success, the `delay_output()` function returns `OK`. Otherwise, it returns `ERR`.

### ERRORS
None.

### SEE ALSO
- `napms(3XC)`

---

3XC-356

SunOS 5.6

modified 1 Jun 1996
NAME
delch, mvdelch, mvwdelch, wdelch – remove a character

SYNOPSIS
#include <curses.h>
int delch(void);
int mvdelch(int y, int x);
int mvwdelch(WINDOW *win, int y, int x);
int wdelch(WINDOW *win);

ARGUMENTS
y    Is the y (row) coordinate of the position of the character to be removed.
x    Is the x (column) coordinate of the position of the character to be removed.
win  Is a pointer to the window containing the character to be removed.

DESCRIPTION
The delch() and wdelch() functions delete the character at the current cursor position
from stdscr and win, respectively. All remaining characters after cursor through to the
end of the line are shifted one character towards the start of the line. The last character
on the line becomes a space; characters on other lines are not affected.
The mvdelch() and mvwdelch() functions delete the character at the position specified
by the x and y parameters; the former deletes the character from stdscr; the latter from
win.

RETURN VALUES
On success, these functions return OK. Otherwise, they return ERR.

ERRORS
None.

SEE ALSO
bkgdset(3XC), insch(3XC)
NAME  del_curterm, restartterm, set_curterm, setterm, setupterm – free space pointed to by terminal

SYNOPSIS  #include <term.h>

int del_curterm(TERMINAL *oterm);
int restartterm(char *term, int fildes, int *errret);
TERMINAL *set_curterm (TERMINAL *nterm);
int setterm (char *term);
int setupterm(char *term, int fildes, int *errret);

ARGUMENTS  oterm  Is the terminal type for which to free space.
       term  Is the terminal type for which variables are set.
      fildes  Is a file descriptor initialized for output.
       errret  Is a pointer to an integer in which the status value is stored.
       nterm  Is the new terminal to become the current terminal.

DESCRIPTION  Within X/Open Curses, the setupterm() function is automatically called by the initscr (3XC) and newterm (3XC) functions. This function can be also be used outside of X/Open Curses when a program has to deal directly with the terminfo database to handle certain terminal capabilities. The use of appropriate X/Open Curses functions is recommended in all other situations.

The setupterm() function loads terminal-dependent variables for the terminfo layer of X/Open Curses. The setupterm() function initializes the terminfo variables lines and columns such that if use_env(FALSE) has been called, the terminfo values assigned in the database are used regardless of the environmental variables LINES and COLUMNS or the program’s window dimensions; when use_env(TRUE) has been called, which is the default, the environment variables LINES and COLUMNS are used, if they exist. If the environment variables do not exist and the program is running in a window, the current window size is used.

The term parameter of setupterm() specifies the terminal; if null, terminal type is taken from the TERM environment variable. All output is sent to fildes which is initialized for output. If errret is not null, OK or ERR is returned and a status value is stored in the integer pointed to by errret. The following status values may be returned:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Normal</td>
</tr>
<tr>
<td>0</td>
<td>Terminal could not be found</td>
</tr>
<tr>
<td>-1</td>
<td>terminfo database could not be found</td>
</tr>
</tbody>
</table>

If errret is null, an error message is printed, and the setupterm() function calls the exit() function with a non-zero parameter.
The `setterm()` macro is an older version of `setupterm()`. It is included for compatibility with previous versions of Curses. New programs should use `setupterm()`.

The `set_curterm()` function sets the `cur_term` variable to `nterm`. The values from `nterm` as well as other state information for the terminal are used by X/Open Curses functions such as `beep(3XC)`, `flash(3XC)`, `mvcur(3XC)`, `tigetflag(3XC)`, `tigetstr(3XC)`, and `tigetnum(3XC)`.

The `del_curterm()` function frees the space pointed to by `oterm`. If `oterm` and the `cur_term` variable are the same, all Boolean, numeric, or string `terminfo` variables will refer to invalid memory locations until you call `setupterm()` and specify a new terminal type.

The `restartterm()` function assumes that a call to `setupterm()` has already been made (probably from `initscr()` or `newterm()`). It allows you to specify a new terminal type in `term` and updates the data returned by `baudrate(3XC)` based on `fd`es. Other information created by the `initscr()`, `newterm()`, and `setupterm()` functions is preserved.

**RETURN VALUES**

On success, the `set_curterm()` function returns the previous value of `cur_term`. Otherwise, it returns a null pointer.

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

**ERRORS**

None.

**SEE ALSO**

`baudrate(3XC)`, `beep(3XC)`, `initscr(3XC)`, `mvcur(3XC)`, `tigetflag(3XC)`, `use_env(3XC)`
NAME     deleteln, wdeleteln – remove a line

SYNOPSIS  #include <curses.h>
           int deleteln(void);
           int wdeleteln (WINDOW *win);

ARGUMENTS  win     Is a pointer to the window from which the line is removed.

DESCRIPTION The deleteln() and wdeleteln() functions delete the line containing the cursor from
              stdscr and win, respectively. All lines below the one deleted are moved up one line. The
              last line of the window becomes blank. The position of the cursor is unchanged.

RETURN VALUES On success, these functions return OK. Otherwise, they return ERR.

ERRORS     None.

SEE ALSO   bkgdset(3XC), insdelln(3XC), insertln(3XC)
### NAME
delscreen – free space associated with the SCREEN data structure

### SYNOPSIS
```c
#include <curses.h>

void delscreen(SCREEN *sp);
```

### ARGUMENTS
- `sp` Is a pointer to the screen structure for which to free space.

### DESCRIPTION
The `delscreen()` function frees space associated with the `SCREEN` data structure. This function should be called after `endwin(3XC)` if a `SCREEN` data structure is no longer needed.

### RETURN VALUES
The `delscreen()` function does not return a value.

### ERRORS
None.

### SEE ALSO
`endwin(3XC)`, `initscr(3XC)`, `newterm(3XC)`
### NAME
```
delwin – delete a window
```

### SYNOPSIS
```
#include <curses.h>

int delwin(WINDOW *win);
```

### ARGUMENTS
```
win Is a pointer to the window that is to be deleted.
```

### DESCRIPTION
The `delwin()` function deletes the specified window, freeing up the memory associated with it.

Deleting a parent window without deleting its subwindows and then trying to manipulate the subwindows will have undefined results.

### RETURN VALUES
On success, this functions returns `OK`. Otherwise, it returns `ERR`.

### ERRORS
None.

### SEE ALSO
`derwin(3XC), dupwin(3XC)`
NAME  
demangle, cplus_demangle – decode a C++ encoded symbol name

SYNOPSIS  
cc [ flag ... ] file [ library... ] −ldemangle
#include <demangle.h>
int cplus_demangle(const char ∗symbol, char ∗prototype, size_t size);

DESCRIPTION  
The cplus_demangle() function decodes (demangles) a C++ linker symbol name (mangled name) into a (partial) C++ prototype, if possible. C++ mangled names may not have enough information to form a complete prototype.

The symbol string argument points to the input mangled name.

The prototype argument points to a user-specified output string buffer, of size bytes.

The cplus_demangle() function operates on mangled names generated by SPARCCompilers C++ 3.0.1, 4.0.1, 4.1 and 4.2.

The cplus_demangle() function improves and replaces the demangle() function.

Refer to the CC.1, dem.1, and c++filt.1 manual pages in the /opt/SUNWspro/man/man1 directory. These pages are only available with the SPROcc package.

RETURN VALUES  
The cplus_demangle() function returns the following values:

0  The symbol argument is a valid mangled name and prototype contains a (partial) prototype for the symbol.

DEMANGLE_ENAME  The symbol argument is not a valid mangled name and the content of prototype is a copy of the symbol.

DEMANGLE_ESPACE  The prototype output buffer is too small to contain the prototype (or the symbol), and the content of prototype is undefined.
NAME
derwin, newwin, subwin – create a new window or subwindow

SYNOPSIS
#include <curses.h>
WINDOW *derwin(WINDOW *orig, int nlines, int ncols,
   int begin_y, int begin_x);
WINDOW *newwin(int nlines, int ncols, int begin_y,
   int begin_x);
WINDOW *subwin(WINDOW *orig, int nlines, int ncols,
   int begin_y, int begin_x);

ARGUMENTS
orig Is a pointer to the parent window for the newly created subwindow.
nlines Is the number of lines in the subwindow.
cols Is the number of columns in the subwindow.
begin_y Is the y (row) coordinate of the upper left corner of the subwindow, relative to
   the parent window.
begin_x Is the x (column) coordinate of the upper left corner of the subwindow, relative to
   the parent window.

DESCRIPTION
The derwin() function creates a subwindow within window orig, with the specified
number of lines and columns, and upper left corner positioned at begin_x, begin_y relative
to window orig. A pointer to the new window structure is returned.
The newwin() function creates a new window with the specified number of lines and
columns and upper left corner positioned at begin_x, begin_y. A pointer to the new win-
dow structure is returned. A full-screen window can be created by calling
newwin(0,0,0,0).
If the number of lines specified is zero, newwin() uses a default value of LINES minus
begin_y; if the number of columns specified is zero, newwin() uses the default value of
COLS minus begin_x.
The subwin() function creates a subwindow within window orig, with the specified
number of lines and columns, and upper left corner positioned at begin_x, begin_y (rela-
tive to the physical screen, not to window orig). A pointer to the new window structure is
returned.
The original window and subwindow share character storage of the overlapping area
(each window maintains its own pointers, cursor location, and other items). This means
that characters and attributes are identical in overlapping areas regardless of which win-
dow characters are written to.
When using subwindows, it is often necessary to call touchwin(3XC) before
wrefresh(3XC) to maintain proper screen contents.
RETURN VALUES
On success, these functions return a pointer to the newly-created window. Otherwise, they return ERR.

ERRORS
None.

SEE ALSO
doupdate(3XC), is_linetouched(3XC)
NAME
devid_get, devid_free, devid_get_minor_name, devid_deviceid_to_nmlist,
devid_free_nmlist, devid_compare, devid_sizeof – device id interfaces for user applications

SYNOPSIS
#include <devid.h>

int devid_get(int fd ddi_devid_t *retdevid);
void devid_free(ddi_devid_t devid);
int devid_get_minor_name(int fd, char **retminor_name);
int devid_deviceid_to_nmlist(char *search_path, ddi_devid_t devid,
   char *minor_name, devid_nmlist_t **retlist);
void devid_free_nmlist(devid_nmlist_t *list);
int devid_compare(ddi_devid_t devid1, ddi_devid_t devid2);
size_t devid_sizeof(ddi_devid_t devid);

DESCRIPTION
The following routines are used to provide unique identifiers, device ids, for devices. Specifically, applications and device drivers use these interfaces to identify and locate devices, independent of the device’s physical connection or its logical device name or number.

devid_get() returns the device id, in retdevid, for the device associated with the open file descriptor fd, which refers to any device. If the device does not have a device id associated with it then an error is returned. The caller of this function must free the memory allocated for the retdevid returned, using the devid_free() function.

devid_free() frees the allocated space for the passed-in devid, allocated by devid_get().

devid_get_minor_name() returns the minor name, in retminor_name, for the device associated with the open file descriptor fd. This name is specific to the particular minor number, but is “instance number” specific. The caller of this function must free the memory allocated for the returned string in retminor_name, using the devid_free() function.

devid_deviceid_to_nmlist() returns an array of devid_nmlist structures, where each entry matches the devid id and minor name passed in. The devid_nmlist structure contains the device name and device number. The last entry of the array has a null pointer for the devname and NODEV for the device number.

This function walks through the file tree, starting at search_path. For each device with a matching device id and minor name tuple, a device name and device number are added to the retlist. If no matches are found, an error is returned. The caller of this function must free the memory allocated for the returned array with the devid_free_nmlist() function.

devid_free_nmlist() frees the memory allocated by the devid_deviceid_to_nmlist() function.
devid_compare() compares two device ids byte-by-byte and determines both equality and sort order. The function returns an integer greater than zero if the device id pointed to by devid1 is greater than the device id pointed to by devid2. It returns zero if the device id pointed to by devid1 is equal to the device id pointed to by devid2. It returns an integer less than zero if the device id pointed to by devid1 is less than the device id pointed to by devid2.

devid_sizeof() returns the size in number of bytes allocated for the devid.

RETURN VALUES

The following functions return 0 upon successful completion: devid_get(), devid_get_minor_name(), and devid_deviceid_to_nmlist(). Otherwise, −1 is returned and errno is set to indicate the error.

The function devid_compare() returns the following values:

−1 The device id pointed to by devid1 is less than the device id pointed to by devid2.
0 The device id pointed to by devid1 is equal to the device id pointed to by devid2.
≥1 The device id pointed to by devid1 is greater than the device id pointed to by devid2.

The return value from devid_sizeof() is the size in number of bytes allocated for the devid.

EXAMPLES

The following example shows the proper use of devid_get() and devid_get_minor_name() to free the space allocated for the device id and minor name.

```c
int fd;
ddi_devid_t devid;
char *minor_name;

if ((fd = open("/dev/dsk/c0t3d0s0", O_RDONLY | O_NDELAY)) < 0) {
    ...
}

if (devid_get(fd, &devid) != 0) {
    ...
}

if (devid_get_minor_name(fd, &minor_name) != 0) {
    ...
}

< process devid and minor_name >

devid_free(devid);
free(minor_name);
```
The following example shows the proper use of `devid_deviceid_to_nmlist()` and `devid_free_nmlist()`:

```c
devid_nmlist_t *list = NULL;
int err;

err = devid_deviceid_to_nmlist("/dev/rdsk", devid, minor_name, &list);
if (err)
    return (err);

< loop through list and process device names and device numbers >

devid_free_nmlist(list);
```

**FILES**

- `/usr/lib/libdevid.so.1` The location of the device id library interfaces.
- `/usr/lib/libdevid.so` A symlink to `/usr/lib/libdevid.so.1`.

**ATTRIBUTES**

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT Level</td>
<td>MT–Safe</td>
</tr>
</tbody>
</table>

**SEE ALSO**

- libdevid(4), attributes(5), ddi_devid_devlist(9F), ddi_devid_free(9F), ddi_devid_init(9F), ddi_devid_register(9F), ddi_devid_sizeof(9F), ddi_devid_unregister(9F), ddi_devid_valid(9F)
NAME
dial – establish an outgoing terminal line connection

SYNOPSIS
cc [ flag ...] file ... -lnsl [ library ...]
#include <dial.h>
int dial(CALL call);
void undial(int fd);

DESCRIPTION
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>).

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.

CALL is defined in the header <dial.h> and has the following members:

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 */

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.

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.

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:

0-9 dial 0-9
* dial *
# dial 
= wait for secondary dial tone
- delay for approximately 4 seconds
The CALL element **modem** is used to specify modem control for direct lines. This element should be non-zero if modem control is required. The CALL element **attr** is a pointer to a **termio** structure, as defined in the header `<termio.h>`. A NULL value for this pointer element may be passed to the **dial** function, but if such a structure is included, the elements specified in it will be set for the outgoing terminal line before the connection is established. This setting is often important for certain attributes such as parity and baud-rate.

The CALL elements **device** and **dev_len** are no longer used. They are retained in the CALL structure for compatibility reasons.

**RETURN VALUES**

On failure, a negative value indicating the reason for the failure will be returned. Mnemonics for these negative indices as listed here are defined in the header `<dial.h>`.

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTRPT</td>
<td>-1</td>
<td>interrupt occurred</td>
</tr>
<tr>
<td>D_HUNG</td>
<td>-2</td>
<td>dialer hung (no return from write)</td>
</tr>
<tr>
<td>NO_ANS</td>
<td>-3</td>
<td>no answer within 10 seconds</td>
</tr>
<tr>
<td>ILL_BD</td>
<td>-4</td>
<td>illegal baud-rate</td>
</tr>
<tr>
<td>A_PROB</td>
<td>-5</td>
<td>acu problem (open() failure)</td>
</tr>
<tr>
<td>L_PROB</td>
<td>-6</td>
<td>line problem (open() failure)</td>
</tr>
<tr>
<td>NO_Ldv</td>
<td>-7</td>
<td>can’t open Devices file</td>
</tr>
<tr>
<td>DV_NT_A</td>
<td>-8</td>
<td>requested device not available</td>
</tr>
<tr>
<td>DV_NT_K</td>
<td>-9</td>
<td>requested device not known</td>
</tr>
<tr>
<td>NO_BD_A</td>
<td>-10</td>
<td>no device available at requested baud</td>
</tr>
<tr>
<td>NO_BD_K</td>
<td>-11</td>
<td>no device known at requested baud</td>
</tr>
<tr>
<td>DV_NT_E</td>
<td>-12</td>
<td>requested speed does not match</td>
</tr>
<tr>
<td>BAD_SYS</td>
<td>-13</td>
<td>system not in Systems file</td>
</tr>
</tbody>
</table>

**FILES**

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

**ATTRIBUTES**

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

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

**SEE ALSO**

`uucp(1C), alarm(2), read(2), write(2), attributes(5), termio(7I)`

**NOTES**

Including the header `<dial.h>` automatically includes the header `<termio.h>`.

An alarm(2) system call for 3600 seconds is made (and caught) within the dial module for the purpose of "touching" the LCK.. file and constitutes the device allocation semaphore for the terminal device. Otherwise, uucp(1C) may simply delete the LCK.. entry on its 90-minute clean-up rounds. The alarm may go off while the user program is in a read(2) or write(2) function, causing an apparent error return. If the user program expects to be around for an hour or more, error returns from read()s should be checked for (errno==EINTR), and the read() possibly reissued.
This interface is unsafe in multithreaded applications. Unsafe interfaces should be called only from the main thread.
NAME  
difftime – computes the difference between two calendar times

SYNOPSIS  
#include <time.h>

double difftime(time_t time1, time_t time0);

DESCRIPTION  
difftime() computes the difference between two calendar times. difftime() 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.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  
cftime(3C), attributes(5)
NAME  
directio – provide advice to file system

SYNOPSIS  
#include <sys/types.h>
#include <sys/fcntl.h>

int directio(int fildes, int advice)

DESCRIPTION  
directio() provides advice to the system about the expected behavior of the application when accessing the data in the file associated with the open file descriptor, fildes. The system uses this information to help optimize accesses to the file's data. directio() has no effect on the semantics of the other operations on the data, though it may affect the performance of other operations.

advice is kept per file; so the last caller of directio() sets the advice for all applications using the file associated with fildes.

Values for advice are defined in <sys/fcntl.h>.

DIRECTIO_OFF

Applications get the default system behavior when accessing file data.

When an application reads data from a file, the data is first cached in system memory and then copied into the application’s buffer (see read(2)). If the system detects that the application is reading sequentially from a file, the system will asynchronously “read ahead” from the file into system memory so the data is immediately available for the next read(2) operation.

When an application writes data into a file, the data is first cached in system memory and is written to the device at a later time (see write(2)). When possible, the system increases the performance of write(2) operations by caching the data in memory pages. The data is copied into system memory and the write(2) operation returns immediately to the application. The data is later written asynchronously to the device. When possible, the cached data is “clustered” into large chunks and written to the device in a single write operation.

The system behavior for DIRECTIO_OFF can change without notice.

DIRECTIO_ON

The system behaves as though the application is not going to reuse the file data in the near future. In other words, the file data is not cached in the system’s memory pages.

When possible, data is read or written directly between the application’s memory and the device when the data is accessed with read(2) and write(2) operations. When such transfers are not possible, the system switches back to the default behavior, but just for that operation. In general, the transfer is possible when the application’s buffer is aligned on a two-byte (short) boundary, the offset into the file is on a device sector boundary, and the size of the operation is a multiple of device sectors.

This advisory is ignored while the file associated with fildes is mapped (see

modified 29 Dec 1996

SunOS 5.6

3C-373
mmap(2)).
The system behavior for DIRECTIO_ON can change without notice.

**RETURN VALUES**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful completion.</td>
</tr>
<tr>
<td>-1</td>
<td>An error occurred and directio() sets errno to indicate the error.</td>
</tr>
</tbody>
</table>

**ERRORS**

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EBADF</td>
<td>fildes is not a valid open file descriptor.</td>
</tr>
<tr>
<td>ENOTTY</td>
<td>fildes is not associated with a file system that accepts advisory functions.</td>
</tr>
<tr>
<td>EINVAL</td>
<td>The value in advice is invalid.</td>
</tr>
</tbody>
</table>

**ATTRIBUTES**

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

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**SEE ALSO**

fstyp(1M), mmap(2), open(2), read(2), write(2), attributes(5), fcntl(5)

**WARNINGS**

Switching between DIRECTIO_OFF and DIRECTIO_ON can slow the system because each
switch to DIRECTIO_ON may entail flushing the file’s data from the system’s memory.
Small sequential I/O generally performs best with DIRECTIO_OFF.
Large sequential I/O generally performs best with DIRECTIO_ON; except when a file is
sparse or is being extended, and the file is opened with O_SYNC or O_DSYNC (see
open(2)).

**NOTES**
directio() is supported for the ufs file system type (see fstyp(1M)).
NAME
dirname – report the parent directory name of a file path name

SYNOPSIS
#include <libgen.h>
char *dirname(char *path);

DESCRIPTION
The dirname() function takes a pointer to a character string that contains a pathname,
and returns a pointer to a string that is a pathname of the parent directory of that file.
Trailing ‘/’ characters in the path are not counted as part of the path.
If path does not contain a ‘/’, then dirname() returns a pointer to the string “.”. If path is
a null pointer or points to an empty string, dirname() returns a pointer to the string “.”.

RETURN VALUES
The dirname() function returns a pointer to a string that is the parent directory of path. If
path is a null pointer or points to an empty string, a pointer to a string “.” is returned.

EXAMPLES

<table>
<thead>
<tr>
<th>Input String</th>
<th>Output String</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>

The following code fragment reads a path name, changes directory to the parent directory
of the named file (see chdir(2)), and opens the file.

char path[100], *pathcopy;
int fd;
gets (path);
pathcopy = strdup (path);
chdir (dirname (pathcopy) );
free (pathcopy);
fd = open (basename (path),O_RDONLY);

USAGE
The dirname() function may modify the string pointed to by path, and may return a
pointer to static storage that may then be overwritten by subsequent calls to dirname().
The dirname() and basename(3C) functions together yield a complete pathname. The
expression dirname(path) obtains the pathname of the directory where basename(path) is
found.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

modified 29 Dec 1996 SunOS 5.6 3C-375
SEE ALSO basename(1), chdir(2), basename(3C), attributes(5)

NOTES When compiling multi-thread applications, the _REENTRANT flag must be defined on the compile line. This flag should only be used in multi-thread applications.
NAME  div, ldiv, lldiv – compute the quotient and remainder

SYNOPSIS  

```c
#include <stdlib.h>

div_t div(int numer, int denom);
ldiv_t ldiv(long int numer, long int denom);
lldiv_t lldiv(long long numer, long long denom);
```

DESCRIPTION  

`div()` computes the quotient and remainder of the division of the numerator `numer` by the denominator `denom`. 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, `quotient * denom + remainder` will equal `numer`.

`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`.

RETURN VALUES  

`div()` returns a structure of type `div_t`, comprising both the quotient and remainder:

```c
int quot; /*quotient*/
int rem; /*remainder*/
```

`ldiv()` returns a structure of type `ldiv_t` and `lldiv()` returns a structure of type `lldiv_t`, comprising both the quotient and remainder:

```c
long int quot; /*quotient*/
long int rem; /*remainder*/
```

ATTRIBUTES  

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  

`attributes(5)`
NAME
dladdr – translate address to symbolic information

SYNOPSIS
cc [ flag ... ] file ... -ldl [ library ... ]
#include <dlfcn.h>
int dladdr(void *address, Dl_info *dli);

DESCRIPTION
dladdr() is one of a family of routines that give the user direct access to the dynamic linking facilities. (See Linker and Libraries Guide). These routines are made available via the library loaded when the option -ldl is passed to the link-editor.

Note: These routines are available to dynamically-linked processes ONLY.

dladdr() determines if the specified address 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 _end 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.

The Dl_info structure must be preallocated by the user. The structure members are filled in by dladdr() based on the specified address. The Dl_info structure includes the following members:

- const char * dli_fname;
- void * dli_fbase;
- const char * dli_sname;
- void * dli_saddr;

Descriptions of these members appear below.

dli_fname Contains a pointer to the filename of the containing object.
dli_fbase Contains the base address of the containing object.
dli_sname Contains a pointer to the symbol name nearest to the specified address. This symbol either has the same address, or is the nearest symbol with a lower address.
dli_saddr Contains the actual address of the above symbol.

RETURN VALUES
If the specified address 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.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

3X-378 SunOS 5.6 modified 31 Dec 1996
SEE ALSO  |  ld(1), dlclose(3X), dldump(3X), dlerror(3X), dlopen(3X), dlsym(3X), attributes(5)  

Linker and Libraries Guide

NOTES  |  The DL_info pointer elements point to addresses within the mapped objects. These may become invalid if objects are removed prior to these elements being used (see dlclose()). If no symbol is found to describe the specified address, both the dli_sname and dli_saddr members are set to 0.
NAME
dlclose – close a shared object

SYNOPSIS
cc [ flag ... ] file ... -ldl [ library ... ]
#include <dlfcn.h>
int dlclose(void *handle);

DESCRIPTION
dlclose() is one of a family of routines that give the user direct access to the dynamic
linking facilities. (See Linker and Libraries Guide). These routines are made available via
the library loaded when the option -ldl is passed to the link-editor.

Note: These routines are available to dynamically-linked processes ONLY.
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 avail-
able to dlsym(). All objects loaded automatically as a result of invoking dlopen() on the
referenced object are also closed. handle is the value returned by a previous invocation of
dlopen().

RETURN VALUES
If the referenced object was successfully closed, dlclose() returns 0. If the object could
not be closed, or if handle does not refer to an open object, dlclose() returns a non-zero
value. More detailed diagnostic information will be available through dlerror().

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
Id(1), dladdr(3X), dldump(3X), dlerror(3X), dlopen(3X), dlsym(3X), attributes(5)
Linker and Libraries Guide

NOTES
A successful invocation of dlclose() does not guarantee that the objects associated with
handle 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.

Once an object has been closed by dlclose(), referencing symbols contained in that object
can cause undefined behavior.
NAME
dldump – create a new file from a dynamic object component of the calling process

SYNOPSIS
cc [ flag ...] file ... -ldl [ library ...]
#include <dlfcn.h>
int dldump(const char * ipath, const char * opath, int flags);

DESCRIPTION
dldump( ) is one of a family of routines that give the user direct access to the dynamic
linking facilities. (See Linker and Libraries Guide). These routines are made available via
the library loaded when the option -ldl is passed to the link-editor.

Note: These routines are available to dynamically-linked processes ONLY.
dldump( ) creates a new dynamic object opath from an existing dynamic object ipath that
is bound to the current process. An ipath value of 0 is interpreted as the dynamic object
that started the process. The new object is constructed from the existing objects’ disc file.
Relocations can be applied to the new object to pre-bind it to other dynamic objects, or fix
the object to a specific memory location. In addition, data elements within the new object
may be obtained from the objects’ memory image as it exists in the calling process.

These techniques allow the new object to be executed with a lower startup cost, either
because there are less relocations required to load the object, or because of a reduction in
the data processing requirements of the object. However, it is important to note that limi-
tations may exist in using these techniques. Applying relocations to the new dynamic
object opath may restrict its flexibility within a dynamically changing environment. In
addition, limitations regarding data usage may make dumping a memory image imprac-
tical (see EXAMPLES).

The runtime linker verifies that the dynamic object ipath is mapped as part of the current
process. Thus, the object must either be the dynamic object that started the process (see
eexec(2)), one of the process’s dependencies, or an object that has been preloaded (see
ld.so.1(1)).

As part of the runtime processing of a dynamic object, relocation records within the object
are interpreted and applied to offsets within the object. These offsets are said to be relo-
cated. Relocations can be categorized into two basic types: non-symbolic and symbolic.
The non-symbolic relocation is a simple relative relocation that requires the base address at
which the object is mapped to perform the relocation. The symbolic relocation requires
the address of an associated symbol, and results in a binding to the dynamic object that
defines this symbol. This symbol definition may originate from any of the dynamic
objects that make up the process, that is, the object that started the process, one of the
process’s dependencies, an object that has been preloaded, or the dynamic object being
relocated.

The flags parameter controls the relocation processing and other attributes of producing
the new dynamic object opath. Without any flags, the new object is constructed solely
from the contents of the ipath disc file without any relocations applied.
Various relocation flags may be or'ed into the flags parameter to affect the relocations applied to the new object. Non-symbolic relocations can be applied using the following:

**RTLD_REL_RELATIVE**
- Relocation records from the object `ipath`, that define relative relocations, are applied to the object `opath`.

A variety of symbolic relocations can be applied using the following flags (each of these flags also implies `RTLD_REL_RELATIVE` is in effect):

**RTLD_REL_EXEC**
- Symbolic relocations that result in binding `ipath` to the dynamic object that started the process (commonly a dynamic executable) are applied to the object `opath`.

**RTLD_REL_DEPENDS**
- Symbolic relocations that result in binding `ipath` to any of the dynamic dependencies of the process are applied to the object `opath`.

**RTLD_REL_PRELOAD**
- Symbolic relocations that result in binding `ipath` to any objects preloaded with the process are applied to the object `opath`. (See `LD_PRELOAD` in `ld.so.1`).

**RTLD_REL_SELF**
- Symbolic relocations that result in binding `ipath` to itself are applied to the object `opath`.

**RTLD_REL_ALL**
- All relocation records defined in the object `ipath` are applied to the new object `opath` (this is basically a concatenation of all the above relocation flags).

Note that for dynamic executables, `RTLD_REL_RELATIVE`, `RTLD_REL_EXEC`, and `RTLD_REL_SELF` have no effect (see EXAMPLES).

If relocations, knowledgeable of the base address of the mapped object, are applied to the new object `opath`, then the new object will become fixed to the location that the `ipath` image is mapped within the current process.

Any relocations applied to the new object `opath` will have the original relocation record removed so that the relocation will not be applied more than once. Otherwise, the new object `opath` will retain the relocation records as they exist in the `ipath` disc file.

The following additional attributes for creating the new dynamic object `opath` can be specified using the flags parameter:

**RTLD_MEMORY**
- The new object `opath` is constructed from the current memory contents of
the ipath image as it exists in the calling process. This option allows data modified by the calling process to be captured in the new object. Note that not all data modifications may be applicable for capture; significant restrictions exist in using this technique (see EXAMPLES).

By default, when processing a dynamic executable, any allocated memory that follows the end of the data segment is captured in the new object (see malloc(3C) and brk(2)). This data, which represents the process heap, is saved as a new .SUNW_heap section in the object opath. The objects’ program headers and symbol entries, such as _end, are adjusted accordingly. See also RTLD_NOHEAP.

When using this attribute, any relocations that have been applied to the ipath memory image that do not fall into one of the requested relocation categories are undone, that is, the relocated element is returned to the value as it existed in the ipath disc file.

RTLD_STRIP

Only collect allocatable sections within the object opath; sections that are not part of the dynamic objects’ memory image are removed. This parameter reduces the size of the opath disc file and is comparable to having run the new object through strip(1).

RTLD_NOHEAP

Do not save any heap to the new object. This option is only meaningful when processing a dynamic executable with the RTLD_MEMORY attribute and allows for reducing the size of the opath disc file. In this case, the executable must confine its data initialization to data elements within its data segment and must not use any allocated data elements that comprise the heap.

It should be emphasized that an object created by dldump() is simply an updated ELF object file. No additional state regarding the process at the time dldump() is called is maintained in the new object. dldump() does not provide a panacea for checkpoint/resume. A new dynamic executable, for example, will not start where the original executable called dldump(); it will gain control at the executable’s normal entry point (see EXAMPLES).

RETURN VALUES

On successful creation of the new object, dldump() returns 0. Otherwise, a non-zero value is returned and more detailed diagnostic information is available through dlerror().

EXAMPLES

The following code fragment, which can be part of a dynamic executable a.out, can be used to create a new shared object from one of the dynamic executables’ dependencies libfoo.so.1:
const char * ipath = "libfoo.so.1";
const char * opath = "/tmp/libfoo.so.1";

.....

if (dldump(ipath, opath, RTLD_REL_RELATIVE) != 0)
    (void) printf("dldump failed: %s\n", dlerror());

The new shared object opath is fixed to the address of the mapped ipath bound to the
dynamic executable a.out. All relative relocations are applied to this new shared object,
which will reduce its relocation overhead when it is used as part of another process.

By performing only relative relocations, any symbolic relocation records remain defined
within the new object, and thus the dynamic binding to external symbols will be
preserved when the new object is used.

Use of the other relocation flags can fix specific relocations in the new object and thus can
reduce even more the runtime relocation startup cost of the new object. However, this
will also restrict the flexibility of using the new object within a dynamically changing
environment, as it will bind the new object to some or all of the dynamic objects presently
mapped as part of the process.

For example, the use of RTLD_REL_SELF will cause any references to symbols from ipath
to be bound to definitions within itself if no other preceding object defined the same sym-
bol. In other words, a call to foo( ) within ipath will bind to the definition foo within the
same object. Therefore, opath will have one less binding that must be computed at run-
time. This reduces the startup cost of using opath by other applications; however, inter-
position of the symbol foo will no longer be possible.

Using a dumped shared object with applied relocations as an applications dependency
normally requires that the application have the same dependencies as the application that
produced the dumped image. Dumping shared objects, and the various flags associated
with relocation processing, have some specialized uses. However, the technique is
intended as a building block for future technology.

The following code fragment, which is part of the dynamic executable a.out, can be used
to create a new version of the dynamic executable:

static char * dumped = 0;
const char * opath = "/.a.out.new";

.....

if (dumped == 0) {
    char buffer[100];
    int size;
    time_t seconds;
    ..... 
    
    /* Perform data initialization */
seconds = time((time_t *)0);
size = cftime(buffer, (char *)0, &seconds);

if ((dumped = (char *)malloc(size + 1)) == 0) {
    (void) printf("malloc failed: %s\n", strerror(errno));
    return (1);
}
(void) strcpy(dumped, buffer);
......

/*
 * Tear down any undesirable data initializations and
 * dump the dynamic executables memory image.
 */
_exithandle();
_exit(dldump(0, opath, RTLD_MEMORY));
}

(void) printf("Dumped: %s\n", dumped);

Any modifications made to the dynamic executable, up to the point the dldump() call is
made, are saved in the new object a.out.new. This mechanism allows the executable to
update parts of its data segment and heap prior to creating the new object. In this case,
the date the executable is dumped is saved in the new object. The new object can then be
executed without having to carry out the same (presumably expensive) initialization.

For greatest flexibility, this example does not save any relocated information. The ele-
ments of the dynamic executable ipath that have been modified by relocations at process
startup, that is, references to external functions, are returned to the values of these ele-
ments as they existed in the ipath disc file. This preservation of relocation records allows
the new dynamic executable to be flexible, and correctly bind and initialize to its depen-
dencies when executed on the same or newer upgrades of the OS.

Fixing relocations by applying some of the relocation flags would bind the new object to
the dependencies presently mapped as part of the process calling dldump(). It may also
remove necessary copy relocation processing required for the correct initialization of its
shared object dependencies. Therefore, if the new dynamic executables’ dependencies
have no specialized initialization requirements, the executable may still only interact
correctly with the dependencies to which it binds if they were mapped to the same loca-
tions as they were when dldump() was called.

Note that for dynamic executables, RTLD_REL_RELATIVE, RTLD_REL_EXEC, and
RTLD_REL_SELF have no effect, as relocations within the dynamic executable will have
been fixed when it was created by Id(1).

When RTLD_MEMORY is used, care should be taken to insure that dumped data sections
that reference external objects are not reused without appropriate re-initialization. For
example, if a data item contains a file descriptor, a variable returned from a shared object,
or some other external data, and this data item has been initialized prior to the `dldump()` call, its value will have no meaning in the new dumped image.

When `RTLD_MEMORY` is used, any modification to a data item that is initialized via a relocation whose relocation record will be retained in the new image will effectively be lost or invalidated within the new image. For example, if a pointer to an external object is incremented prior to the `dldump()` call, this data item will be reset to its disk file contents so that it can be relocated when the new image is used; hence, the previous increment is lost.

Non-idempotent data initializations may prevent the use of `RTLD_MEMORY`. For example, the addition of elements to a linked-list via `init` sections can result in the linked-list data being captured in the new image. Running this new image may result in `init` sections continuing to add new elements to the list without the prerequisite initialization of the list head. It is recommended that `_exithandle(3C)` be called before `dldump()` to tear down any data initializations established via initialization code. Note that this may invalidate the calling image; thus, following the call to `dldump()`, only a call to `_exit(2)` should be made.

**ATTRIBUTES**

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Availability</td>
<td>SUNWcsu</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**SEE ALSO**

`ld(1), ld.so.1(1), strip(1), _exit(2), brk(2), exec(2), _exithandle(3C), dladdr(3X), dlclose(3X), dierror(3X), dlopen(3X), dlerror(3X), end(3C), malloc(3C), attributes(5)`

**NOTES**

Any `NOBITS` sections within the `ipath` are expanded to `PROGBITS` sections within the `opath`. `NOBITS` sections occupy no space within an ELF file image. They declare memory that must be created and zero-filled when the object is mapped into the runtime environment. `.bss` is a typical example of this section type. `PROGBITS` sections, on the other hand, hold information defined by the object within the ELF file image. This section conversion reduces the runtime initialization cost of the new dumped object but increases the objects’ disk space requirement.

When a shared object is dumped, and relocations are applied which are knowledgeable of the base address of the mapped object, the new object is fixed to this new base address and thus its ELF type is reclassified to be a dynamic executable. This new object can be processed by the runtime linker, but is not valid as input to the link-editor.

If relocations are applied to the new object, any remaining relocation records will be reorganized for better locality of reference. The relocation sections are renamed to `.SUNW_reloc` and the association to the section they were to relocate is lost. Only the
offset of the relocation record itself is meaningful. This change does not make the new object invalid to either the runtime linker or link-editor, but may reduce the objects analysis via some ELF readers.
NAME dLError – get diagnostic information

SYNOPSIS
c
[
 \[ flag \] \] file \[ file \] \-ldl \[ library \]

\#include <dlfcn.h>

char *dLError(void);

DESCRIPTION dLError() is one of a family of routines that give the user direct access to the dynamic
linking facilities. (See Linker and Libraries Guide). These routines are made available via
the library loaded when the option \-ldl is passed to the link-editor.

Note: These routines are available to dynamically-linked processes ONLY.

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.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO ld(1), dladdr(3X), dlclose(3X), ddump(3X), dlopen(3X), dlsym(3X), attributes(5)

Linker and Libraries Guide

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
dlinfo – dynamic load information

SYNOPSIS
cc [ flag ... ] file ... -ldl [ library ... ]
#include <dlfcn.h>
int dlinfo(void *handle, int request, void *p);

DESCRIPTION
dlinfo() extracts information about a dynamically-loaded object. This interface is loosely modeled after the ioctl() interface. request and a third argument with varying type are passed to dlinfo(). The action taken by dlinfo() depends on the value of the request provided. handle is a value returned from a dlopen() or dlmopen() call.

The following are possible values for request to be passed into dlinfo():

RTLD_DI_LMID obtains the id for the link-map list upon which the handle is loaded. p is a Lmid_t pointer (Lmid_t *p).

RTLD_DI_LINKMAP obtains the Link_map for the handle specified. p points to a Link_map pointer (Link_map **p). The actual storage for the Link_map structure is maintained by ld.so.1.

The Link_map structure includes the following members:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>l_addr</td>
<td>The base address of the object loaded into memory.</td>
</tr>
<tr>
<td>l_name</td>
<td>Full name of the loaded object. This is the filename of the object as referenced by ld.so.1.</td>
</tr>
<tr>
<td>l_ld</td>
<td>Points to the SHT_DYNAMIC structure.</td>
</tr>
<tr>
<td>l_next</td>
<td>The next Link_map on the link-map list, other objects on the same link-map list as the current object may be examined by following the and l_prev fields.</td>
</tr>
<tr>
<td>l_prev</td>
<td>The previous Link_map on the link-map list.</td>
</tr>
<tr>
<td>l_refname</td>
<td>If the object referenced is a filter this field points to the name of the object being filtered. If the object is not a filter, this field will be 0. See Linker and Libraries Guide.</td>
</tr>
</tbody>
</table>

modified 24 Jan 1997
ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO

ld(1), ioctl(2), dlclose(3X), dldump(3X), dlerror(3X), dlmopen(3X), dlopen(3X), dlSYM(3X), attributes(5)

Linker and Libraries Guide

NOTES

These routines are available to dynamically-linked processes only.
NAME
dlopen, dlmopen – gain access to an executable object file

SYNOPSIS
cc [ flag ...] file ... -ldl [ library ... ]
#include <dlfcn.h>
void * dlopen(const char * pathname, int mode);
void * dlmopen(Lmid_t lmid, const char * pathname, int mode);

DESCRIPTION
dlopen() and dlmopen() are members of a family of routines that give the user direct access to the dynamic linking facilities. (See Linker and Libraries Guide). These routines are made available through the library loaded when the option –ldl is passed to the link-editor.

Note: These routines are available to dynamically-linked processes ONLY.
dlopen() makes an executable object file available to a running process. dlopen() returns to the process a handle which the process may use on subsequent calls to dlsym() and dlclose(). The value of this handle should not be interpreted in any way by the process. pathname is the path name of the object to be opened. A path name containing an embedded ‘/’ is interpreted as an absolute path or relative to the current directory; otherwise, the set of search paths currently in effect by the runtime linker will be used to locate the specified file. See NOTES below.

Any dependencies recorded within pathname are also loaded as part of the dlopen(). These dependencies are searched, in the order they are loaded, to locate any additional dependencies. This process will continue until all the dependencies of pathname are loaded. This dependency tree is referred to as a group.

If the value of pathname is 0, dlopen() provides a handle on a global symbol object. This object provides access to the symbols from an ordered set of objects consisting of the original program image file, together with any dependencies loaded at program startup, and any objects that were loaded using dlopen() together with the RTLD_GLOBAL flag. As the latter set of objects can change during process execution, the set identified by handle can also change dynamically.

dlmopen() is identical to the dlopen() routine, except that an identifying link-map id (lmid) is passed into it. This link-map id informs the dynamic linking facilities upon which link-map list to load the object. See Linker and Libraries Guide.

The mode parameter describes how dlopen() will operate upon pathname with respect to the processing of relocations and the scope of visibility of the symbols provided by pathname and its dependencies. When an object is brought into the address space of a process, it may contain references to symbols for which addresses are not known until the object is loaded. These references must be relocated before the symbols can be accessed. The mode parameter governs when these relocations take place and may have the following values:

RTLD_LAZY Only references to data symbols are relocated when the object is first loaded. References to functions are not relocated until a given function is invoked for the first time. This mode should improve
performance, since a process may not reference all of the functions in any given object. This behavior imitates the normal loading of dependencies during process initialization.

**RTLD_NOW**

All necessary relocations are performed when the object is first loaded. This may waste some processing, if relocations are performed for functions that are never referenced. This behavior may be useful for applications that need to know as soon as an object is loaded that all symbols referenced during execution will be available. This option mimics the loading of dependencies when the environment variable `LD_BIND_NOW` is in effect.

To determine the scope of visibility for symbols loaded with a `dlopen()` invocation, the `mode` parameter should be bitwise or'ed with one of the following values:

**RTLD_GLOBAL**

The object’s global symbols are made available for the relocation processing of any other object. In addition, symbol lookup using `dlopen(NULL, mode)` and an associated `dlsym()`, allows objects loaded with `RTLD_GLOBAL` to be searched.

**RTLD_LOCAL**

The object’s globals symbols are only available for the relocation processing of other objects that comprise the same group.

The program image file, and any objects loaded at program startup, have the mode `RTLD_GLOBAL`. The mode `RTLD_LOCAL` is the default mode for any objects acquired with `dlopen()`. A local object may be a dependency of more then one group. Any object of mode `RTLD_LOCAL` that is referenced as a dependency of an object of mode `RTLD_GLOBAL` will be promoted to `RTLD_GLOBAL`. In other words, the `RTLD_LOCAL` mode is ignored.

Any object loaded by `dlopen()` that requires relocations against global symbols can reference the symbols in any `RTLD_GLOBAL` object, which are at least the program image file and any objects loaded at program startup, from the object itself, and from any dependencies the object references. However, the `mode` parameter may also be bitwise or’ed with the following values to affect the scope of symbol availability:

**RTLD_GROUP**

Only symbols from the associated group are made available for relocation. A group is established from the defined object and all the dependencies of that object. A group must be completely self-contained. All dependency relationships between the members of the group must be sufficient to satisfy the relocation requirements of each object that comprises the group.

**RTLD_PARENT**

The symbols of the object initiating the `dlopen()` call are made available to the objects obtained by `dlopen()` itself. This option is useful when hierarchical `dlopen()` families are created. Note that although the parent object can supply symbols for the relocation of this object, the parent object is not available to `dlsym()` through the returned handle.

**RTLD_WORLD**

Only symbols from `RTLD_GLOBAL` objects are made available for relocation.
The default modes for `dlopen()` are both `RTLD_WORLD` and `RTLD_GROUP`. These modes are or'ed together if an object is required by different dependencies specifying differing modes.

The following modes provide additional capabilities outside of relocation processing:

- **RTLD_NODELETE**: The specified object will not be deleted from the address space as part of a `dlclose()`.
- **RTLD_NOLOAD**: The specified object is not loaded as part of the `dlopen()`, but a valid handle is returned if the object already exists as part of the process address space. Additional modes can be specified and will be or'ed with the present mode of the object and its dependencies. The `RTLD_NOLOAD` mode provides a means of querying the presence, or promoting the modes, of an existing dependency.

The `Lmid` passed to `dlmopen()` identifies the link-map list where the object will be loaded. This can be any valid `Lmid_t` returned by `dlinfo()` or one of the following special values:

- **LM_ID_BASE**: Load the object on the applications link-map list.
- **LM_ID_LDSO**: Load the object on the dynamic linkers (`ld.so.1`) link-map list.
- **LM_ID_NEWLM**: Causes the object to create a new link-map list as part of loading. It is vital that any object opened on a new link-map list have all of its dependencies expressed because there will be no other objects on this link-map.

**RETURN VALUES**

If `pathname` cannot be found, cannot be opened for reading, is not a shared or relocatable object, or if an error occurs during the process of loading `pathname` or relocating its symbolic references, `dlopen()` will return NULL. More detailed diagnostic information will be available through `dlerror()`.

**ATTRIBUTES**

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**SEE ALSO**

`ld(1), ld.so.1(1), dladdr(3X), dlclose(3X), dldump(3X), dlerror(3X), dlinfo(3X), dlsym(3X), attributes(5)`

`Linker and Libraries Guide`

**NOTES**

If other objects were link-edited with `pathname` when `pathname` was built, that is, the `pathname` has dependencies on other objects, those objects will automatically be loaded by `dlopen()`. The directory search path used to find both `pathname` and the other `needed` objects may be affected by setting the environment variable `LD_LIBRARY_PATH`, which is analyzed once at process startup, and from a runpath setting within the object from which the call to `dlopen()` originated. These search rules will only be applied to path names that do not contain an embedded ‘/’. Objects whose names resolve to the same absolute or relative path name may be opened any number of times using `dlopen()`;

modified 19 Mar 1997  SunOS 5.6  3X-393
however, the object referenced will only be loaded once into the address space of the current process.

When loading shared objects the application should open a specific version of the shared object, as opposed to relying on the version of the shared object pointed to by the symbolic link.

When building objects that are to be loaded on a new link-map list (see LM_ID_NEWLM), some precautions need to be taken. In general, all dependencies must be included when building an object. Also, include /usr/lib/libmalloc.so.1 before /usr/lib/libc.so.1 when building an object.

When an object is loaded into memory on a new link-map list, it is isolated from the main running program. There are certain global resources that are only usable from one link-map list. A few examples of these would be the sbrk() based malloc(), libthread(), and the signal vectors. Because of this, care must be taken not to use any of these resources on any but the primary link-map list. These issues are discussed in further detail in the Linker and Libraries Guide.

Some symbols defined in dynamic executables or shared objects may not be available to the runtime linker. The symbol table created by ld for use by the runtime linker might contain only a subset of the symbols defined in the object.
NAME
dlsym – get the address of a symbol in a shared object

SYNOPSIS
c\[ flag \ldots \] file \ldots –ldl [ library \ldots ]
#include <dlfcn.h>
void *dlsym(void *handle, const char *name);

DESCRIPTION
dlsym() is one of a family of routines that give the user direct access to the dynamic linking facilities. (See Linker and Libraries Guide). These routines are made available via the library loaded when the option –ldl is passed to the link-editor.

Note: These routines are available to dynamically-linked processes ONLY.
dlsym() allows a process to obtain the address of a symbol defined within a shared object. handle is either the value returned from a call to dlopen() or one of the special flags RTLD_NEXT or RTLD_DEFAULT. name is the symbol’s name as a character string.

In the case of a handle returned from dlopen() the corresponding shared object must not have been closed using dlclose(). dlsym() will search for the named symbol in all shared objects loaded automatically as a result of loading the object referenced by handle. See dlopen(3X).

In the case of the special handle RTLD_NEXT, dlsym() will search for the named symbol in the objects that were loaded following the object from which the dlsym() call is being made.

In the case of the special handle RTLD_DEFAULT, dlsym() will search for the named symbol, starting with the first object loaded and proceeding through the list of loaded objects until a match is found. This search follows the default model employed to relocate all objects within the process.

In the case of both RTLD_NEXT and RTLD_DEFAULT, if the objects being searched have been loaded from dlopen() calls, dlsym() will search the object only if the caller is part of the same dlopen() dependency hierarchy, or if the object was given global search access. See dlopen(3X) for a discussion of the RTLD_GLOBAL mode.

RETURN VALUES
If handle 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 handle, dlsym() will return NULL. More detailed diagnostic information is available through dlerror().

EXAMPLES
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.

```c
void *handle;
int *iptr, (*fptr)(int);

/* open the needed object */
handle = dlopen("/usr/home/me/libfoo.so.1", RTLD_LAZY);
```

modified 12 Mar 1997

SunOS 5.6

3X-395
The following code fragment shows how `dlsym()` can be used to check to see that a particular function is defined and to call it only if it is.

```c
int (*fptr)();

if ((fptr = (int (*)(int))dlsym(RTLD_DEFAULT, "my_function") != NULL) {
   (*fptr)();
}
```

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**SEE ALSO**
`ld(1), dladdr(3X), dlclose(3X), dlerror(3X), dlopen(3X), attributes(5)`

`Linker and Libraries Guide`
NAME

DmiAddComponent, DmiAddGroup, DmiAddLanguage, DmiDeleteComponent,
DmiDeleteGroup, DmiDeleteLanguage – Management Interface database administration
functions

SYNOPSIS

cc [ flag ... ] file ... -ldmimi -ldmi -lnsl -lrwtool [ library ... ]
#include <server.h>
#include <miapi.h>

bool_t DmiAddComponent(DmiAddComponentIN argin, 
    DmiAddComponentOUT *result, DmiRpcHandle *dmi_rpc_handle);
bool_t DmiAddGroup(DmiAddGroupIN argin, 
    DmiAddGroupOUT *result, DmiRpcHandle *dmi_rpc_handle);
bool_t DmiAddLanguage(DmiAddLanguageIN argin, 
    DmiAddLanguageOUT *result, DmiRpcHandle *dmi_rpc_handle);
bool_t DmiDeleteComponent(DmiDeleteComponentIN argin, 
    DmiDeleteComponentOUT *result, DmiRpcHandle *dmi_rpc_handle);
bool_t DmiDeleteGroup(DmiDeleteGroupIN argin, 
    DmiDeleteGroupOUT *result, DmiRpcHandle *dmi_rpc_handle);
bool_t DmiDeleteLanguage(DmiDeleteLanguageIN argin, 
    DmiDeleteLanguageOUT *result, DmiRpcHandle *dmi_rpc_handle);

DESCRIPTION

The database administration functions add a new component to the database or add a
ew language mapping for an existing component. You may also remove an existing
component, remove a specific language mapping, or remove a group from a component.

The DmiAddComponent() function adds a new component to the DMI database. It takes
the name of a file, or the address of memory block containing MIF data, checks the data
for adherence to the DMI MIF grammar, and installs the MIF in the database. The pro-
procedure returns a unique component ID for the newly installed component. The argin
parameter is an instance of a DmiAddComponentIN structure containing the following
members:

DmiHandle_t handle; /* an open session handle */
DmiFileDataList_t *fileData; /* MIF data for component */

The result parameter is a pointer to a DmiAddComponentOUT structure containing the
following members:

DmiErrorStatus_t error_status;
DmiId_t compId; /* SP-allocated component ID */
DmiStringList_t *errors; /* installation error messages */

The DmiAddLanguage() function adds a new language mapping for an existing com-
ponent in the database. It takes the name of a file, or the address of memory block con-
taining translated MIF data, checks the data for adherence to the DMI MIF grammar, and
installs the language MIF in the database. The argin parameter is an instance of a
**DmiAddComponent (3X)**

**Miscellaneous Library Functions**

**DmiAddLanguageIN** structure containing the following members:

- `DmiHandle_t handle;` /* an open session handle */
- `DmiFileDataList_t *fileData;` /* language mapping file */
- `DmiId_t compId;` /* component to access */

The *result* parameter is a pointer to a **DmiAddLanguageOUT** structure containing the following members:

- `DmiErrorStatus_t error_status;`
- `DmiStringList_t *errors;` /* installation error messages */

The **DmiAddGroup( )** function adds a new group to an existing component in the database. It takes the name of a file, or the address of memory block containing the group’s MIF data, checks the data for adherence to the DMI MIF grammar, and installs the group MIF in the database. The *argin* parameter is an instance of a **DmiAddGroupIN** structure containing the following members:

- `DmiHandle_t handle;` /* an open session handle */
- `DmiFileDataList_t *fileData;` /* MIF file data for group */
- `DmiId_t compId;` /* component to access */

The *result* parameter is a pointer to a **DmiAddGroupOUT** structure containing the following members:

- `DmiErrorStatus_t error_status;`
- `DmiId_t groupId;` /* SP-allocated group ID */
- `DmiStringList_t *errors;` /* installation error messages */

The **DmiDeleteComponent( )** function removes an existing component from the database. The *argin* parameter is an instance of a **DmiDeleteComponentIN** structure containing the following members:

- `DmiHandle_t handle;` /* an open session handle */
- `DmiId_t compId;` /* component to delete */

The *result* parameter is a pointer to a **DmiDeleteComponentOUT** structure containing the following members:

- `DmiErrorStatus_t error_status;`

The **DmiDeleteLanguage( )** function removes a specific language mapping for a component. You specify the language string and component ID. The *argin* parameter is an instance of a **DmiDeleteLanguageIN** structure containing the following members:

- `DmiHandle_t handle;` /* an open session handle */
- `DmiString_t *language;` /* language to delete */
- `DmiId_t compId;` /* component to access */

The *result* parameter is a pointer to a **DmiDeleteLanguageOUT** structure containing the following members:
The `DmiDeleteGroup()` function removes a group from a component. The caller specifies the component and group IDs. The `argin` parameter is an instance of a `DmiDeleteGroupIN` structure containing the following members:

- `DmiHandle_t handle; /* an open session handle */`
- `DmiId_t compId; /* component containing group */`
- `DmiId_t groupId; /* group to delete */`

The `result` parameter is a pointer to a `DmiDeleteGroupOUT` structure containing the following members:

- `DmiErrorStatus_t error_status;`

**RETURN VALUES**

The `DmiAddComponent()` function returns the following possible values:

- `DMIERR_NO_ERROR`
- `DMIERR_ILLEGAL_RPC_HANDLE`
- `DMIERR_OUT_OF_MEMORY`
- `DMIERR_ILLEGAL_PARAMETER`
- `DMIERR_SP_INACTIVE`
- `DMIERR_FILE_ERROR`
- `DMIERR_BAD_SCHEMA_DESCRIPTION_FILE`

The `DmiAddGroup()` function returns the following possible values:

- `DMIERR_NO_ERROR`
- `DMIERR_ILLEGAL_RPC_HANDLE`
- `DMIERR_OUT_OF_MEMORY`
- `DMIERR_ILLEGAL_PARAMETER`
- `DMIERR_SP_INACTIVE`
- `DMIERR_INSUFFICIENT_PRIVILEGES`
- `DMIERR_COMPONENT_NOT_FOUND`
- `DMIERR_FILE_ERROR`
- `DMIERR_BAD_SCHEMA_DESCRIPTION_FILE`

The `DmiAddLanguage()` function returns the following possible values:

- `DMIERR_NO_ERROR`
- `DMIERR_ILLEGAL_RPC_HANDLE`
- `DMIERR_OUT_OF_MEMORY`
- `DMIERR_ILLEGAL_PARAMETER`
- `DMIERR_SP_INACTIVE`
- `DMIERR_COMPONENT_NOT_FOUND`
- `DMIERR_FILE_ERROR`
- `DMIERR_BAD_SCHEMA_DESCRIPTION_FILE`
The `DmiDeleteComponent()` function returns the following possible values:

- DMIERR_NO_ERROR
- DMIERR_ILLEGAL_RPC_HANDLE
- DMIERR_OUT_OF_MEMORY
- DMIERR_ILLEGAL_PARAMETER
- DMIERR_SP_INACTIVE
- DMIERR_INSUFFICIENT_PRIVILEGES
- DMIERR_COMPONENT_NOT_FOUND
- DMIERR_FILE_ERROR

The `DmiDeleteGroup()` function returns the following possible values:

- DMIERR_NO_ERROR
- DMIERR_ILLEGAL_RPC_HANDLE
- DMIERR_OUT_OF_MEMORY
- DMIERR_ILLEGAL_PARAMETER
- DMIERR_SP_INACTIVE
- DMIERR_INSUFFICIENT_PRIVILEGES
- DMIERR_COMPONENT_NOT_FOUND
- DMIERR_FILE_ERROR

The `DmiDeleteLanguage()` function returns the following possible values:

- DMIERR_NO_ERROR
- DMIERR_ILLEGAL_RPC_HANDLE
- DMIERR_OUT_OF_MEMORY
- DMIERR_ILLEGAL_PARAMETER
- DMIERR_SP_INACTIVE
- DMIERR_INSUFFICIENT_PRIVILEGES
- DMIERR_COMPONENT_NOT_FOUND
- DMIERR_FILE_ERROR

**ATTRIBUTES**

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO `attributes(5)`
NAME
DmiAddRow, DmiDeleteRow, DmiGetAttribute, DmiGetMultiple, DmiSetAttribute,
DmiSetMultiple – Management Interface operation functions

SYNOPSIS
cc [ flag ...] file ... -ldmimi -ldmi -lnsl -lrwtool [ library ... ]
#include <server.h>
#include <miapi.h>

bool_t DmiAddRow(DmiAddRowIN argin,
                  DmiAddRowOUT *result, DmiRpcHandle *dmi_rpc_handle);
bool_t DmiDeleteRow(DmiDeleteRowIN argin,
                    DmiDeleteRowOUT *result, DmiRpcHandle *dmi_rpc_handle);
bool_t DmiGetAttribute(DmiGetAttributeIN argin,
                       DmiGetAttributeOUT *result, DmiRpcHandle *dmi_rpc_handle);
bool_t DmiGetMultiple(DmiGetMultipleIN argin,
                      DmiGetMultipleOUT *result, DmiRpcHandle *dmi_rpc_handle);
bool_t DmiSetAttribute(DmiSetAttributeIN argin,
                       DmiSetAttributeOUT *result, DmiRpcHandle *dmi_rpc_handle);
bool_t DmiSetMultiple(DmiSetMultipleIN argin,
                     DmiSetMultipleOUT *result, DmiRpcHandle *dmi_rpc_handle);

DESCRIPTION
The operation functions provide a method for retrieving a single value from the Service
Provider and for setting a single attribute value. In addition, you may also retrieve attri-
bute values from the Service Provider. You may perform a set operation on an attribute
or a list of attributes and add or delete a row from an existing table.

The DmiAddRow() function adds a row to an existing table. The rowData parameter
contains the full data, including key attribute values, for a row. It is an error for the key
list to specify an existing table row. The argin parameter is an instance of a DmiAd-

dRowIN structure containing the following members:

DmiHandle_t handle; /* An open session handle */
DmiRowData_t *rowData; /* Attribute values to set */

The result parameter is a pointer to a DmiAddRowOUT structure containing the follow-
ing members:

DmiErrorStatus_t error_status;

DmiDeleteRow() function removes a row from an existing table. The key list must
specify valid keys for a table row. The argin parameter is an instance of a

DmiDeleteRowIN structure containing the following members:

DmiHandle_t handle; /* An open session handle */
DmiRowData_t *rowData; /* Row to delete */
The `result` parameter is a pointer to a `DmiDeleteRowOUT` structure containing the following members:

```
DmiErrorStatus_t error_status;
```

The `DmiGetAttribute()` function provides a simple method for retrieving a single attribute value from the Service Provider. The `compId`, `groupId`, `attribId`, and `keyList` identify the desired attribute. The resulting attribute value is returned in a newly allocated `DmiDataUnion` structure. The address of this structure is returned through the `value` parameter. The `argin` parameter is an instance of a `DmiListComponentsIN` structure containing the following members:

```
DmiHandle_t handle; /* an open session handle */
DmiId_t compId; /* Component to access */
DmiId_t groupId; /* Group within component */
DmiId_t attribId; /* Attribute within a group */
DmiAttributeValues_t *keyList; /* Keylist to specify a table row */
```

The `result` parameter is a pointer to a `DmiGetAttributeOUT` structure containing the following members:

```
DmiErrorStatus_t error_status;
DmiDataUnion_t *value; /* Attribute value returned */
```

The `DmiGetMultiple()` function retrieves attribute values from the Service Provider. This procedure may get the value for an individual attribute, or for multiple attributes across groups, components, or rows of a table.

The `DmiSetAttribute()` function provides a simple method for setting a single attribute value. The `compId`, `groupId`, `attribId`, and `keyList` identify the desired attribute. The `setMode` parameter defines the procedure call as a Set, Reserve, or Release operation. The new attribute value is contained in the `DmiDataUnion` structure whose address is passed in the `value` parameter. The `argin` parameter is an instance of a `DmiSetAttributeIN` structure containing the following members:

```
DmiHandle_t handle;
DmiId_t compId;
DmiId_t groupId;
DmiId_t attribId;
DmiAttributeValues_t *keyList;
DmiSetMode_t setMode;
DmiDataUnion_t *value;
```

The `result` parameter is a pointer to a `DmiSetAttributeOUT` structure containing the following members:

```
DmiErrorStatus_t error_status;
```
The `DmiSetMultiple()` function performs a set operation on an attribute or list of attributes. Set operations include actually setting the value, testing and reserving the attribute for future setting, or releasing the set reserve. These variations on the set operation are specified by the parameter `setMode`. The `argin` parameter is an instance of a `DmiSetMultipleIN` structure containing the following members:

```
DmiHandle_t handle; /* An open session handle */
DmiSetMode_t setMode; /* set, reserve, or release */
DmiMultiRowData_t *rowData; /* Attribute values to set */
```

The `result` parameter is a pointer to a `DmiSetMultipleOUT` structure containing the following members:

```
DmiErrorStatus_t error_status;
```

The `rowData` array describes the attributes to set, and contains the new attribute values. Each element of rowData specifies a component, group, key list (for table accesses), and attribute list to set. No data is returned from this function.

**RETURN VALUES**

The `DmiAddRow()` function returns the following possible values:

- `DMIERR_NO_ERROR`
- `DMIERR_ILLEGAL_RPC_HANDLE`
- `DMIERR_OUT_OF_MEMORY`
- `DMIERR_ILLEGAL_PARAMETER`
- `DMIERR_SP_INACTIVE`
- `DMIERR_VALUE_UNKNOWN`
- `DMIERR_COMPONENT_NOT_FOUND`
- `DMIERR_GROUP_NOT_FOUND`
- `DMIERR_ILLEGAL_KEYS`
- `DMIERR_DIRECT_INTERFACE_NOT_REGISTERED`
- `DMIERR_UNKNOWN_CI_REGISTRY`
- `DMIERR_VALUE_UNKNOWN`
- `DMIERR_UNABLE_TO_ADD_ROW`

The `DmiDeleteRow()` function returns the following possible values:

- `DMIERR_NO_ERROR`
- `DMIERR_ILLEGAL_RPC_HANDLE`
- `DMIERR_OUT_OF_MEMORY`
- `DMIERR_ILLEGAL_PARAMETER`
- `DMIERR_SP_INACTIVE`
- `DMIERR_ATTRIBUTE_NOT_FOUND`
- `DMIERR_COMPONENT_NOT_FOUND`
- `DMIERR_GROUP_NOT_FOUND`
- `DMIERR_ILLEGAL_KEYS`
- `DMIERR_ILLEGAL_TO_GET`
- `DMIERR_DIRECT_INTERFACE_NOT_REGISTERED`
- `DMIERR_ROW_NOT_FOUND`
The `DmiGetAttribute()` function returns the following possible values:

- DMIERR_NO_ERROR
- DMIERR_ILLEGAL_RPC_HANDLE
- DMIERR_OUT_OF_MEMORY
- DMIERR_ILLEGAL_PARAMETER
- DMIERR_SP_INACTIVE
- DMIERR_ATTRIBUTE_NOT_FOUND
- DMIERR_COMPONENT_NOT_FOUND
- DMIERR_GROUP_NOT_FOUND
- DMIERR_ILLEGAL_KEYS
- DMIERR_ILLEGAL_TO_GET
- DMIERR_DIRECT_INTERFACE_NOT_REGISTERED
- DMIERR_ROW_NOT_FOUND
- DMIERR_UNKNOWN_CI_REGISTRY
- DMIERR_FILE_ERROR
- DMIERR_VALUE_UNKNOWN

The `DmiGetMultiple()` function returns the following possible values:

- DMIERR_NO_ERROR
- DMIERR_ILLEGAL_RPC_HANDLE
- DMIERR_OUT_OF_MEMORY
- DMIERR_ILLEGAL_RPC_PARAMETER
- DMIERR_SP_INACTIVE
- DMIERR_ATTRIBUTE_NOT_FOUND
- DMIERR_COMPONENT_NOT_FOUND
- DMIERR_GROUP_NOT_FOUND
- DMIERR_ILLEGAL_KEYS
- DMIERR_ILLEGAL_TO_GET
- DMIERR_DIRECT_INTERFACE_NOT_REGISTERED
- DMIERR_ROW_NOT_FOUND
- DMIERR_UNKNOWN_CI_REGISTRY
- DMIERR_FILE_ERROR
- DMIERR_VALUE_UNKNOWN

The `DmiSetAttribute()` function returns the following possible values:

- DMIERR_NO_ERROR
- DMIERR_ILLEGAL_RPC_HANDLE
- DMIERR_OUT_OF_MEMORY
- DMIERR_ILLEGAL_PARAMETER
- DMIERR_SP_INACTIVE
- DMIERR_ATTRIBUTE_NOT_FOUND
The DmiSetMultiple() function returns the following possible values:

- DMIERR_NO_ERROR
- DMIERR_ILLEGAL_RPC_HANDLE
- DMIERR_OUT_OF_MEMORY
- DMIERR_ILLEGAL_PARAMETER
- DMIERR_SP_INACTIVE
- DMIERR_ATTRIBUTE_NOT_FOUND
- DMIERR_COMPONENT_NOT_FOUND
- DMIERR_GROUP_NOT_FOUND
- DMIERR_ILLEGAL_KEYS
- DMIERR_ILLEGAL_TO_SET
- DMIERR_DIRECT_INTERFACE_NOT_REGISTERED
- DMIERR_ROW_NOT_FOUND
- DMIERR_UNKNOWN_CI_REGISTRY
- DMIERR_FILE_ERROR
- DMIERR_VALUE_UNKNOWN

ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO attributes(5)
NAME
DmiGetConfig, DmiGetVersion, DmiRegister, DmiSetConfig, DmiUnregister – Management Interface initialization functions

SYNOPSIS
cc [ flag . . ] file . . . -ldmimi -ldmi -linsl -lrwtool [ library . . ]
#include <server.h>
#include <miapi.h>

bool_t DmiGetConfig(DmiGetConfigIN argin,
DmiGetConfigOUT *result, DmiRpcHandle *dmi_rpc_handle);

bool_t DmiGetVersion(DmiGetVersionIN argin,
DmiGetVersionOUT *result, DmiRpcHandle *dmi_rpc_handle);

bool_t DmiRegister(DmiRegisterIN argin,
DmiRegisterOUT *result, DmiRpcHandle *dmi_rpc_handle);

bool_t DmiSetConfig(DmiSetConfigIN argin,
DmiSetConfigOUT *result, DmiRpcHandle *dmi_rpc_handle);

bool_t DmiUnregister(DmiUnregisterIN argin,
DmiUnregisterOUT *result, DmiRpcHandle *dmi_rpc_handle);

DESCRIPTION
The Management Interface initialization functions enable you to register management applications to the Service Provider. You may also retrieve information about the Service Provider, get and set session configuration information for your session.

The DmiGetConfig() function retrieves the per-session configuration information. The configuration information consists of a string describing the current language being used for the session. The argin parameter is an instance of a DmiGetConfigIN structure containing the following member:

DmiHandle_t handle; /* an open session handle */

The result parameter is a pointer to a DmiGetConfigOUT structure containing the following members:

DmiErrorStatus_t error_status;
DmiString_t *language; /* current session language */

The DmiGetVersion() function retrieves information about the Service Provider. The management application uses the DmiGetVersion() procedure to determine the DMI specification level supported by the Service Provider. This procedure also returns the service provided description string, and may contain version information about the Service Provider implementation. The argin parameter is an instance of a DmiGetVersionIN structure containing the following member:

DmiHandle_t handle; /* an open session handle */

The result parameter is a pointer to a DmiGetVersionOUT structure containing the following members:
The `DmiRegister()` function provides the management application with a unique per-session handle. The Service Provider uses this procedure to initialize to an internal state for subsequent procedure calls made by the application. This procedure must be the first command executed by the management application. `argin` is an instance of a `DmiRegisterIN` structure containing the following member:

```c
DmiHandle_t handle; /* an open session handle */
```

The `result` parameter is a pointer to a `DmiRegisterOUT` structure containing the following members:

```c
DmiErrorStatus_t error_status;
DmiHandle_t *handle; /* an open session handle */
```

The `DmiSetConfig()` function sets the per-session configuration information. The configuration information consists of a string describing the language required by the management application. The `argin` parameter is an instance of a `DmiSetConfigIN` structure containing the following member:

```c
DmiHandle_t handle; /* an open session handle */
DmiString_t *language; /* current language required */
```

The `result` parameter is a pointer to a `DmiSetConfigOUT` structure containing the following member:

```c
DmiErrorStatus_t error_status;
```

The `DmiUnregister()` function is used by the Service Provider to perform end-of-session cleanup actions. On return from this function, the session handle is no longer valid. This function must be the last DMI command executed by the management application. The `argin` parameter is an instance of a `DmiUnregisterIN` structure containing the following member:

```c
DmiHandle_t handle; /* an open session handle */
```

The `result` parameter is a pointer to a `DmiUnregisterOUT` structure containing the following members:

```c
DmiErrorStatus_t error_status;
```

**RETURN VALUES**

The `DmiGetConfig()` function returns the following possible values:

```c
DMIERR_NO_ERROR
DMIERR_ILLEGAL_RPC_HANDLE
DMIERR_OUT_OF_MEMORY
DMIERR_ILLEGAL_PARAMETER
DMIERR_SP_INACTIVE
```

modified 17 Dec 1996  SunOS 5.6  3X-407
The `DmiGetVersion()` function returns the following possible values:

- DMIERR_NO_ERROR
- DMIERR_ILLEGAL_RPC_HANDLE
- DMIERR_OUT_OF_MEMORY
- DMIERR_SP_INACTIVE

The `DmiRegister()` function returns the following possible values:

- DMIERR_NO_ERROR
- DMIERR_ILLEGAL_RPC_HANDLE
- DMIERR_OUT_OF_MEMORY
- DMIERR_SP_INACTIVE

The `DmiSetConfig()` function returns the following possible values:

- DMIERR_NO_ERROR
- DMIERR_ILLEGAL_RPC_HANDLE
- DMIERR_OUT_OF_MEMORY
- DMIERR_ILLEGAL_PARAMETER
- DMIERR_SP_INACTIVE
- DMIERR_ILLEGAL_TO_SET

The `DmiUnRegister()` function returns the following possible values:

- DMIERR_NO_ERROR
- DMIERR_ILLEGAL_RPC_HANDLE
- DMIERR_OUT_OF_MEMORY
- DMIERR_ILLEGAL_PARAMETER
- DMIERR_SP_INACTIVE

**ATTRIBUTES**

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

**SEE ALSO** attributes(5)
NAME
DmiListAttributes, DmiListClassNames, DmiListComponents, DmiListComponentsByClass, DmiListGroups, DmiListLanguages – Management Interface listing functions

SYNOPSIS
cc [ flag ... ] file ... -ldmimi -ldmi -linsl -lrwtool [ library ... ]
#include <server.h>
#include <miapi.h>

bool_t DmiListAttributes(DmiListAttributesIN argin,
                        DmiListAttributesOUT *result, DmiRpcHandle *dmi_rpc_handle);

bool_t DmiListClassNames(DmiListClassNamesIN argin,
                        DmiListClassNamesOUT *result, DmiRpcHandle *dmi_rpc_handle);

bool_t DmiListComponents(DmiListComponentsIN argin,
                        DmiListComponentsOUT *result, DmiRpcHandle *dmi_rpc_handle);

bool_t DmiListComponentsByClass(DmiListComponentsByClassIN argin,
                                DmiListComponentsByClassOUT *result, DmiRpcHandle *dmi_rpc_handle);

bool_t DmiListGroups(DmiListGroupsIN argin,
                     DmiListGroupsOUT *result, DmiRpcHandle *dmi_rpc_handle);

bool_t DmiListLanguages(DmiListLanguagesIN argin,
                        DmiListLanguagesOUT *result, DmiRpcHandle *dmi_rpc_handle);

DESCRIPTION
The listing functions enables you to retrieve the names and the description of components in a system. You may also list components by class that match a specified criteria. The listing functions retrieve the set of language mappings installed for a specified component, retrieve class name strings for all groups in a component, retrieve a list of groups within a component, and retrieve the properties for one or more attributes in a group.

The DmiListComponents() function retrieves the name and (optionally) the description of components in a system. You may also list components by class that match a specified criteria. The listing functions retrieve the set of language mappings installed for a specified component, retrieve class name strings for all groups in a component, retrieve a list of groups within a component, and retrieve the properties for one or more attributes in a group.

The DmiListComponents() function retrieves the name and (optionally) the description of components in a system. Use this to interrogate a system to determine what components are installed. The argin parameter is an instance of a DmiListComponentsIN structure containing the following members:

- DmiHandle_t handle; /* an open session handle */
- DmiRequestMode_t requestMode; /* Unique, first, or next */
- DmiUnsigned_t maxCount; /* maximum number to return, 0 for all */
- DmiBoolean_t getPragma; /* get optional pragma string */
- DmiBoolean_t getDescription /* get optional component description */
- DmiId_t compId; /* component ID to start with */

The result parameter is a pointer to a DmiListComponentsOUT structure containing the following members:

modified 17 Dec 1996
SunOS 5.6
3X-409
An enumeration accesses a specific component or may be used to sequentially access all components in a system. The caller may choose not to retrieve the component description by setting the value `getDescription` to false. The caller may choose not to retrieve the pragma string by setting the value of `gutta-percha` to false. The `maxCount`, `requestMode`, and `complId` parameters allow the caller to control the information returned by the Service Provider. When the `requestMode` is `DMI_UNIQUE`, `complId` specifies the first component requested (or only component if `maxCount` is one). When the `requestMode` is `DMI_NEXT`, `complId` specifies the component just before the one requested. When `requestMode` is `DMI_FIRST`, `complId` is unused.

To control the amount of information returned, the caller sets `maxCount` to something other than zero. The service provider must honor this limit on the amount of information returned. When `maxCount` is 0 the service provider returns information for all components, subject to the constraints imposed by `requestMode` and `complId`.

The `DmiListComponentsByClass()` function lists components that match specified criteria. Use this function to determine if a component contains a certain group or a certain row in a table. A filter condition may be that a component contains a specified group class name or that it contains a specific row in a specific group. As with `DmiListComponents()`, the description and pragma strings are optional return values. `argin` is an instance of a `DmiListComponentsByClassIN` structure containing the following members:

- `DmiHandle_t handle;` /* an open session handle */
- `DmiRequestMode_t requestMode;` /* Unique, first or next */
- `DmiUnsigned_t maxCount;` /* maximum number to return, or 0 for all */
- `DmiBoolean_t getPragma;` /* get the optional pragma string */
- `DmiBoolean_t getDescription;` /* get optional component description */
- `DmiId_t complId;` /* component ID to start with */
- `DmiString_t *className;` /* group class name string to match */
- `DmiAttributeValues_t *keyList;` /* group row keys to match */

The `result` parameter is a pointer to a `DmiListComponentsByClassOUT` structure containing the following members:

- `DmiErrorStatus_t error_status;`
- `DmiComponentList_t *reply;` /* list of components */
The `DmiListLanguages()` function retrieves the set of language mappings installed for the specified component. The `argin` parameter is an instance of a `DmiListLanguagesIN` structure containing the following members:

- `DmiHandle_t handle;` /* An open session handle */
- `DmiUnsigned_t maxCount;` /* maximum number to return, or 0 for all */
- `DmiId_t compId;` /* Component to access */

The `result` parameter is a pointer to a `DmiListLanguagesOUT` structure containing the following members:

- `DmiErrorStatus_t error_status;`
- `DmiStringList_t *reply;` /* List of language strings */

The `DmiListClassNames()` function retrieves the class name strings for all groups in a component. This enables the management application to easily determine if a component contains a specific group, or groups. The `argin` parameter is an instance of a `DmiListClassNamesIN` structure containing the following members:

- `DmiHandle_t handle;` /* An open session handle */
- `DmiUnsigned_t maxCount;` /* maximum number to return, or 0 for all */
- `DmiId_t compId;` /* Component to access */

The `result` parameter is a pointer to a `DmiListClassNamesOUT` structure containing the following members:

- `DmiErrorStatus_t error_status;`
- `DmiClassNameList_t *reply;` /* List of class names and group IDs */

The `DmiListGroups()` function retrieves a list of groups within a component. With this function you can access a specific group or sequentially access all groups in a component. All enumerations of groups occur within the specified component and do not span components. The `argin` parameter is an instance of a `DmiListGroupsIN` structure containing the following members:

- `DmiHandle_t handle;` /* An open session handle */
- `DmiRequestMode_t requestMode;` /* Unique, first or next group */
- `DmiUnsigned_t maxCount;` /* Maximum number to return, or 0 for all */
- `DmiBoolean_t getPragma;` /* Get the optional pragma string */
- `DmiBoolean_t getDescription;` /* Get optional group description */
- `DmiId_t compId;` /* Component to access */
- `DmiId_t groupId;` /* Group to start with, refer to requestMode */
The **result** parameter is a pointer to a **DmiListGroupsOUT** structure containing the following members:

```c
DmiErrorStatus_t error_status;
DmiGroupList_t *reply;
```

The caller may choose not to retrieve the group description by setting the value **getDescription** to false. The caller may choose not to retrieve the pragma string by setting the value of **getPragma** to false. The **maxCount**, **requestMode**, and **groupId** parameters allow the caller to control the information returned by the Service Provider. When the **requestMode** is **DMI_UNIQUE**, **groupId** specifies the first group requested (or only group if **maxCount** is one). When the **requestMode** is **DMI_NEXT**, **groupId** specifies the group just before the one requested. When **requestMode** is **DMI_FIRST**, **groupId** is unused. To control the amount of information returned, the caller sets **maxCount** to something other than zero. The service provider must honor this limit on the amount of information returned. When **maxCount** is zero the service provider returns information for all groups, subject to the constraints imposed by **requestMode** and **groupId**.

The **DmiListAttributes()** function retrieves the properties for one or more attributes in a group. All enumerations of attributes occur within the specified group, and do not span groups. The **argin** parameter is an instance of a **DmiListAttributesIN** structure containing the following members:

```c
DmiHandle_t handle; /* An open session handle */
DmiRequestMode_t requestMode; /* Unique, first or next group */
DmiUnsigned_t maxCount; /* Maximum number to return, or 0 for all */
DmiBoolean_t getPragma; /* Get the optional pragma string */
DmiBoolean_t getDescription; /* Get optional group description */
DmiId_t compId; /* Component to access */
DmiId_t groupId; /* Group to access */
DmiId_t attribId; /* Attribute to start with, refer to requestMode */
```

The **result** parameter is a pointer to a **DmiListAttributesOUT** structure containing the following members:

```c
DmiErrorStatus_t error_status;
DmiAttributeList_t *reply; /* List of attributes */
```

You may choose not to retrieve the description string by setting the value of **getDescription** to false. Likewise, you may choose not to retrieve the pragma string by setting the value of **getPragma** to false. The **maxCount**, **requestMode**, and **attribId** parameters allow you to control the information returned by the Service Provider. When the **requestMode** is **DMI_UNIQUE**, **attribId** specifies the first attribute requested (or only attribute if **maxCount** is one). When the **requestMode** is **DMI_NEXT**, **attribId** specifies the attribute just before the one requested. When **requestMode** is **DMI_FIRST**, **attribId** is unused. To control the amount of information returned, the caller sets **maxCount** to something other than zero. The Service Provider must honor this limit on the amount of information.
returned. When `maxCount` is zero the service provider returns information for all attributes, subject to the constraints imposed by `requestMode` and `attribId`.

**RETURN VALUES**

The `DmiListAttributes()` function returns the following possible values:

- `DMIERR_NO_ERROR`
- `DMIERR_ILLEGAL_RPC_HANDLE`
- `DMIERR_OUT_OF_MEMORY`
- `DMIERR_ILLEGAL_PARAMETER`
- `DMIERR_SP_INACTIVE`
- `DMIERR_ATTRIBUTE_NOT_FOUND`
- `DMIERR_COMPONENT_NOT_FOUND`
- `DMIERR_GROUP_NOT_FOUND`
- `DMIERR_FILE_ERROR`

The `DmiListClassNames()` function returns the following possible values:

- `DMIERR_NO_ERROR`
- `DMIERR_ILLEGAL_RPC_HANDLE`
- `DMIERR_OUT_OF_MEMORY`
- `DMIERR_ILLEGAL_PARAMETER`
- `DMIERR_SP_INACTIVE`
- `DMIERR_COMPONENT_NOT_FOUND`
- `DMIERR_FILE_ERROR`

The `DmiListComponents()` function returns the following possible values:

- `DMIERR_NO_ERROR`
- `DMIERR_ILLEGAL_RPC_HANDLE`
- `DMIERR_OUT_OF_MEMORY`
- `DMIERR_ILLEGAL_PARAMETER`
- `DMIERR_SP_INACTIVE`
- `DMIERR_COMPONENT_NOT_FOUND`
- `DMIERR_FILE_ERROR`

The `DmiListComponentsByClass()` function returns the following possible values:

- `DMIERR_NO_ERROR`
- `DMIERR_ILLEGAL_RPC_HANDLE`
- `DMIERR_OUT_OF_MEMORY`
- `DMIERR_ILLEGAL_PARAMETER`
- `DMIERR_SP_INACTIVE`
- `DMIERR_COMPONENT_NOT_FOUND`
- `DMIERR_FILE_ERROR`

The `DmiListGroup()` function returns the following possible values:

- `DMIERR_NO_ERROR`
- `DMIERR_ILLEGAL_RPC_HANDLE`
- `DMIERR_OUT_OF_MEMORY`
The `DmiListLanguages()` function returns the following possible values:

- DMIERR_NO_ERROR
- DMIERR_ILLEGAL_RPC_HANDLE
- DMIERR_OUT_OF_MEMORY
- DMIERR_ILLEGAL_PARAMETER
- DMIERR_SP_INACTIVE
- DMIERR_COMPONENT_NOT_FOUND
- DMIERR_FILE_ERROR

**ATTRIBUTES**

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

**SEE ALSO**

attributes(5)
| NAME | DmiRegisterCi, DmiUnRegisterCi, DmiOriginateEvent – Service Provider functions for components |
| SYNOPSIS | `cc [ flag ... ] file ... –lci –ldmi –linsl –lrwtool [ library ... ]`
| | `#include <server.h>`
| | `#include <ciapi.h>`
| | `extern bool_t DmiRegisterCi(DmiRegisterCiIN argin, DmiRegisterCiOUT *result, DmiRpcHandle *dmi_rpc_handle);`
| | `bool_t DmiUnregisterCi(DmiUnregisterCiIN argin, DmiUnregisterCiOUT *result, DmiRpcHandle *dmi_rpc_handle);`
| | `bool_t DmiOriginateEvent(DmiOriginateEventIN argin, DmiOriginateEventOUT *result, DmiRpcHandle *dmi_rpc_handle);` |
| DESCRIPTION | These three functions provide component communication with the DMI through the Component Interface (CI).
| | Component instrumentation code may register with the Service Provider to override its current mechanism for the registered attributes. Instead of manipulating the data in the MIF database or invoking programs, the Service Provider calls the entry points provided in the registration call. Once the component unregisters, the Service Provider returns to a normal method of processing requests for the data as defined in the MIF. Component instrumentation can temporarily interrupt normal processing to perform special functions.
| | Registering attributes through the direct interface overrides attributes that are already being served through the direct interface. RPC is used for communication from the Service Provider to the component instrumentation.
| | For all three functions, `argin` is the parameter passed to initiate an RPC call, `result` is the result of the RPC call, and `dmi_rpc_handle` is an open session RPC handle.
| | The `DmiRegisterCi()` function registers a callable interface for components that have resident instrumentation code and/or to get the version of the Service Provider.
| | The `DmiUnRegisterCi()` function communicates to the Service Provider to remove a direct component instrumentation interface from the Service Provider table of registered interfaces.
| | The `DmiOriginateEvent()` function originates an event for filtering and delivery. Any necessary indication filtering is performed by this function (or by subsequent processing) before the event is forwarded to the management applications.
| | A component ID value of zero (0) specifies the event was generated by something that has not been installed as a component, and has no component ID.
RETURN VALUES

The `DmiRegisterCi()` function returns the following possible values:

- DMIERR_NO_ERROR
- DMIERR_ILLEGAL_HANDLE
- DMIERR_OUT_OF_MEMORY
- DMIERR_INSUFFICIENT_PRIVILEGES
- DMIERR_SP_INACTIVE
- DMIERR_ATTRIBUTE_NOT_FOUND
- DMIERR_COMPONENT_NOT_FOUND
- DMIERR_GROUP_NOT_FOUND
- DMIERR_DATABASE_CORRUPT
- DMIERR_OUT_OF_MEMORY
- DMIERR_ILLEGAL_DMI_LEVEL

The `DmiUnRegisterCi()` function returns the following possible values:

- DMIERR_NO_ERROR
- DMIERR_ILLEGAL_HANDLE
- DMIERR_OUT_OF_MEMORY
- DMIERR_INSUFFICIENT_PRIVILEGES
- DMIERR_SP_INACTIVE
- DMIERR_UNKNOWN_CI_REGISTRY

The `DmiOriginateEvent()` function returns the following possible values:

- DMIERR_NO_ERROR
- DMIERR_ILLEGAL_HANDLE
- DMIERR_OUT_OF_MEMORY
- DMIERR_INSUFFICIENT_PRIVILEGES
- DMIERR_SP_INACTIVE
- DMIERRUNKNOWN_CI_REGISTRY

ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO `attributes(5)`
Network Functions

NAME
doconfig – execute a configuration script

SYNOPSIS
cc [ flag ... ] file ...  

# include <sac.h>

int doconfig(int fildes, char *script, long rflag);

DESCRIPTION
doconfig() is a Service Access Facility library function that interprets the configuration
scripts contained in the files </etc/saf/pmtag/config>, </etc/saf/sysconfig>, and
</etc/saf/pmtag/svctag>, where pmtag specifies the tag associated with the port monitor,
and svctag specifies the service tag associated with a given service. See pmadm(1M) and
sacadm(1M).

script is the name of the configuration script; fildes is a file descriptor that designates the
stream to which stream manipulation operations are to be applied; rflag is a bitmask that
indicates the mode in which script is to be interpreted. If rflag is zero, all commands in
the configuration script are eligible to be interpreted. If rflag has the NOASSIGN bit set,
the assign command is considered illegal and will generate an error return. If rflag has
the NORUN bit set, the run and runwait commands are considered illegal and will gen-
erate error returns.

The configuration language in which script is written consists of a sequence of com-
mands, 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.

assign variable=value

Used to define environment variables. variable is the name of the environment
variable and value is the value to be assigned to it. The value assigned must be a
string constant; no form of parameter substitution is available. value 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.

push module1[, module2, module3,...]

Used to push STREAMS modules onto the stream designated by fildes. module1 is
the name of the first module to be pushed, module2 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.

pop [module]

Used to pop STREAMS modules off the designated stream. If pop is invoked with
no arguments, the top module on the stream is popped. If an argument is given,
modules will be popped one at a time until the named module is at the top of the
stream. If the named module is not on the designated stream, the stream is left as

modified 30 Dec 1996

SunOS 5.6

3N-417
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**
docfig() returns 0 if the script was interpreted successfully. If a command in the script fails, the interpretation of the script ceases at that point and a positive number is returned; this number indicates which line in the script failed. If a system error occurs, a value of −1 is returned. When a script fails, the process whose environment was being established should not be started.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

**SEE ALSO**
sh(1), pmadm(1M), sacadm(1M), attributes(5)

**NOTES**
This interface is unsafe in multithreaded applications. Unsafe interfaces should be called only from the main thread.
NAME  
door_bind, door_unbind – bind or unbind the current thread with the door server pool

SYNOPSIS
#include <door.h>
int door_bind (int did);
int door_unbind();

DESCRIPTION  
door_bind() associates the current thread with a door server pool. A door server pool is a private pool of server threads that is available to serve door invocations associated with the door did.

door_unbind() breaks the association of door_bind() by removing any private door pool binding that is associated with the current thread.

Normally, door server threads are placed in a global pool of available threads that invocations on any door can use to dispatch a door invocation. A door that has been created with DOOR_PRIVATE only uses server threads that have been associated with the door by door_bind(). Therefore, it is necessary to bind at least one server thread to doors created with DOOR_PRIVATE.

The server thread create routine, door_server_create(), is initially called by the system during a door_create() operation. See door_server_create(3X) and door_create(3X).

The current thread is added to the private pool of server threads associated with a door during the next door_return() (that has been issued by the current thread after an associated door_bind()). See door_return(3X). A server thread performing a door_bind() on a door that is already bound to a different door performs an implicit door_unbind() of the previous door.

RETURN VALUES
Upon successful completion, a 0 is returned. Upon failure, a -1 is returned and errno is set to indicate the error.

ERRORS
The door_bind() and door_unbind() functions fail if one or more of the following are true:

EBADF  
did is not a valid door

EBADF  
door_unbind() with a server thread that is currently not bound

EINVAL  
did was not created with the DOOR_PRIVATE attribute

EXAMPLES
The following example shows the use of door_bind() to create private server pools for two doors, d1 and d2. Function my_create() is called when a new server thread is needed; it creates a thread running function, my_server_create(), which binds itself to one of the two doors.
#include <door.h>
#include <thread.h>
#include <pthread.h>
thread_key_t door_key;
int d1 = -1;
int d2 = -1;
extern foo(); extern bar();

static void *
my_server_create(void *arg)
{
    while (d2 == -1)
        yield(); /* Wait for door descriptor to initialize */
    if (arg == (void *)&foo){
        /* bind thread with pool associated with d1 */
        thr_setspecific(door_key, (void *)&foo); 
        if (door_bind(d1) < 0) {
            perror("door_bind"); exit (-1);
        }
    } else if (arg == (void *)&bar) {
        /* bind thread with pool associated with d2 */
        thr_setspecific(door_key, (void *)&bar); 
        if (door_bind(d2) < 0) {
            perror("door_bind"); exit (-1);
        }
    }

    pthread_setcancelstate(POSIX_CANCEL_DISABLE, NULL);
    door_return(NULL, 0, NULL, 0); /* Wait for door invocation */
}

static void
my_create(door_info_t *dip)
{
    /* Pass the door identity information to create function */
    thr_create(NULL, 0, my_server_create, (void *)&dip->di_proc,
                THR_BOUND | THR_DETACHED, NULL);
}

main()
{
    (void)door_server_create(my_create);
    d1 = door_create(foo, NULL, DOOR_PRIVATE); /* Private pool */
    d2 = door_create(bar, NULL, DOOR_PRIVATE); /* Private pool */
    while (1)
        pause();
}
ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>all</td>
</tr>
<tr>
<td>Availability</td>
<td>SUNWcsu</td>
</tr>
<tr>
<td>Stability</td>
<td>Evolving</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
door_create(3X), door_return(3X), door_server_create(3X), attributes(5)
NAME       door_call – invoke the function associated with a door descriptor

SYNOPSIS   #include <door.h>

typedef struct {
  char      *data_ptr;    /* Argument/result buf ptr*/
  size_t    data_size;    /* Argument/result buf size */
  door_desc_t *desc_ptr;  /* Argument/result descriptors */
  size_t    desc_num;     /* Argument/result num desc */
  char      *rbuf;        /* Result buffer */
  size_t    rsize;        /* Result buffer size */
} door_arg_t;

int door_call(int d, door_arg_t *params);

DESCRIPTION The door_call() function invokes the function associated with the door descriptor d, and
passes the arguments (if any) specified in params. All of the params members are treated
as in/out parameters during a door invocation and may be updated upon returning from
a door call. Passing NULL for params indicates there are no arguments to be passed and
no results expected.

Arguments are specified using the data_ptr and desc_ptr members of params. The size of
the argument data in bytes is passed in data_size and the number of argument descrip-
tors is passed in desc_num.

Results from the door invocation are placed in the buffer, rbuf. See door_return(3X).
The data_ptr and desc_ptr members of params are updated to reflect the location of the
results within the rbuf buffer. The size of the data results and number of descriptors
returned are updated in the data_size and desc_num members. It is acceptable to use the
same buffer for input argument data and results, so door_call() may be called with
data_ptr and desc_ptr pointing to the buffer rbuf.

If the results of a door invocation exceed the size of the buffer specified by rsize, the sys-

tem automatically allocates a new buffer in the caller’s address space and updates the
rbuf and rsize members to reflect this location. In this case, the caller is responsible for
reclaiming this area using munmap(rbuf, rsize) when the buffer is no longer required.
See munmap(2).

Descriptors passed in a door_desc_t structure are identified by the d_attributes member.
The client marks the d_attributes member with the type of object being passed by logi-
cally OR-ing the value of object type. Currently, the only object type that may be passed
or returned is a file descriptor, denoted by the DOOR_DESCRIPTOR attribute.
The door_desc_t structure includes the following members:

```c
typedef struct {
    door_attr_t d_attributes; /* Describes the parameter */
    union {
        struct {
            int d_descriptor; /* Descriptor */
            door_id_t d_id; /* Unique door id */
        } d_desc;
        } d_data;
    } door_desc_t;
```

When file descriptors are passed or returned, a new descriptor is created in the target
address space and the d_descriptor member in the target argument is updated to reflect
the new descriptor. In addition, the system passes a system-wide unique number associ-
ated with each door in the door_id member and marks the d_attributes member with
other attributes associated with a door including the following:

- **DOOR_LOCAL** The door received was created by this process using `door_create()`. 
  (see `door_create(3X)`).
- **DOOR_PRIVATE** The door received has a private pool of server threads associated
  with the door.
- **DOOR_UNREF** The door received is expecting an unreferenced notification.
- **DOOR_REVOKED** The door received has been revoked by the server.

The `door_call()` function is not a restartable system call. If returns EINTR if a signal was
caught and handled by this thread. If the door invocation is not idempotent the caller
should mask any signals that may be generated during a `door_call()` operation. If the
client aborts in the middle of a `door_call()`, the server thread is notified using the POSIX
(see `standards(5)`) thread cancellation mechanism. See `cancellation(3T)`.

The descriptor returned from `door_create()` is marked as close on exec (FD_CLOEXEC).
Information about a door is available for all clients of a door using `door_info()`. Programs
concerned with security should not place secure information in door data that is
accessible by `door_info()`. In particular, secure data should not be stored in the data item
cookie. See `door_info(3X)`.

### RETURN VALUES

Upon successful completion, 0 is returned. Upon failure, −1 is returned and `errno` is set
to indicate the error.

### ERRORS

The `door_call()` function fails if:

- **EBADF** Invalid door descriptor was passed
- **EINVAL** Bad arguments were passed
- **EFAULT** Argument pointers pointed outside the allocated address space

modified 21 Mar 1997 SunOS 5.6
3X-423
E2BIG Arguments were too big for server thread stack
EOVERFLOW System could not create overflow area in caller for results.
EAGAIN Server was out of available resources
EINTR Signal was caught in the client during the invocation
EMFILE The client or server has too many open descriptors

**ATTRIBUTES**

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>all</td>
</tr>
<tr>
<td>Availability</td>
<td>SUNWcsu</td>
</tr>
<tr>
<td>Stability</td>
<td>Evolving</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

**SEE ALSO**

munmap(2), cancellation(3T), door_create(3X), door_info(3X), door_return(3X), attributes(5), standards(5)
NAME       door_create – create a door descriptor

SYNOPSIS  #include <door.h>
           
           int door_create (void (*server_procedure) (void *cookie, char *argp,
                  size_t arg_size, door_desc_t *dp, size_t n_desc),
                  void *cookie, u_int attributes);

DESCRIPTION The door_create() function creates a door descriptor that describes the procedure
specified by the function server_procedure. The data item, cookie, is associated with the
door descriptor, and is passed as an argument to the invoked function server_procedure
during door_call(3X) invocations. Other arguments passed to server_procedure from an
associated door_call() are placed on the stack and include argp and dp. argp points to
arg_size bytes of data and dp points to n_desc door_desc_t structures. The attributes flag
specifies attributes associated with the newly created door. Valid values for attributes are
constructed by OR-ing in one or more of the following values:

DOOR_UNREF  Delivers a special invocation on the door when the number of
descriptors that refer to this door drops to one. In order to trigger
this condition, more than one descriptor must have referred to this
door at some time. DOOR_UNREF_DATA designates an unreferenced
invocation, as the argp argument passed to server_procedure. In the
case of an unreferenced invocation, the values for arg_size, dp and
n_desc are 0. Only one unreferenced invocation is delivered on behalf
of a door.

DOOR_PRIVATE  Maintains a separate pool of server threads on behalf of the door.
Server threads are associated with a door’s private server pool using
door_bind(3X).

The descriptor returned from door_create() will be marked as close on exec
(FD_CLOEXEC). Information about a door is available for all clients of a door using
door_info(3X). Programs concerned with security should not place secure information in
door data that is accessible by door_info(). In particular, secure data should not be
stored in the data item cookie.

RETURN VALUES  Upon successful completion, door_create() returns a non-negative value. Upon failure,
door_create returns -1 and sets errno to indicate the error.

ERRORS  The door_create() function fails if one or more of the following are true:

EINVAL  Invalid attributes are passed.
EMFILE  The process has too many open descriptors.
ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>all</td>
</tr>
<tr>
<td>Availability</td>
<td>SUNWcsu</td>
</tr>
<tr>
<td>Stability</td>
<td>Evolving</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO door_bind(3X), door_call(3X), door_info(3X), door_revoke(3X), door_server_create(3X), attributes(5)
NAME
door_cred – return credential information associated with the client

SYNOPSIS
#include <door.h>
int door_cred (door_cred_t *info);

DESCRIPTION
The door_cred() function returns credential information associated with the client (if any) of the current door invocation.

The contents of the info argument include the following fields:

uid_t dc_euid; /* Effective uid of client */
gid_t dc_egid; /* Effective gid of client */
uid_t dc_ruid; /* Real uid of client */
gid_t dc_rgid; /* Real gid of client */
pid_t dc_pid; /* pid of client */

The credential information associated with the client refers to the information from the immediate caller; not necessarily from the first thread in a chain of door calls.

RETURN VALUES
Upon successful completion, door_cred() returns 0. Upon failure, door_cred() returns -1 and sets errno to indicate the error.

ERRORS
The door_cred() function fails if one or more of the following are true:

EFAULT The address of the info argument is invalid.
EINVAL There is no associated door client.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>all</td>
</tr>
<tr>
<td>Availability</td>
<td>SUNWcsu</td>
</tr>
<tr>
<td>Stability</td>
<td>Evolving</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
door_call(3X), door_create(3X), attributes(5)

modified 26 Nov 1996
SunOS 5.6
3X-427
NAME

door_info – return information associated with a door descriptor

SYNOPSIS

#include <door.h>

int door_info (int d, struct door_info *info);

DESCRIPTION

The door_info() function returns information associated with a door descriptor.
door_info() obtains information about the door descriptor d and places the information
that is relevant to the door in the structure pointed to by the argument info.

The contents of the info argument contains the following fields:

pid_t di_target;    /* door server pid */
door_ptr_t di_proc; /* server function */
door_ptr_t di_data; /* data cookie for invocation */
door_attr_t di_attributes; /* door attributes */
door_id_t di_uniquifier; /* unique id among all doors */

The values for di_attributes may be composed of the following:

DOOR_LOCAL The door descriptor refers to a service procedure in this process.

DOOR_UNREF The door has requested notification when all but the last reference remain.

DOOR_REVOKED The door descriptor refers to a door that has been revoked.

DOOR_PRIVATE The door has a separate pool of server threads associated with it.

The di_proc and di_data fields are returned as door_ptr_t objects rather than void *
pointers in order to allow clients and servers to interoperate in environments where the
pointer sizes may vary in size (for example, 32-bit clients and 64-bit servers). Each door
has a system-wide unique number associated with it that is set when the door is created
by door_create(). This number is returned in di_uniquifier.

RETURN VALUES

Upon successful completion, 0 is returned. Upon failure, -1 is returned and errno is set to
indicate the error.

ERRORS

The door_info() function fails if one or more of the following are true:

EFAULT   The address of argument info is an invalid address.

EBADF     d is not a door descriptor.

SEE ALSO

door_bind(3X), door_server_create(3X)
### NAME
door_return – return from a door invocation

### SYNOPSIS
```c
#include <door.h>

int door_return(void *data_ptr, size_t data_size,
                 door_desc_t *desc_ptr, size_t num_desc);
```

### DESCRIPTION
The `door_return()` function returns from a door invocation. It returns control to the thread that issued the associated `door_call()` and blocks waiting for the next door invocation. See `door_call(3X)`. Results, if any, from the door invocation are passed back to the client in the buffers pointed to by `data_ptr` and `desc_ptr`. If there is not a client associated with the `door_return()`, the calling thread discards the results and blocks waiting for the next door invocation.

### RETURN VALUES
Upon successful completion, `door_return()` does not return to the calling process. Upon failure, `door_return()` returns `-1` to the calling process and sets `errno` to indicate the error.

### ERRORS
The `door_return()` function fails and returns to the calling process, if one or more of the following are true:
- **EINVAL** Invalid `door_return()` arguments were passed.
- **EFAULT** The address of `data_ptr` or `desc_ptr` is invalid.

### SEE ALSO
`door_call(3X)`

---

**modified 26 Nov 1996**

SunOS 5.6

3X-429
NAME

door_revoke – revoke access to a door descriptor

SYNOPSIS

#include <door.h>
int door_revoke(int d);

DESCRIPTION

The door_revoke() function revokes access to a door descriptor. Door descriptors are created with door_create(3X). door_revoke() performs an implicit call to close(2), marking the door descriptor d as invalid.

A door descriptor can only be revoked by the process that created it. Door invocations that are in progress during a door_revoke() invocation are allowed to complete normally.

RETURN VALUES

Upon successful completion, door_revoke() returns 0. Upon failure, door_revoke() returns -1 and sets errno to indicate the error.

ERRORS

The door_revoke() function fails if one or more of the following are true:

EBADF An invalid door descriptor was passed.
EPERM The door descriptor was not created by this process (with door_create(3X)).

ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>all</td>
</tr>
<tr>
<td>Availability</td>
<td>SUNWcsu</td>
</tr>
<tr>
<td>Stability</td>
<td>Evolving</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO

close(2), door_create(3X), attributes(5)
NAME
door_server_create – specify an alternative door server thread creation function

SYNOPSIS
#include <door.h>
void (*) () door_server_create(void (*create_proc) (door_info_t *));

DESCRIPTION
Normally, the doors library creates new door server threads in response to incoming concurrent door invocations automatically. There is no pre-defined upper limit on the number of server threads that the system creates in response to incoming invocations (1 server thread for each active door invocation). These threads are created with the default thread stack size and POSIX (see standards(5)) threads cancellation disabled. The created threads also have the THR_BOUND | THR_DETACHED attributes for Solaris threads and the PTHREAD_SCOPE_SYSTEM | PTHREAD_CREATE_DETACHED attributes for POSIX threads. The signal disposition, and scheduling class of the newly created thread are inherited from the calling thread (initially from the thread calling door_create(), and subsequently from the current active door server thread).

The door_server_create() function allows control over the creation of server threads needed for door invocations. The procedure create_proc is called every time the available server thread pool is depleted. In the case of private server pools associated with a door (see the DOOR_PRIVATE attribute in door_create()), information on which pool is depleted is passed to the create function in the form of a door_info_t structure. The di_proc and di_data members of the door_info_t structure may be used as a door identifier associated with the depleted pool. The create_proc procedure may limit the number of server threads created and may also create server threads with appropriate attributes (stack size, thread-specific data, POSIX thread cancellation, signal mask, scheduling attributes, and so forth) for use with door invocations.

The specified server creation function should create user level threads using thr_create() with the THR_BOUND flag, or in the case of POSIX threads, pthread_create() with the PTHREAD_SCOPE_SYSTEM attribute. The server threads make themselves available for incoming door invocations on this process by issuing a door_return(NULL, 0, NULL, 0). In this case, the door_return() arguments are ignored. See door_return(3X) and thr_create(3T).

The server threads created by default are enabled for POSIX thread cancellations which may lead to unexpected thread terminations while holding resources (such as locks) if the client aborts the associated door_call(). See door_call(3X). Unless the server code is truly interested in notifications of client aborts during a door invocation and is prepared to handle such notifications using cancellation handlers, POSIX thread cancellation should be disabled for server threads using pthread_setcancelstate (PTHREAD_CANCEL_DISABLE, NULL).

The create_proc procedure need not create any additional server threads if there is at least one server thread currently active in the process (perhaps handling another door invocation) or it may create as many as seem fit each time it is called. If there are no available server threads during an incoming door invocation, the associated door_call() blocks until a server thread becomes available. The create_proc procedure must be MT-Safe.
RETURN VALUES
Upon successful completion, `door_server_create()` returns a pointer to the previous server creation function. This function has no failure mode (it cannot fail).

EXAMPLES
The following example creates door server threads with cancellation disabled and a 8k stack instead of the default stack size:

```c
#include <door.h>
#include <pthread.h>
#include <thread.h>

void *
my_thread(void *arg)
{
    pthread_setcancelstate(PTHREAD_CANCEL_DISABLE, NULL);
    door_return(NULL, 0, NULL, 0);
}

void *
my_create(door_info_t *dip)
{
    thr_create(NULL, 8192, my_thread, NULL,
                THR_BOUND | THR_DETACHED, NULL);
}

main()
{
    (void)door_server_create(my_create);
    ...
}
```

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>all</td>
</tr>
<tr>
<td>Availability</td>
<td>SUNWcsu</td>
</tr>
<tr>
<td>Stability</td>
<td>Evolving</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
`cancellation(3T)`, `door_bind(3X)`, `door_call(3X)`, `door_create(3X)`, `door_return(3X)`, `pthread_create(3T)`, `pthread_setcancelstate(3T)`, `thr_create(3T)`, `attributes(5)`, `standards(5)`
### NAME

doupdate, refresh, wnoutrefresh, wrefresh – refresh windows and lines

### SYNOPSIS

```c
#include <curses.h>
int doupdate(void);
int refresh(void);
int wnoutrefresh(WINDOW *win);
int wrefresh(WINDOW *win);
```

### ARGUMENTS

*win*  
Is a pointer to the window in which to refresh.

### DESCRIPTION

The `refresh()` and `wrefresh()` functions copy `stdscr` and `win`, respectively, to the terminal screen. These functions call the `wnoutrefresh()` function to copy the specified window to `curscr` and the `doupdate()` function to do the actual update. The physical cursor is mapped to the same position as the logical cursor of the last window to update `curscr` unless `leaveok(3XC)` is enabled (in which case, the cursor is placed in a position that X/Open Curses finds convenient).

When outputting several windows at once, it is often more efficient to call the `wnoutrefresh()` and `doupdate()` functions directly. A call to `wnoutrefresh()` for each window, followed by only one call to `doupdate()` to update the screen, results in one burst of output, fewer characters sent, and less CPU time used.

If the `win` parameter to `wrefresh()` is the global variable `curscr`, the screen is immediately cleared and repainted from scratch.

For details on how the `wnoutrefresh()` function handles overlapping windows with broad glyphs, see the Overlapping Windows section of the `curses(3XC)` man page.

### RETURN VALUES

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

### ERRORS

None.

### SEE ALSO

clearok(3XC), curses(3XC), prefresh(3XC), redrawwin(3XC)
NAME
drand48, erand48, lrand48, nrand48, mrand48, jrand48, srand48, seed48, lcong48 – generate uniformly distributed pseudo-random numbers

SYNOPSIS
#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]);

DESCRIPTION
This family of functions generates pseudo-random numbers using the well-known linear congruential algorithm and 48-bit integer arithmetic.

Functions drand48() and erand48() return non-negative double-precision floating-point values uniformly distributed over the interval [0.0, 1.0).

Functions lrand48() and nrand48() return non-negative long integers uniformly distributed over the interval [0, 2^{31}).

Functions mrand48() and jrand48() return signed long integers uniformly distributed over the interval [−2^{31}, 2^{31}).

Functions srand48(), seed48(), and lcong48() are initialization entry points, one of which should be invoked before either drand48(), lrand48(), or mrand48() is called. (Although it is not recommended practice, constant default initializer values will be supplied automatically if drand48(), lrand48(), or mrand48() is called without a prior call to an initialization entry point.) Functions erand48(), nrand48(), and jrand48() do not require an initialization entry point to be called first.

All the routines work by generating a sequence of 48-bit integer values, \(X_n\), according to the linear congruential formula
\[
X_{n+1} = (aX_n + c) \mod m \\
\text{for } n \geq 0.
\]
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
\[
a = 5DEECE66D_{16} = 273673163155_8
\]
\[
c = B_{16} = 13_8.
\]
The value returned by any of the functions `drand48()`, `erand48()`, `lrand48()`, `nrand48()`, `mrand48()`, 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 `mrand48()` store the last 48-bit \( X_i \) generated in an internal buffer. \( X_i \) must be initialized prior to being invoked. The functions `erand48()`, `nrand48()`, and `jrand48()` require the calling program to provide storage for the successive \( X_i \) values in the array specified as an argument when the functions are invoked. These routines do not have to be initialized; the calling program must place the desired initial value of \( X_i \) into the array and pass it as an argument. By using different arguments, functions `erand48()`, `nrand48()`, and `jrand48()` allow separate modules of a large program to generate several independent streams of pseudo-random numbers, that is, the sequence of numbers in each stream will not depend upon how many times the routines have been called to generate numbers for the other streams.

The initializer function `srand48()` sets the high-order 32 bits of \( X_i \) to the 32 bits contained in its argument. The low-order 16 bits of \( X_i \) are set to the arbitrary value 330E16.

The initializer function `seed48()` sets the value of \( X_i \) to the 48-bit value specified in the argument array. In addition, the previous value of \( X_i \) is copied into a 48-bit internal buffer, used only by `seed48()`, and a pointer to this buffer is the value returned by `seed48()`. This returned pointer, which can just be ignored if not needed, is useful if a program is to be restarted from a given point at some future time — use the pointer to get at and store the last \( X_i \) value, and then use this value to reinitialize using `seed48()` when the program is restarted.

The initialization function `lcong48()` allows the user to specify the initial \( X_i \), the multiplier value \( a \), and the addend value \( c \). Argument array elements `param[0-2]` specify \( X_i \), `param[3-5]` specify the multiplier \( a \), and `param[6]` specifies the 16-bit addend \( c \). After `lcong48()` has been called, a subsequent call to either `srand48()` or `seed48()` will restore the “standard” multiplier and addend values, \( a \) and \( c \), specified above.

**ATTRIBUTES**

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

**SEE ALSO**

`rand(3C)`, `attributes(5)`

modified 22 Jan 1993

SunOS 5.6

3C-435
NAME
dup2 – duplicate an open file descriptor

SYNOPSIS
#include <unistd.h>
int dup2(int fd1des, int fd2des);

DESCRIPTION
The dup2() function causes the file descriptor fd2des to refer to the same file as fd1des. The fd1des argument is a file descriptor referring to an open file, and fd2des 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 fd2des already refers to an open file, not fd1des, it is closed first. If fd2des refers to fd1des or if fd1des is not a valid open file descriptor, fd2des will not be closed first.

The dup2() function is equivalent to fcntl(fd1des, F_DUP2FD, fd2des).

RETURN VALUES
Upon successful completion a non-negative integer representing the file descriptor is returned. Otherwise, −1 is returned and errno is set to indicate the error.

ERRORS
The dup2() function will fail if:
EBADF The fd1des argument is not a valid open file descriptor.
EBADF The fd2des argument is negative or is not less than the current resource limit returned by getrlimit(RLIMIT_NOFILE, …).
EINTR A signal was caught during the dup2() call.
EMFILE The process has too many open files. See fcntl(2).

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
close(2), creat(2), exec(2), fcntl(2), getrlimit(2), open(2), pipe(2), lockf(3C), attributes(5)
NAME
dupwin – duplicate a window

SYNOPSIS
#include <curses.h>
WINDOW *dupwin(WINDOW, *win);

ARGUMENTS
win Is a pointer to the window that is to be duplicated.

DESCRIPTION
The dupwin() function creates a duplicate of window win. A pointer to the new window structure is returned.

RETURN VALUES
On success, this function returns a pointer to new window structure; otherwise, it returns a null pointer.

ERRORS
None.

SEE ALSO
delwin(3XC), derwin(3XC)
NAME  echo, noecho – enable/disable terminal echo

SYNOPSIS  #include <curses.h>
int echo(void);
int noecho(void);

DESCRIPTION  The `echo()` and `noecho()` functions enable and disable terminal echo, respectively. When enabled, characters received by `getch(3XC)` are echoed back to the terminal. When disabled, characters are transferred to the program without echoing them to the terminal display. The program may instead echo the characters to an area of the screen controlled by the program or may not echo the characters at all. Terminal echo is enabled, by default.

Subsequent calls to `echo()` or `noecho()` do not flush type-ahead.

The tty driver echo is disabled by `initscr(3XC)` and `newterm(3XC)`. All echoing is controlled by X/Open Curses.

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

ERRORS  None.

SEE ALSO  `getch(3XC)`, `getstr(3XC)`, `initscr(3XC)`, `scanw(3XC)`
NAME  echochar, wechochar – add a single-byte character and refresh window

SYNOPSIS  
```
#include <curses.h>
int echochar(const chtype ch);
int wechochar(WINDOW *win, const chtype ch);
```

ARGUMENTS  
* `ch` Is a pointer to the character to be written to the window.
* `win` Is a pointer to the window in which the character is to be added.

DESCRIPTION  
The `echochar()` function produces the same effect as calling `addch(3XC)` and then `refresh(3XC)`. The `wechochar()` function produces the same effect as calling `waddch(3XC)` and then `wrefresh(3XC)`. 

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

ERRORS  
None.

SEE ALSO  
`addch(3XC)`, `doupdate(3XC)`, `echo_wchar(3XC)`
NAME  echo_wchar, wecho_wchar – add a complex character and refresh window

SYNOPSIS  
#include <curses.h>

int echo_wchar(const cchar_t *wch);
int wecho_wchar(WINDOW *win, const cchar_t *wch);

ARGUMENTS  
wch  Is a pointer to the complex character to be written to the window.
win  Is a pointer to the window in which the character is to be added.

DESCRIPTION  
The echo_wchar() function produces the same effect as calling add_wch(3XC) and then refresh(3XC). The wecho_wchar() function produces the same effect as calling wadd_wch(3XC) and then wrefresh(3XC).

RETURN VALUES  On success, these functions return OK. Otherwise, they return ERR.

ERRORS  None.

SEE ALSO  add_wch(3XC), doupdate(3XC), echochar(3XC)
NAME
econvert, fconvert, gconvert, seconvert, sfconvert, sgconvert, qeconvert, qfconvert, qgconvert, ecvt, fcvt, gcvt – output conversion

SYNOPSIS
#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 *decpt, int *sign, 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 *decpt, int *sign, 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 *decpt, int *sign, 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);

DESCRIPTION
The econvert() function converts the value to a null-terminated string of ndigit ASCII digits in buf and returns a pointer to buf. buf should contain at least ndigit+1 characters. The position of the decimal point relative to the beginning of the string is stored indirectly through decpt. Thus buf == "314" and *decpt == 1 corresponds to the numerical value 3.14, while buf == "314" and *decpt == -1 corresponds to the numerical value .0314. If the sign of the result is negative, the word pointed to by sign is nonzero; otherwise it is zero. The least significant digit is rounded.

The fconvert() function works much like econvert(), except that the correct digit has been rounded as if for sprintf(%w.nf) output with n=ndigit digits to the right of the decimal point. ndigit can be negative to indicate rounding to the left of the decimal point. The return value is a pointer to buf. buf should contain at least 310+max(0,ndigit) characters to accommodate any double-precision value.

The gconvert() function converts the value to a null-terminated ASCII string in buf and returns a pointer to buf. It produces ndigit 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 buf is ready for printing, with sign and exponent. The result corresponds to that obtained by

(void) sprintf(buf,"%w.ng",value);

If trailing = 0, trailing zeros and a trailing point are suppressed, as in sprintf(%g). If trailing != 0, trailing zeros and a trailing point are retained, as in sprintf(%#g).

The seconvert(), sfconvert(), and sgconvert() functions are single-precision versions of these functions, and are more efficient than the corresponding double-precision versions. A pointer rather than the value itself is passed to avoid C’s usual conversion of single-
precision arguments to double.

The \texttt{qfconvert()}, \texttt{qgconvert()} functions are quadruple-precision versions of these functions. The \texttt{qfconvert()} function can overflow the \textit{decimal_record} field \texttt{ds} if \texttt{value} is too large. In that case, \texttt{buf[0]} is set to zero.

The \texttt{ecvt()} and \texttt{fcvt()} functions are versions of \texttt{econvert()} and \texttt{fconvert()} that create a string in a static data area, overwritten by each call, and return values that point to that static data. These functions are therefore not reentrant.

The \texttt{gcvt()} function is an version of \texttt{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 \texttt{ndigit} $\geq 8$.

\textbf{ATTRIBUTES}

See \texttt{attributes}(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

\textbf{SEE ALSO} \texttt{sprintf}(3S), \texttt{attributes}(5)
NAME
ecvt, fcvt, gcvt – convert floating-point number to string

SYNOPSIS
#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);

DESCRIPTION
The ecvt(), fcvt() and gcvt() functions convert floating-point numbers to null-terminated strings.

ecvt() Converts value to a null-terminated string of ndigit digits (where ndigit is reduced to an unspecified limit determined by the precision of a double) and returns a pointer to the string. The high-order digit is non-zero, unless the value is 0. The low-order digit is rounded. The position of the radix character relative to the beginning of the string is stored in the integer pointed to by decpt (negative means to the left of the returned digits). The radix character is not included in the returned string. If the sign of the result is negative, the integer pointed to by sign is non-zero, otherwise it is 0.

If the converted value is out of range or is not representable, the contents of the returned string are unspecified.

fcvt() Identical to ecvt() except that ndigit specifies the number of digits desired after the radix point. The total number of digits in the result string is restricted to an unspecified limit as determined by the precision of a double.

gcvt() Converts value to a null-terminated string (similar to that of the %g format of printf(3S)) in the array pointed to by buf and returns buf. It produces ndigit significant digits (limited to an unspecified value determined by the precision of a double) in %f if possible, or %e (scientific notation) otherwise. A minus sign is included in the returned string if value is less than 0. A radix character is included in the returned string if value is not a whole number. Trailing zeros are suppressed where value is not a whole number. The radix character is determined by the current locale. If setlocale(3C) has not been called successfully, the default locale, POSIX, is used. The default locale specifies a period (.) as the radix character. The LC_NUMERIC category determines the value of the radix character within the current locale.

RETURN VALUES
The ecvt() and fcvt() functions return a pointer to a null-terminated string of digits. The gcvt() function returns buf.

ERRORS
No errors are defined.

USAGE
The return values from ecvt() and fcvt() may point to static data which may be overwritten by subsequent calls to these functions.

modified 29 Dec 1996
SunOS 5.6
3C-443
For portability to implementations conforming to earlier versions of this document, `sprintf(3S)` is preferred over this function.

**ATTRIBUTES**

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

**SEE ALSO**

`printf(3S), setlocale(3C), sprintf(3S), attributes(5)`
NAME    elf – object file access library

SYNOPSIS  cc [ flag . . . ] file . . . -l elf [ library . . . ]
            #include <libelf.h>

DESCRIPTION  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.

  Programs communicate with many of the higher-level routines using an ELF descriptor. 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 section descriptors 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, data descriptors are available through section descriptors, allowing the program to manipulate the information associated with a section. A data descriptor “belongs” to a section descriptor.

  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.

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 class, or capacity, of the file. The 32-bit class supports architectures in which a 32-bit object can represent addresses, file sizes, and so on, as in the following:

<table>
<thead>
<tr>
<th>Name</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elf32_Addr</td>
<td>Unsigned address</td>
</tr>
<tr>
<td>Elf32_Half</td>
<td>Unsigned medium integer</td>
</tr>
<tr>
<td>Elf32_Off</td>
<td>Unsigned file offset</td>
</tr>
<tr>
<td>Elf32_Sword</td>
<td>Signed large integer</td>
</tr>
<tr>
<td>Elf32_Word</td>
<td>Unsigned large integer</td>
</tr>
<tr>
<td>unsigned char</td>
<td>Unsigned small integer</td>
</tr>
</tbody>
</table>

  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
Data Representation

Conceptually, two parallel sets of objects support cross compilation environments. One set corresponds to file contents, while the other set corresponds to the native memory image of the program manipulating the file. Type definitions supplied by the headers work on the native machine, which may have different data encodings (size, byte order, and so on) than the target machine. Although native memory objects should be at least as big as the file objects (to avoid information loss), they may be bigger if that is more natural for the host machine.

Translation facilities exist to convert between file and memory representations. Some library routines convert data automatically, while others leave conversion as the program’s responsibility. Either way, programs that create object files must write file-typed objects to those files; programs that read object files must take a similar view. See `elf32_xlatetof(3E)` and `elf32_fsize(3E)` for more information.

Programs may translate data explicitly, taking full control over the object file layout and semantics. If the program prefers not to have and exercise complete control, the library provides a higher-level interface that hides many object file details. `elf_begin()` and related functions let a program deal with the native memory types, converting between memory objects and their file equivalents automatically when reading or writing an object file.

ELF Versions

Object file versions allow ELF to adapt to new requirements. Three independent versions can be important to a program. First, an application program knows about a particular version by virtue of being compiled with certain headers. Second, the access library similarly is compiled with header files that control what versions it understands. Third, an ELF object file holds a value identifying its version, determined by the ELF version known by the file’s creator. Ideally, all three versions would be the same, but they may differ.

If a program’s version is newer than the access library, the program might use information unknown to the library. Translation routines might not work properly, leading to undefined behavior. This condition merits installing a new library.

The library’s version might be newer than the program’s and the file’s. The library understands old versions, thus avoiding compatibility problems in this case.

Finally, a file’s version might be newer than either the program or the library understands. The program might or might not be able to process the file properly, depending on whether the file has extra information and whether that information can be safely ignored. Again, the safe alternative is to install a new library that understands the file’s version.

To accommodate these differences, a program must use `elf_version(3E)` to pass its version to the library, thus establishing the working version for the process. Using this, the library accepts data from and presents data to the program in the proper representations. When the library reads object files, it uses each file’s version to interpret the data. When writing files or converting memory types to the file equivalents, the library uses the
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_ctl(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.

Names associated with the library take several forms.

- **elf_name** These class-independent names perform some service, *name*, 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 *Type*.
- **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 *KIND* 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`.

The basic interpretation of an ELF file consists of:

- opening an ELF object file
The following example opens the file, obtains the ELF descriptor, and prints out the names of each section in the file.

```c
#include <fcntl.h>
#include <stdio.h>
#include <libelf.h>
#include <stdlib.h>
#include <string.h>

static void failure(void);

void
main(int argc, char ** argv)
{
    Elf32_Shdr * shdr;
    Elf32_Ehdr * ehdr;
    Elf * elf;
    Elf_Scn * scn;
    Elf_Data * data;
    int fd;
    unsigned int cnt;

    /* Open the input file */
    if ((fd = open(argv[1], O_RDONLY)) == -1)
        exit(1);

    /* Obtain the ELF descriptor */
    (void) elf_version(EV_CURRENT);
    if ((elf = elf_begin(fd, ELF_C_READ, NULL)) == NULL)
        failure();

    /* Obtain the .shstrtab data buffer */
    if (((ehdr = elf32_getehdr(elf)) == NULL) ||
        ((scn = elf_getscn(elf, ehdr->e_shstrndx)) == NULL) ||
        ((data = elf_getdata(scn, NULL)) == NULL))
        failure();

    /* Traverse input filename, printing each section */
    for (cnt = 1, scn = NULL; scn = elf_nextscn(elf, scn); cnt++) {
        if ((shdr = elf32_getshdr(scn)) == NULL)
            failure();

        (void) printf("[%d]%s\n", cnt,
            (char *)data->d_buf + shdr->sh_name);
    }
}
```

3E-448 SunOS 5.6 modified 12 Feb 1997
static void
failure()
{
    (void) fprintf(stderr, "%s\n", elf_errmsg(elf_errno()));
    exit(1);
}

Below is sample output from compiling and executing the above code, which prints the
names of the sections using itself as the input file

% cc -o elfprint example.c -lelf
% elfprint elfprint
[1] .interp
[2] .hash
[3] .dynsym
[4] .dynstr
[5] .rela.ex_shared
[6] .rela.bss
[7] .rela.plt
[8] .text
[9] .init
[10] .fini
...

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO elf32_fsize(3E), elf32_getshdr(3E), elf32_xlatetof(3E), elf_begin(3E), elfcntl(3E),
elf_errmsg(3E), elf_fill(3E), elf_getarhdr(3E), elf_getarsym(3E), elf_getbase(3E),
elf_getdata(3E), elf_getident(3E), elf_getscn(3E), elf_hash(3E), elf_kind(3E),
elf_memory(3E), elf_rawfile(3E), elf_strptr(3E), elf_update(3E), elf_version(3E), ar(4),
attributes(5)

ANSI C Programmer’s Guide

SPARC only a.out(4)

NOTES Information in the ELF headers is separated into common parts and processor-specific parts. A program can make a processor’s information available by including the appropriate header: <sys/elf_NAME.h> where NAME matches the processor name as

modified 12 Feb 1997
SunOS 5.6
3E-449
used in the ELF file header.

Name  Processor
M32    AT&T WE 32100
SPARC  SPARC
386    Intel 80386, 80486, Pentium

Other processors will be added to the table as necessary.

To illustrate, a program could use the following code to “see” the processor-specific information for the SPARC based system.

```
#include <libelf.h>
#include <sys/elf_SPARC.h>
```

Without the `<sys/elf_SPARC.h>` definition, only the common ELF information would be visible.

A program could use the following code to “see” the processor-specific information for the Intel 80386:

```
#include <libelf.h>
#include <sys/elf_386.h>
```

Without the `<sys/elf_386.h>` definition, only the common ELF information would be visible.

Although reading the objects is rather straightforward, writing/updating them can corrupt the shared offsets among sections. Upon creation, relationships are established among the sections that must be maintained even if the object’s size is changed.
NAME    elf32_fsize – return the size of an object file type

SYNOPSIS cc [ flag ...] file ... -lelf [ library ...]
#include <libelf.h>
size_t elf32_fsize(Elf_Type type, size_t count, unsigned ver);

DESCRIPTION elf32_fsize() gives the size in bytes of the 32-bit file representation of count data objects
with the given type. The library uses version ver to calculate the size (see elf(3E) and elf_version(3E)).

Constant values are available for the sizes of fundamental types:

<table>
<thead>
<tr>
<th>Elf_Type</th>
<th>File Size</th>
<th>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>ELF32_FSZ_HALF</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>ELT_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>

elf32_fsize() returns 0 if the value of type or ver is unknown. See elf32_xlatetof(3E) for a
list of the type values.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO elf(3E), elf32_xlatetof(3E), elf_version(3E), attributes(5)
NAME  elf32_getehdr, elf32_newehdr – retrieve class-dependent object file header

SYNOPSIS  cc [ flag ... ] file ... -lelf [ library ... ]
#include <libelf.h>
Elf32_Ehdr *elf32_getehdr(Elf *elf);
Elf32_Ehdr *elf32_newehdr(Elf *elf);

DESCRIPTION  For a 32-bit class file, elf32_getehdr() returns a pointer to an ELF header, if one is available for the ELF descriptor elf. 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 elf is null, both functions return a null pointer.

The header includes the following members:

- unsigned char e_ident[ELI_NIDENT];
- Elf32_Half e_type;
- Elf32_Half e_machine;
- Elf32_Word e_version;
- Elf32_Addr e_entry;
- Elf32_Offset e_phoff;
- Elf32_Offset e_shoff;
- Elf32_Word e_flags;
- Elf32_Half e_ehsize;
- Elf32_Half e_phentsize;
- Elf32_Half e_phnum;
- Elf32_Half e_shsize;
- Elf32_Half e_shnum;
- Elf32_Half e_shstrndx;

elf32_newehdr() automatically sets the ELF_F_DIRTY bit (see elf_flagdata(3E)). A program may use elf_getident() to inspect the identification bytes from a file.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  elf(3E), elf_begin(3E), elf_flagdata(3E), elf_getident(3E), attributes(5)
NAME
elf32_getphdr, elf32_newphdr – retrieve class-dependent program header table

SYNOPSIS
cc [ flag ...] file ... -lelf [ library ...]
#include <libelf.h>
Elf32_Phdr *elf32_getphdr(Elf *elf);
Elf32_Phdr *elf32_newphdr(Elf *elf, size_t count);

DESCRIPTION
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 elf.

elf32_newphdr() allocates a new table with count entries, regardless of whether one existed previously, and sets the ELF_F_DIRTY bit for the table (see elf_flagdata(3E)). Specifying a zero count 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.

If no program header table exists, the file is not a 32-bit class file, an error occurs, or elf is NULL, both functions return a null pointer. Additionally, elf32_newphdr() returns a null pointer if count is 0.

The table is an array of Elf32_Phdr structures, each of which includes the following members:

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;

The ELF header’s e_phnum member tells how many entries the program header table has (see elf32_getehdr(3E)). A program may inspect this value to determine the size of an existing table; elf32_newphdr() automatically sets the member’s value to count. If the program is building a new file, it is responsible for creating the file’s ELF header before creating the program header table.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
elf(3E), elf32_getehdr(3E), elf_begin(3E), elf_flagdata(3E), attributes(5)

modified 29 Dec 1996
SunOS 5.6
3E-453
NAME    elf32_getshdr – retrieve class-dependent section header

SYNOPSIS   cc [ flag ...] file ... -lelf [ library ...]
#include <libelf.h>
Elf32_Shdr *elf32_getshdr(Elf_Scn *scn);

DESCRIPTION For a 32-bit class file, elf32_getshdr() returns a pointer to a section header for the section descriptor scn. Otherwise, the file is not a 32-bit class file, scn was NULL, or an error occurred; elf32_getshdr() then returns NULL.

The header includes the following members.

    Elf32_Word sh_name;
    Elf32_Word sh_type;
    Elf32_Word sh_−−ags;
    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;

If the program is building a new file, it is responsible for creating the file's ELF header before creating sections.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO       elf(3E), elf_flagdata(3E), elf_getscn(3E), elf_strptr(3E), attributes(5)
NAME       elf32_xlatetof, elf32_xlatetom – class-dependent data translation

SYNOPSIS   cc [ flag ...] file ... -lelf [ library ...]
#include <libelf.h>
Elf_Data *elf32_xlatetof(Elf_Data *dst, const Elf_Data *src, unsigned encode);
Elf_Data *elf32_xlatetom(Elf_Data *dst, const Elf_Data *src, unsigned encode);

DESCRIPTION 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. src is a pointer to the source buffer that holds the original data; dst is a pointer to a destination buffer that will hold the translated copy. encode 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 dst. Otherwise, they return NULL because an error occurred, such as incompatible types, destination buffer overflow, etc.

elf_getdata(3E) describes the Elf_Data descriptor, which the translation routines use as follows:

| 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 the version number of the objects (desired) in the buffer. The source and destination versions are independent. |

Translation routines allow the source and destination buffers to coincide. That is, dst→d_buf may equal src→d_buf. Other cases where the source and destination buffers overlap give undefined behavior.
Elf32_xlatetof (3E)  

C Library Functions

### 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
- **ELT_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.

#### ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

#### SEE ALSO

elf(3E), elf32_fsize(3E), elf_getdata(3E), elf_getident(3E), attributes(5)
NAME    elf_begin, elf_end, elf_memory, elf_next, elf_rand – process ELF object files

SYNOPSIS    cc [ flag ... ] file ... -l elf [ library ... ]
            #include <libelf.h>
            Elf *elf_begin(int fildes, Elf_Cmd cmd, Elf *ref);
            int elf_end(Elf *elf);
            Elf *elf_memory(char *image, size_t sz);
            Elf_Cmd elf_next(Elf *elf);
            size_t elf_rand(Elf *elf, size_t offset);

DESCRIPTION    elf_begin(), elf_end(), elf_memory(), elf_next(), and elf_rand() work together to process Executable and Linking Format (ELF) object files, either individually or as members of archives. After obtaining an ELF descriptor from elf_begin() or elf_memory(), the program may read an existing file, update an existing file, or create a new file. fildes is an open file descriptor that elf_begin() uses for reading or writing. elf 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.

cmd may have the following values:

ELF_C_NULL        When a program sets cmd to this value, elf_begin() returns a null pointer, without opening a new descriptor. ref is ignored for this command. See the examples below for more information.

ELF_C_READ        When a program wishes to examine the contents of an existing file, it should set cmd to this value. Depending on the value of ref, this command examines archive members or entire files. Three cases can occur.

First, if ref 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.

Second, if ref is a non-null descriptor associated with an archive file, elf_begin() lets a program obtain a separate ELF descriptor associated with an individual member. The program should have used elf_next() or elf_rand() to position ref appropriately (except for the initial member, which elf_begin() prepares; see the example below). In this case, fildes should be the same file descriptor used for the parent archive.
Finally, if ref is a non-null ELF descriptor that is not an archive, elf_begin() increments the number of activations for the descriptor and returns ref, without allocating a new descriptor and without changing the descriptor’s read/write permissions. To terminate the descriptor for ref, the program must call elf_end() once for each activation. See the examples below for more information.

**ELF_C_RDWR**

This command duplicates the actions of ELF_C_READ and additionally allows the program to update the file image (see elf_update(3E)). That is, using ELF_C_READ gives a read-only view of the file, while ELF_C_RDWR lets the program read and write the file. ELF_C_RDWR is not valid for archive members. If ref is non-null, it must have been created with the ELF_C_RDWR command.

**ELF_C_WRITE**

If the program wishes to ignore previous file contents, presumably to create a new file, it should set cmd to this value. ref is ignored for this command.

elf_begin() “works” on all files (including files with zero bytes), providing it can allocate memory for its internal structures and read any necessary information from the file. Programs reading object files thus may call elf_kind(3E) or elf32_getehdr(3E) to determine the file type (only object files have an ELF header). If the file is an archive with no more members to process, or an error occurs, elf_begin() returns a null pointer. Otherwise, the return value is a non-null ELF descriptor.

Before the first call to elf_begin(), a program must call elf_version() to coordinate versions.

elf_end() is used to terminate an ELF descriptor, elf, and to deallocate data associated with the descriptor. Until the program terminates a descriptor, the data remain allocated. A null pointer is allowed as an argument, to simplify error handling. If the program wishes to write data associated with the ELF descriptor to the file, it must use elf_update() before calling elf_end().

Calling elf_end() removes one activation and returns the remaining activation count. The library does not terminate the descriptor until the activation count reaches 0. Consequently, a 0 return value indicates the ELF descriptor is no longer valid.

elf_memory() returns a pointer to an ELF descriptor, the ELF image has read operations enabled (ELF_C_READ). image is a pointer to an image of the Elf file mapped into memory, sz is the size of the ELF image. An ELF image that is mapped in with elf_memory() may be read and modified, but the ELF image size may not be changed.

elf_next() provides sequential access to the next archive member. That is, having an ELF descriptor, elf, associated with an archive member, elf_next() prepares the containing archive to access the following member when the program calls elf_begin(). After successfully positioning an archive for the next member, elf_next() returns the value ELF_C_READ. Otherwise, the open file was not an archive, elf was NULL, or an error occurred, and the return value is ELF_C_NULL. In either case, the return value may be passed as an argument to elf_begin(), specifying the appropriate action.
elf_rand() provides random archive processing, preparing elf to access an arbitrary archive member. elf must be a descriptor for the archive itself, not a member within the archive. offset gives the byte offset from the beginning of the archive to the archive header of the desired member. See elf_getarsym(3E) for more information about archive member offsets. When elf_rand() works, it returns offset. Otherwise, it returns 0, because an error occurred, elf was NULL, or the file was not an archive (no archive member can have a zero offset). A program may mix random and sequential archive processing.

System Services

When processing a file, the library decides when to read or write the file, depending on the program’s requests. Normally, the library assumes the file descriptor remains usable for the life of the ELF descriptor. If, however, a program must process many files simultaneously and the underlying operating system limits the number of open files, the program can use elf_cntl() to let it reuse file descriptors. After calling elf_cntl() with appropriate arguments, the program may close the file descriptor without interfering with the library.

All data associated with an ELF descriptor remain allocated until elf_end() terminates the descriptor’s last activation. After the descriptors have been terminated, the storage is released; attempting to reference such data gives undefined behavior. Consequently, a program that deals with multiple input (or output) files must keep the ELF descriptors active until it finishes with them.

EXAMPLES

A prototype for reading a file appears on the next page. If the file is a simple object file, the program executes the loop one time, receiving a null descriptor in the second iteration. In this case, both elf and arf will have the same value, the activation count will be 2, and the program calls elf_end() twice to terminate the descriptor.
If the file is an archive, the loop processes each archive member in turn, ignoring those that are not object files.

```c
if (elf_version(EV_CURRENT) == EV_NONE)
{
    /* library out of date */
    /* recover from error */
}
```
```
cmd = ELF_C_READ;
arf = elf_begin(fldes, cmd, (Elf *)0);
while ((elf = elf_begin(fldes, 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.

```c
e1f_version(EV_CURRENT);
    arf = elf_begin(fldes, 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(fldes, 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).

```c
#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.

```c
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.

```c
elf_version(EV_CURRENT);
fildes = open("path/name", O_RDWR);
elf = elf_begin(fildes, ELF_C_RDWR, (Elf *)0);

/* add new or delete old information */
...

/* ensure that the memory image of the file is complete */
elf_update(elf, ELF_C_NULL);

elf_update(elf, ELF_C_WRITE); /* update file */
elf_end(elf);
```

Notice that both file creation examples open the file with write and read permissions. On systems that support `mmap(2)`, the library uses it to enhance performance, and `mmap(2)` requires a readable file descriptor. Although the library can use a write-only file...
descriptor, the application will not obtain the performance advantages of \texttt{mmap(2)}.

**ATTRIBUTES**

See \texttt{attributes(5)} for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**SEE ALSO**  \texttt{creat(2), lseek(2), mmap(2), open(2), elf(3E), elf32_getehdr(3E), elf_ctl(3E), elf_getahdr(3E), elf_getarsym(3E), elf_getbase(3E), elf_getdata(3E), elf_getscn(3E), elf_kind(3E), elf_rawfile(3E), elf_update(3E), elf_version(3E), ar(4), attributes(5)}
NAME  
elf_cntl – control an elf file descriptor

SYNOPSIS  
cc [ flag ... ] file ... -lelf [ library ... ]
#include <libelf.h>
int elf_cntl(Elf *elf, Elf_Cmd cmd);

DESCRIPTION  
elf_cntl() instructs the library to modify its behavior with respect to an ELF descriptor, elf. As elf_begin(3E) describes, an ELF descriptor can have multiple activations, and multiple ELF descriptors may share a single file descriptor. Generally, elf_cntl() commands apply to all activations of elf. Moreover, if the ELF descriptor is associated with an archive file, descriptors for members within the archive will also be affected as described below. Unless stated otherwise, operations on archive members do not affect the descriptor for the containing archive.

The cmd argument tells what actions to take and may have the following values:

ELF_C_FDDONE  
This value tells the library not to use the file descriptor associated with elf. A program should use this command when it has requested all the information it cares to use and wishes to avoid the overhead of reading the rest of the file. The memory for all completed operations remains valid, but later file operations, such as the initial elf_getdata() for a section, will fail if the data are not in memory already.

ELF_C_FDREAD  
This command is similar to ELF_C_FDDONE, except it forces the library to read the rest of the file. A program should use this command when it must close the file descriptor but has not yet read everything it needs from the file. After elf_cntl() completes the ELF_C_FDREAD command, future operations, such as elf_getdata(), will use the memory version of the file without needing to use the file descriptor.

If elf_cntl() succeeds, it returns 0. Otherwise elf was NULL or an error occurred, and the function returns -1.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  
elf(3E), elf_begin(3E), elf_getdata(3E), elf_rawfile(3E), attributes(5)

NOTES  
If the program wishes to use the “raw” operations (see elf_rawdata(), which elf_getdata(3E) describes, and elf_rawfile(3E)) after disabling the file descriptor with ELF_C_FDDONE or ELF_C_FDREAD, it must execute the raw operations explicitly beforehand. Otherwise, the raw file operations will fail. Calling elf_rawfile() makes the entire image available, thus supporting subsequent elf_rawdata() calls.

modified 29 Dec 1996  SunOS 5.6  3E-463
NAME

elf_errmsg, elf_errno – error handling

SYNOPSIS

cc [ flag ...] file ... -l elf [ library ...]
#include <libelf.h>
const char *elf_errmsg (int err);
int elf_errno(void);

DESCRIPTION

If an ELF library function fails, a program may call elf_errno() to retrieve the library’s internal error number. As a side effect, this function resets the internal error number to 0, which indicates no error.

elf_errmsg() takes an error number, err, and returns a null-terminated error message (with no trailing new-line) that describes the problem. A zero err 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 err 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.

EXAMPLES

The following fragment clears the internal error number and checks it later for errors. Unless an error occurs after the first call to elf_errno(), the next call will return 0.

(void)elf_errno();
/* processing ... */
while (more_to_do)
{
    if ((err = elf_errno()) != 0)
    {
        /* print msg */
        msg = elf_errmsg(err);
    }
}

ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO

elf(3E), attributes(5)
NAME
elf_fill — set fill byte

SYNOPSIS
cc [ flag ... ] file ... -lelf [ library ... ]
#include <libelf.h>
void elf_fill(int fill);

DESCRIPTION
Alignment constraints for ELF files sometimes require the presence of “holes.” For example, if the data for one section are required to begin on an eight-byte boundary, but the preceding section is too “short,” the library must fill the intervening bytes. These bytes are set to the fill character. The library uses zero bytes unless the application supplies a value. See elf_getdata(3E) for more information about these holes.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
elf(3E), elf_flagdata(3E), elf_getdata(3E), elf_update(3E), attributes(5)

NOTES
An application can assume control of the object file organization by setting the ELF_F_LAYOUT bit (see elf_flagdata(3E)). When this is done, the library does not fill holes.
NAME
elf_flagdata, elf_flagehdr, elf_flagelf, elf_flagphdr, elf_flagscn, elf_flagshdr – manipulate flags

SYNOPSIS
cc [ flag ... ] file ... -lelf [ library ... ]
#include <libelf.h>

unsigned elf_flagdata(Elf_Data *data, Elf_Cmd cmd, unsigned flags);
unsigned elf_flagehdr(Elf *elf, Elf_Cmd cmd, unsigned flags);
unsigned elf_flagelf(Elf *elf, Elf_Cmd cmd, unsigned flags);
unsigned elf_flagphdr(Elf *elf, Elf_Cmd cmd, unsigned flags);
unsigned elf_flagscn(Elf_Scn *scn, Elf_Cmd cmd, unsigned flags);
unsigned elf_flagshdr(Elf_Scn *scn, Elf_Cmd cmd, unsigned flags);

DESCRIPTION
These functions manipulate the flags associated with various structures of an ELF file.
Given an ELF descriptor (elf), a data descriptor (data), or a section descriptor (scn), the
functions may set or clear the associated status bits, returning the updated bits. A null
descriptor is allowed, to simplify error handling; all functions return 0 for this degenerate
case.

cmd may have the following values:

ELF_C_CLR  The functions clear the bits that are asserted in flags. Only the non-
            zero bits in flags are cleared; zero bits do not change the status of the
            descriptor.

ELF_C_SET  The functions set the bits that are asserted in flags. Only the non-zero
            bits in flags are set; zero bits do not change the status of the descrip-
            tor.

Descriptions of the defined flags bits appear below:

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 flags and
             cmd equal to ELF_C_SET. A later call to elf_update() would write the
             marked header to the file.

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 associ-
              ated with the descriptor.

When a flag bit is set for an item, it affects all the subitems as well. Thus, for example, if
the program sets the ELF_F_DIRTY bit with elf_flagelf(), the entire logical file is “dirty.”
EXAMPLES

The following fragment shows how one might mark the ELF header to be written to the output file:

```c
/* dirty ehdr ... */
ehdr = elf32_getehdr(elf);
elf_flagehdr(elf, ELF_C_SET, ELF_F_DIRTY);
```

ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO

elf(3E), elf32_getehdr(3E), elf_getdata(3E), elf_update(3E), attributes(5)
NAME    elf_getarhdr – retrieve archive member header

SYNOPSIS cc [ flag ...] file ... -lelf [ library ...]
#include <libelf.h>
Elf_Arhdr *elf_getarhdr(Elf *elf);

DESCRIPTION elf_getarhdr() returns a pointer to an archive member header, if one is available for the
ELF descriptor elf. Otherwise, no archive member header exists, an error occurred, or elf
was null; elf_getarhdr() then returns a null value. The header includes the following
members.

    char    *ar_name;
    time_t    ar_date;
    long    ar_uid;
    long    ar_gid;
    unsigned long    ar_mode;
    off_t    ar_size;
    char    *ar_rawname;

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 ter-
mminating slash and trailing blanks as specified in the archive format.

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.

    /    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)).

    //    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.

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.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
SEE ALSO  
elf(3E), elf_begin(3E), elf_getarsym(3E), ar(4), attributes(5)
NAME
elf_getarsym – retrieve archive symbol table

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

DESCRIPTION
elf_getarsym() returns a pointer to the archive symbol table, if one is available for the
ELF descriptor elf. Otherwise, the archive doesn’t have a symbol table, an error occurred,
or elf was null; elf_getarsym() then returns a null value. The symbol table is an array of
structures that include the following members.

- char *as_name;
- size_t as_off;
- unsigned long as_hash;

These members have the following semantics:

as_name   A pointer to a null-terminated symbol name resides here.
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 argu-
ments to elf_rand(). See elf_begin(3E) to access the desired archive
member.
as_hash   This is a hash value for the name, as computed by elf_hash().

If ptr is non-null, the library stores the number of table entries in the location to which ptr
points. This value is set to 0 when the return value is NULL. The table’s last entry, which
is included in the count, has a null as_name, a zero value for as_off, and ~0UL for
as_hash.

The hash value returned is guaranteed not to be the bit pattern of all ones (~0UL).

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
elf(3E), elf_begin(3E), elf_getarhdr(3E), elf_hash(3E), ar(4), attributes(5)
NAME                      elf_getbase – get the base offset for an object file
SYNOPSIS                  cc [ flag ... ] file ... -l elf [ library ... ]
                          #include <libelf.h>
                          off_t elf_getbase(Elf *elf);
DESCRIPTION                elf_getbase() returns the file offset of the first byte of the file or archive member associated with elf, if it is known or obtainable, and −1 otherwise. A null elf is allowed, to simplify error handling; the return value in this case is −1. The base offset of an archive member is the beginning of the member’s information, not the beginning of the archive member header.
ATTRIBUTES                See attributes(5) for descriptions of the following attributes:
                          ATTRIBUTE TYPE     ATTRIBUTE VALUE
                          MT-Level           MT-Safe
SEE ALSO                  elf(3E), elf_begin(3E), ar(4), attributes(5)
**NAME**
elf_getdata, elf_newdata, elf_rawdata – get section data

**SYNOPSIS**
```c
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);
```

**DESCRIPTION**
These functions access and manipulate the data associated with a section descriptor, `scn`. 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.

`elf_getdata()` lets a program step through a section’s data list. If the incoming data descriptor, `data`, is null, the function returns the first buffer associated with the section. Otherwise, `data` should be a data descriptor associated with `scn`, and the function gives the program access to the next data element for the section. If `scn` is null or an error occurs, `elf_getdata()` returns a null pointer.

`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 `class` (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.

`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 `scn` is null or an error occurs, `elf_newdata()` returns a null pointer.

`elf_rawdata()` differs from `elf_getdata()` by returning only uninterpreted bytes, regardless of the section type. This function typically should be used only to retrieve a section image from a file being read, and then only when a program must avoid the automatic data translation described below. Moreover, a program may not close or disable (see `elf_cntl(3E)`) the file descriptor associated with `elf` before the initial raw operation, because `elf_rawdata()` might read the data from the file to ensure it doesn’t interfere with `elf_getdata()`. See `elf_rawfile(3E)` for a related facility that applies to the entire file.

When `elf_getdata()` provides the right translation, its use is recommended over `elf_rawdata()`. If `scn` is null or an error occurs, `elf_rawdata()` returns a null pointer.
The Elf_Data structure includes the following members:

- **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 4, 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 0, 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.
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).

<table>
<thead>
<tr>
<th>Section Type</th>
<th>Elf_Type</th>
<th>32-Bit Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHT_DYNAMIC</td>
<td>ELF_T_DYN</td>
<td>Elf32_Dyn</td>
</tr>
<tr>
<td>SHT_DYNSYM</td>
<td>ELF_T_SYM</td>
<td>Elf32_Sym</td>
</tr>
<tr>
<td>SHT_HASH</td>
<td>ELF_T_WORD</td>
<td>Elf32_Word</td>
</tr>
<tr>
<td>SHT_NOBITS</td>
<td>ELF_T_BYTE</td>
<td>unsigned char</td>
</tr>
<tr>
<td>SHT_NOTE</td>
<td>ELF_T_BYTE</td>
<td>unsigned char</td>
</tr>
<tr>
<td>SHT_NULL</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td>SHT_PROGBITS</td>
<td>ELF_T_BYTE</td>
<td>unsigned char</td>
</tr>
<tr>
<td>SHT_REL</td>
<td>ELF_T_REL</td>
<td>Elf32_Rel</td>
</tr>
<tr>
<td>SHT_RELA</td>
<td>ELF_T_RELA</td>
<td>Elf32_Rea</td>
</tr>
<tr>
<td>SHT_SYMTAB</td>
<td>ELF_T_BYTE</td>
<td>unsigned char</td>
</tr>
<tr>
<td>SHT_SUNW_verdef</td>
<td>ELF_T_SYM</td>
<td>Elf32_Sym</td>
</tr>
<tr>
<td>SHT_SUNW_verneed</td>
<td>ELF_T_VDEF</td>
<td>Elf32_Verdef</td>
</tr>
<tr>
<td>SHT_SUNW_versym</td>
<td>ELF_T_HALF</td>
<td>Elf32_Versym</td>
</tr>
<tr>
<td>other</td>
<td>ELF_T_BYTE</td>
<td>unsigned char</td>
</tr>
</tbody>
</table>

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.

**ATTRIBUTES**

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**SEE ALSO**

elf(3E), elf32_getehdr(3E), elf32_getshdr(3E), elf32_xlatetof(3E), elf_cntl(3E),
elf_fill(3E), elf_flagdata(3E), elf_getscn(3E), elf_rawfile(3E), elf_strptr(3E),
elf_version(3E), attributes(5)
NAME       elf_getident – retrieve file identification data

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

DESCRIPTION As elf(3E) explains, ELF provides a framework for various classes of files, where basic objects may have 32 bits, 64 bits, etc. To accommodate these differences, without forcing the larger sizes on smaller machines, the initial bytes in an ELF file hold identification information common to all file classes. Every ELF header’s e_ident has EI_NIDENT bytes with the following interpretation:

<table>
<thead>
<tr>
<th>e_ident Index</th>
<th>Value</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>EI_MAG0</td>
<td>ELFMAG0</td>
<td>File identification</td>
</tr>
<tr>
<td>EI_MAG1</td>
<td>ELFMAG1</td>
<td></td>
</tr>
<tr>
<td>EI_MAG2</td>
<td>ELFMAG2</td>
<td></td>
</tr>
<tr>
<td>EI_MAG3</td>
<td>ELFMAG3</td>
<td></td>
</tr>
<tr>
<td>EI_CLASS</td>
<td>ELFCLASSTNONE</td>
<td>File class</td>
</tr>
<tr>
<td></td>
<td>ELFCCLASS32</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ELFCCLASS64</td>
<td></td>
</tr>
<tr>
<td>EI_DATA</td>
<td>ELFDATANONE</td>
<td>Data encoding</td>
</tr>
<tr>
<td></td>
<td>ELFDATA2LSB</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ELFDATA2MSB</td>
<td></td>
</tr>
<tr>
<td>EI_VERSION</td>
<td>EV_CURRENT</td>
<td>File version</td>
</tr>
<tr>
<td>7-15</td>
<td>0</td>
<td>Unused, set to zero</td>
</tr>
</tbody>
</table>

Other kinds of files (see elf_kind(3E)) also may have identification data, though they would not conform to e_ident.

elf_getident() returns a pointer to the file’s “initial bytes.” If the library recognizes the file, a conversion from the file image to the memory image may occur. In any case, the identification bytes are guaranteed not to have been modified, though the size of the unmodified area depends on the file type. If ptr is non-null, the library stores the number of identification bytes in the location to which ptr points. If no data are present, elf is null, or an error occurs, the return value is a null pointer, with 0 stored through ptr, if ptr is non-null.
ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO elf(3E), elf32_getehdr(3E), elf_begin(3E), elf_kind(3E), elf_rawfile(3E), attributes(5)
NAME
elf_getscn, elf_ndxscn, elf_newscn, elf_nextscn — get section information

SYNOPSIS
cc [ flag ...] file ... -l elf [ 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);

DESCRIPTION
These functions provide indexed and sequential access to the sections associated with the
ELF descriptor elf. 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).

elf_getscn() returns a section descriptor, given an index into the file’s section header
table. Note that the first “real” section has an index of 1. Although a program can get a
section descriptor for the section whose index 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 elf is null, elf_getscn() returns a null
pointer.

elf_newscn() creates a new section and appends it to the list for elf. 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 elf is null, elf_newscn() returns a null
pointer.

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 asso-
ciated 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 sec-
tion (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.

elf_nextscn() takes an existing section descriptor, scn, and returns a section descriptor
for the next higher section. One may use a null scn 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.

elf_ndxscn() takes an existing section descriptor, scn, and returns its section table index.
If scn is null or an error occurs, elf_ndxscn() returns SHN_UNDEF.
EXAMPLES
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 */
}
```

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO elf(3E), elf32_getehdr(3E), elf32_getshdr(3E), elf_begin(3E), elf_flagdata(3E), elf_getdata(3E), attributes(5)
NAME
elf_hash — compute hash value

SYNOPSIS
cc [ flag ...] file ... -lelf [ library ...]
#include <libelf.h>
unsigned long elf_hash(const char *name);

DESCRIPTION
elf_hash() computes a hash value, given a null terminated string, name. The returned hash value, h, can be used as a bucket index, typically after computing h mod x to ensure appropriate bounds.

Hash tables may be built on one machine and used on another because elf_hash() uses unsigned arithmetic to avoid possible differences in various machines’ signed arithmetic. Although name is shown as char* above, elf_hash() treats it as unsigned char* to avoid sign extension differences. Using char* eliminates type conflicts with expressions such as elf_hash(name).

ELF files’ symbol hash tables are computed using this function (see elf_getdata(3E) and elf32_xlatetof(3E)). The hash value returned is guaranteed not to be the bit pattern of all ones ("0UL").

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
elf(3E), elf32_xlatetof(3E), elf_getdata(3E), attributes(5)
NAME       elf_kind – determine file type

SYNOPSIS  cc [ flag ...] file ... -lelf [ library ...]
#include <libelf.h>
Elf_Kind elf_kind(Elf *elf);

DESCRIPTION This function returns a value identifying the kind of file associated with an ELF descriptor (elf). Defined values are below:

ELF_K_AR    The file is an archive [see ar(4)]. An ELF descriptor may also be associated with an archive member, not the archive itself, and then elf_kind() identifies the member’s type.

ELF_K_COFF  The file is a COFF object file. elf_begin(3E) describes the library’s handling for COFF files.

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.

ELF_K_NONE  This indicates a kind of file unknown to the library.

Other values are reserved, to be assigned as needed to new kinds of files. elf 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.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO    elf(3E), elf32_getehdr(3E), elf_begin(3E), elf_getident(3E), ar(4), attributes(5)
elf_rawfile — retrieve uninterpreted file contents

NAME

elf_rawfile — retrieve uninterpreted file contents

SYNOPSIS

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

DESCRIPTION

elf_rawfile() returns a pointer to an uninterpreted byte image of the file. This function
should be used only to retrieve a file being read. For example, a program might use
elf_rawfile() to retrieve the bytes for an archive member.

A program may not close or disable (see elf_cntl(3E)) the file descriptor associated with
elf before the initial call to elf_rawfile(), because elf_rawfile() might have to read the
data from the file if it does not already have the original bytes in memory. Generally, this
function is more efficient for unknown file types than for object files. The library impli-
citly translates object files in memory, while it leaves unknown files unmodified. Thus,
asking for the uninterpreted image of an object file may create a duplicate copy in
memory.

elf_rawdata() is a related function, providing access to sections within a file. See
elf_getdata(3E).

If ptr is non-null, the library also stores the file’s size, in bytes, in the location to which ptr
points. If no data are present, elf is null, or an error occurs, the return value is a null
pointer, with 0 stored through ptr, if ptr is non-null.

ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO

elf(3E), elf32_getehdr(3E), elf_begin(3E), elf_cntl(3E), elf_getdata(3E), elf_getident(3E),
elf_kind(3E), attributes(5)

NOTES

A program that uses elf_rawfile() and that also interprets the same file as an object file
potentially has two copies of the bytes in memory. If such a program requests the raw
image first, before it asks for translated information (through such functions as
elf32_getehdr(), elf_getdata(), and so on), the library “freezes” its original memory copy
for the raw image. It then uses this frozen copy as the source for creating translated
objects, without reading the file again. Consequently, the application should view the
raw file image returned by elf_rawfile() as a read-only buffer, unless it wants to alter its
own view of data subsequently translated. In any case, the application may alter the
translated objects without changing bytes visible in the raw image.

Multiple calls to elf_rawfile() with the same ELF descriptor return the same value; the
library does not create duplicate copies of the file.

3E-482 SunOS 5.6 modified 29 Dec 1996
NAME
elf_strptr – make a string pointer

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

DESCRIPTION
This function converts a string section offset to a string pointer. elf identifies the file in
which the string section resides, and section identifies the section table index for the
strings. elf_strptr() normally returns a pointer to a string, but it returns a null pointer
when elf is null, section is invalid or is not a section of type SHT_STRTAB, the section data
cannot be obtained, offset is invalid, or an error occurs.

EXAMPLES
A prototype for retrieving section names appears below. The file header specifies the sec-
tion name string table in the e_shstrndx member. The following code loops through the
sections, printing their names.

```c
/* 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)");
}
```

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
elf(3E), elf32_getshdr(3E), elf32_xlatetof(3E), elf_getdata(3E), attributes(5)

NOTES
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().

modified 29 Dec 1996 SunOS 5.6 3E-483
NAME
elf_update – update an ELF descriptor

SYNOPSIS
cc [ flag ...] file ... -lelf [ library ...]
#include <libelf.h>
off_t elf_update(Elf *elf, Elf_Cmd cmd);

DESCRIPTION
elf_update() causes the library to examine the information associated with an ELF
descriptor, elf, and to recalculate the structural data needed to generate the file’s image.
cmd may have the following values:

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)).

ELF_C_WRITE
If cmd has this value, elf_update() duplicates its ELF_C_NULL actions
and also writes any “dirty” information associated with the ELF
descriptor to the file. That is, when a program has used
elf_getdata(3E) or the elf_flagdata(3E) facilities to supply new (or
update existing) information for an ELF descriptor, those data will be
examined, coordinated, translated if necessary (see elf32_xlatetof(3E)),
and written to the file. When portions of the file are written, any
ELF_F_DIRTY bits are reset, indicating those items no longer need to be
written to the file (see elf_flagdata(3E)). The sections’ data are written
in the order of their section header entries, and the section header table
is written to the end of the file.

When the ELF descriptor was created with elf_begin(), it must have
allowed writing the file. That is, the elf_begin() command must have
been either ELF_C_RDWR or ELF_C_WRITE.

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.

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 pro-
gram.
The following table shows ELF Header members:

<table>
<thead>
<tr>
<th>Member</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>e_ident[EI_DATA]</td>
<td>Library controls other e_ident values</td>
</tr>
<tr>
<td>e_type</td>
<td></td>
</tr>
<tr>
<td>e_machine</td>
<td></td>
</tr>
<tr>
<td>e_version</td>
<td></td>
</tr>
<tr>
<td>e_entry</td>
<td></td>
</tr>
<tr>
<td>e_phooff</td>
<td>Only when ELF_F_LAYOUT asserted</td>
</tr>
<tr>
<td>e_shoff</td>
<td>Only when ELF_F_LAYOUT asserted</td>
</tr>
<tr>
<td>e_flags</td>
<td></td>
</tr>
<tr>
<td>e_shstrndx</td>
<td></td>
</tr>
</tbody>
</table>

The following table shows the Program Header members:

<table>
<thead>
<tr>
<th>Member</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_type</td>
<td>The application controls all program header entries</td>
</tr>
<tr>
<td>p_offset</td>
<td></td>
</tr>
<tr>
<td>p_vaddr</td>
<td></td>
</tr>
<tr>
<td>p_paddr</td>
<td></td>
</tr>
<tr>
<td>p_filesz</td>
<td></td>
</tr>
<tr>
<td>p_memsz</td>
<td></td>
</tr>
<tr>
<td>p_flags</td>
<td></td>
</tr>
<tr>
<td>p_align</td>
<td></td>
</tr>
</tbody>
</table>

The following table shows the Section Header members:

<table>
<thead>
<tr>
<th>Member</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>sh_name</td>
<td></td>
</tr>
<tr>
<td>sh_type</td>
<td></td>
</tr>
<tr>
<td>sh_flags</td>
<td></td>
</tr>
<tr>
<td>sh_addr</td>
<td></td>
</tr>
<tr>
<td>sh_offset</td>
<td>Only when ELF_F_LAYOUT asserted</td>
</tr>
<tr>
<td>sh_size</td>
<td>Only when ELF_F_LAYOUT asserted</td>
</tr>
<tr>
<td>sh_link</td>
<td></td>
</tr>
<tr>
<td>sh_info</td>
<td></td>
</tr>
<tr>
<td>sh_addralign</td>
<td>Only when ELF_F_LAYOUT asserted</td>
</tr>
<tr>
<td>sh_entsize</td>
<td></td>
</tr>
</tbody>
</table>
The following table shows the Data Descriptor members:

<table>
<thead>
<tr>
<th>Member</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>d_buf</td>
<td></td>
</tr>
<tr>
<td>d_type</td>
<td></td>
</tr>
<tr>
<td>d_size</td>
<td></td>
</tr>
<tr>
<td>d_off</td>
<td>Only when ELF_F_LAYOUT asserted</td>
</tr>
<tr>
<td>d_align</td>
<td></td>
</tr>
<tr>
<td>d_version</td>
<td></td>
</tr>
</tbody>
</table>

Note that the program is responsible for two particularly important members (among others) in the ELF header. The e_version member controls the version of data structures written to the file. If the version is EV_NONE, the library uses its own internal version.

The e_ident[EI_DATA] entry controls the data encoding used in the file. As a special case, the value may be ELFDATANONE to request the native data encoding for the host machine. An error occurs in this case if the native encoding doesn’t match a file encoding known by the library.

Further note that the program is responsible for the sh_entsize section header member. Although the library sets it for sections with known types, it cannot reliably know the correct value for all sections. Consequently, the library relies on the program to provide the values for unknown section types. If the entry size is unknown or not applicable, the value should be set to 0.

When deciding how to build the output file, elf_update() obeys the alignments of individual data buffers to create output sections. A section’s most strictly aligned data buffer controls the section’s alignment. The library also inserts padding between buffers, as necessary, to ensure the proper alignment of each buffer.

**ATTRIBUTES**

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**SEE ALSO**

elf(3E), elf32_fsize(3E), elf32_getehdr(3E), elf32_getshdr(3E), elf32_xlatetof(3E), elf_begin(3E), elf_flagdata(3E), elf_getdata(3E), attributes(5)

**NOTES**

As mentioned above, the ELF_C_WRITE command translates data as necessary, before writing them to the file. This translation is not always transparent to the application program. If a program has obtained pointers to data associated with a file (for example, see elf32_getehdr(3E) and elf_getdata(3E)), the program should reestablish the pointers after calling elf_update().
NAME     elf_version – coordinate ELF library and application versions

SYNOPSIS cc [ flag ...] file ... -lelf [ library ...]
#include <libelf.h>
unsigned elf_version(unsigned ver);

DESCRIPTION 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 internal version. It further lets the program specify what memory types it uses by giving its own working version, ver, to the library. Every program that uses the ELF library must coordinate versions as described below.

The header <libelf.h> supplies the version to the program with the macro EV_CURRENT. 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.

Passing ver equal to EV_NONE causes elf_version() to return the library’s internal version, without altering the working version. If ver 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 EV_NONE.

EXAMPLES The following excerpt from an application program protects itself from using an older library:

    if (elf_version(EV_CURRENT) == EV_NONE) {
        /* library out of date */
        /* recover from error */
    }

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO elf(3E), elf32_xlatetof(3E), elf_begin(3E), attributes(5)

NOTES The working version should be the same for all operations on a particular ELF descriptor. Changing the version between operations on a descriptor will probably not give the expected results.
NAME encrypt – encoding function

SYNOPSIS

```c
#include <unistd.h>
void encrypt (char block[64], int edflag);
```

DESCRIPTION

The `encrypt()` function provides (rather primitive) access to the hashing algorithm employed by the `crypt(3C)` function. The key generated by `setkey(3C)` is used to encrypt the string `block` with `encrypt()`. The `block` argument to `encrypt()` is an array of length 64 bytes containing only the bytes with numerical value of 0 and 1. The array is modified in place to a similar array using the key set by `setkey(3C)`. If `edflag` is 0, the argument is encoded. If `edflag` is 1, the argument may be decoded (see the USAGE section below); if the argument is not decoded, `errno` will be set to `ENOSYS`.

RETURN VALUES

The `encrypt()` function returns no value.

ERRORS

The `encrypt()` function will fail if:

- `ENOSYS` The functionality is not supported on this implementation.

USAGE

In some environments, decoding may not be implemented. This is related to U.S. Government restrictions on encryption and decryption routines: the DES decryption algorithm cannot be exported outside the U.S.A. Historical practice has been to ship a different version of the encryption library without the decryption feature in the routines supplied. Thus the exported version of `encrypt()` does encoding but not decoding. Because `encrypt()` does not return a value, applications wishing to check for errors should set `errno` to 0, call `encrypt()`, then test `errno` and, if it is non-zero, assume an error has occurred.

ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO

`crypt(3C)`, `setkey(3C)`, `attributes(5)`
NAME

end, _end, etext, _etext, edata, _edata – last locations in program

SYNOPSIS

extern _etext;
extern _edata;
extern _end;

DESCRIPTION

These names refer neither to routines nor to locations with interesting contents; only their addresses are meaningful.

_ etext  The address of _etext is the first location after the program text.
_ edata  The address of _edata is the first location after the initialized data region.
_ end    The address of _end is the first location after the uninitialized data region.

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)).

References to end, etext, and edata, without a preceding underscore can be made by the user; if this case is detected the symbol will be aliased to the associated symbol which begins with the underscore.
NAME
endhostent, gethostbyaddr, gethostbyname, gethostent, sethostent – network host database functions

SYNOPSIS
cc [ flag ...] file ... -lxnet [ library ...]
#include <netdb.h>
extern int h_errno;
void endhostent(void);
struct hostent *gethostbyaddr(const void *addr, size_t len, int type);
struct hostent *gethostbyname(const char *name);
struct hostent *gethostent(void);
void sethostent(int stayopen);

DESCRIPTION
The gethostent(), gethostbyaddr(), and gethostbyname() functions each return a pointer to a hostent structure, the members of which contain the fields of an entry in the network host database.

The gethostent() function reads the next entry of the database, opening a connection to the database if necessary.

The gethostbyaddr() function searches the database from the beginning and finds the first entry for which the address family specified by type matches the h_addrtype member and the address pointed to by addr occurs in h_addrlist, opening a connection to the database if necessary. The addr argument is a pointer to the binary-format (that is, not null-terminated) address in network byte order, whose length is specified by the len argument. The datatype of the address depends on the address family. For an address of type AF_INET, this is an in_addr structure, defined in <netinet/in.h>.

The gethostbyname() function searches the database from the beginning and finds the first entry for which the host name specified by name matches the h_name member, opening a connection to the database if necessary.

The sethostent() function opens a connection to the network host database, and sets the position of the next entry to the first entry. If the stayopen argument is non-zero, the connection to the host database will not be closed after each call to gethostent() (either directly, or indirectly through one of the other gethost*() functions).

The endhostent() function closes the connection to the database.

RETURN VALUES
On successful completion, gethostbyaddr(), gethostbyname() and gethostent() return a pointer to a hostent structure if the requested entry was found, and a null pointer if the end of the database was reached or the requested entry was not found. Otherwise, a null pointer is returned.

On unsuccessful completion, gethostbyaddr() and gethostbyname() functions set h_errno to indicate the error.
ERRORS
No errors are defined for endhostent(), gethostent() and sethostent().

The gethostbyaddr() and gethostbyname() functions will fail in the following cases, setting h_errno to the value shown in the list below. Any changes to errno are unspecified.

HOST_NOT_FOUND  No such host is known.
TRY_AGAIN       A temporary and possibly transient error occurred, such as a failure of a server to respond.
NO_RECOVERY     An unexpected server failure occurred which cannot be recovered.
NO_DATA         The server recognized the request and the name but no address is available. Another type of request to the name server for the domain might return an answer.

USAGE
The gethostent(), gethostbyaddr(), and gethostbyname() functions may return pointers to static data, which may be overwritten by subsequent calls to any of these functions. These functions are generally used with the Internet address family.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
endservent(3XN), htonl(3XN), inet_addr(3XN), attributes(5), netdb(5)
NAME
endnetent, getnetbyaddr, getnetbyname, getnetent, setnetent – network database functions

SYNOPSIS
c
cc [flag ...] file ... -lxnet [library ...]
#include <netdb.h>
void endnetent(void);
struct netent *getnetbyaddr(in_addr_t net, int type);
struct netent *getnetbyname(const char *name);
struct netent *getnetent(void);
void setnetent(int stayopen);

DESCRIPTION
The getnetbyaddr(), getnetbyname() and getnetent(), functions each return a pointer to
a netent structure, the members of which contain the fields of an entry in the network
database.
The getnetent() function reads the next entry of the database, opening a connection to the
database if necessary.
The getnetbyaddr() function searches the database from the beginning, and finds the first
entry for which the address family specified by type matches the n_addrtype member
and the network number net matches the n_net member, opening a connection to the
database if necessary. The net argument is the network number in host byte order.
The getnetbyname() function searches the database from the beginning and finds the
first entry for which the network name specified by name matches the n_name member,
opening a connection to the database if necessary.
The setnetent() function opens and rewinds the database. If the stayopen argument is
non-zero, the connection to the net database will not be closed after each call to getnetent()
(either directly, or indirectly through one of the other getnet*() functions).
The endnetent() function closes the database.

RETURN VALUES
On successful completion, getnetbyaddr(), getnetbyname() and getnetent(), return a
pointer to a netent structure if the requested entry was found, and a null pointer if the
end of the database was reached or the requested entry was not found. Otherwise, a null
pointer is returned.

ERRORS
No errors are defined.

USAGE
The getnetbyaddr(), getnetbyname() and getnetent(), functions may return pointers to
static data, which may be overwritten by subsequent calls to any of these functions.
These functions are generally used with the Internet address family.
ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO

attributes(5), netdb(5)
NAME  
endprotoent, getprotobynumber, getprotobyname, getprotoent, setprotoent – network protocol database functions

SYNOPSIS  
cc [ flag ... ] file ... -lxnet [ library ... ]
#include <netdb.h>
void endprotoent(void);
struct protoent *getprotobynumber(const char *name);
struct protoent *getprotobynumber(int proto);
struct protoent *getprotoent(void);
void setprotoent(int stayopen);

DESCRIPTION  
The getprotobynumber(), getprotobynumber() and getprotoent(), functions each return a pointer to a protoent structure, the members of which contain the fields of an entry in the network protocol database.

The getprotoent() function reads the next entry of the database, opening a connection to the database if necessary.

The getprotobynumber() function searches the database from the beginning and finds the first entry for which the protocol name specified by name matches the p_name member, opening a connection to the database if necessary.

The getprotobynumber() function searches the database from the beginning and finds the first entry for which the protocol number specified by number matches the p_proto member, opening a connection to the database if necessary.

The setprotoent() function opens a connection to the database, and sets the next entry to the first entry. If the stayopen argument is non-zero, the connection to the network protocol database will not be closed after each call to getprotoent() (either directly, or indirectly through one of the other getproto*() functions).

The endprotoent() function closes the connection to the database.

RETURN VALUES  
On successful completion, getprotobynumber(), getprotobynumber() and getprotoent() functions return a pointer to a protoent structure if the requested entry was found, and a null pointer if the end of the database was reached or the requested entry was not found. Otherwise, a null pointer is returned.

ERRORS  
No errors are defined.

USAGE  
The getprotobynumber(), getprotobynumber() and getprotoent() functions may return pointers to static data, which may be overwritten by subsequent calls to any of these functions.

These functions are generally used with the Internet address family.
ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO

attributes(5), netdb(5)
NAME .endservent, getservbyport, getservbyname, getservent, setservent – network services database functions

SYNOPSIS  cc [ flag  ... ] file  ... -lxnet [ library  ... ]
#include <netdb.h>
void endservent(void);
struct servent *getservbyname(const char *name, const char *proto);
struct servent *getservbyport(int port, const char *proto);
struct servent *getservent(void);
void setservent(int stayopen);

DESCRIPTION  The getservbyname(), getservbyport() and getservent() functions each return a pointer to a servent structure, the members of which contain the fields of an entry in the network services database.

The getservent() function reads the next entry of the database, opening a connection to the database if necessary.

The getservbyname() function searches the database from the beginning and finds the first entry for which the service name specified by name matches the s_name member and the protocol name specified by proto matches the s_proto member, opening a connection to the database if necessary. If proto is a null pointer, any value of the s_proto member will be matched.

The getservbyport() function searches the database from the beginning and finds the first entry for which the port specified by port matches the s_port member and the protocol name specified by proto matches the s_proto member, opening a connection to the database if necessary. If proto is a null pointer, any value of the s_proto member will be matched. The port argument must be in network byte order.

The setservent() function opens a connection to the database, and sets the next entry to the first entry. If the stayopen argument is non-zero, the net database will not be closed after each call to the getservent() function (either directly, or indirectly through one of the other getserv*() functions).

The endservent() function closes the database.

RETURN VALUES  On successful completion, getservbyname(), getservbyport() and getservent() return a pointer to a servent structure if the requested entry was found, and a null pointer if the end of the database was reached or the requested entry was not found. Otherwise, a null pointer is returned.

ERRORS  No errors are defined.

USAGE  The port argument of getservbyport() need not be compatible with the port values of all address families.

3XN-496  SunOS 5.6  modified 16 May 1997
The `getservent()`, `getservbyname()` and `getservbyport()` functions may return pointers to static data, which may be overwritten by subsequent calls to any of these functions. These functions are generally used with the Internet address family.

**ATTRIBUTES**

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

**SEE ALSO**

`endhostent(3XN)`, `endprotoent(3XN)`, `htonl(3XN)`, `inet_addr(3XN)`, `attributes(5)`, `netdb(5)`
NAME
endwin, isendwin – restore initial terminal environment

SYNOPSIS
#include <curses.h>
int endwin(void);
int isendwin(void);

DESCRIPTION
The endwin() function restores tty modes, resets the terminal, and moves the cursor to
the lower left corner of the screen. This function should be called before exiting or escaping X/Open Curses temporarily. To resume X/Open Curses after a temporary escape, call refresh(3XC) or doupdate(3XC).

If the program interacts with multiple terminals, call endwin() for each terminal.

The isendwin() function determines whether or not a screen has been refreshed.

RETURN VALUES
On success, the endwin() function returns OK. Otherwise, it returns ERR.

The isendwin() function returns TRUE if endwin() has been called without subsequent calls to refresh(). Otherwise, it returns FALSE.

ERRORS
None.

SEE ALSO
doupdate(3XC)
NAME
erasechar, erasewchar, killchar, killwchar – return current ERASE or KILL characters

SYNOPSIS
#include <curses.h>
char erasechar(void);
int erasewchar(wchar_t *ch);
char killchar(void);
int killwchar(wchar_t *ch);

ARGUMENTS
ch Is a pointer to a location where a character may be stored.

DESCRIPTION
The erasechar() function returns the current ERASE character from the tty driver. This character is used to delete the previous character during keyboard input. The returned value can be used when including deletion capability in interactive programs.

The killchar() function is similar to erasechar(). It returns the current KILL character.

The erasewchar() and killwchar() functions are similar to erasechar() and killchar() respectively, but store the ERASE or KILL character in the object pointed to by ch.

RETURN VALUES
For erasechar() and killchar(), the terminal’s current ERASE or KILL character is returned.

On success, the erasewchar() and killwchar() functions return OK. Otherwise, they return ERR.

SEE ALSO
getch(3XC), getstr(3XC), get_wch(3XC)
NAME | erf, erfc – error and complementary error functions

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

DESCRIPTION | The erf() function computes the error function of x, defined as:
\[
\frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} \, dt
\]
The erfc() function computes 1.0 – erf(x).

RETURN VALUES | Upon successful completion, erf() and erfc() return the value of the error function and complementary error function, respectively.
If x is NaN, NaN is returned.

ERRORS | No errors will occur.

USAGE | The erfc() function is provided because of the extreme loss of relative accuracy if erf(x) is called for large x and the result subtracted from 1.0.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO | isnan(3M), attributes(5)
NAME ethers, ether_ntoa, ether_aton, ether_ntohost, ether_hostton, ether_line – Ethernet address mapping operations

SYNOPSIS cc [flag ...] file ... -lssocket -lnsl [ library ... ]
#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);

DESCRIPTION

These routines are useful for mapping 48 bit Ethernet numbers to their ASCII representations or their corresponding host names, and vice versa.

The function ether_ntoa() converts a 48 bit Ethernet number pointed to by e to its standard ASCII representation; it returns a pointer to the ASCII string. The representation is of the form x:x:x:x:x:x where x 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.

The function ether_ntohost() maps an Ethernet number (pointed to by e) to its associated hostname. The string pointed to by hostname must be long enough to hold the hostname and a NULL character. The function returns zero upon success and non-zero upon failure. Inversely, the function ether_hostton() maps a hostname string to its corresponding Ethernet number; the function modifies the Ethernet number pointed to by e. 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).

The function ether_line() scans a line (pointed to by l) and sets the hostname and the Ethernet number (pointed to by e). The string pointed to by hostname must be long enough to hold the hostname and a NULL character. The function returns zero upon success and non-zero upon failure. The format of the scanned line is described by ethers(4).
### FILES

/etc/ethers  
/etc/nsswitch.conf

### ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### SEE ALSO

`ethers(4), nsswitch.conf(4), attributes(5)`

### BUGS

Programs that call `ether_hostton()` or `ether_ntohost()` routines cannot be linked statically since the implementation of these routines requires dynamic linker functionality to access shared objects at run time.
NAME

euclen, euccol, eucscol – get byte length and display width of EUC characters

SYNOPSIS

#include <euc.h>

int euclen(const unsigned char *s);
int euccol(const unsigned char *s);
int eucscol(const unsigned char *str);

DESCRIPTION

euclen() returns the length in bytes of the Extended Unix Code (EUC) character pointed to by \texttt{s}, including single-shift characters, if present.
euccol() returns the screen column width of the EUC character pointed to by \texttt{s}.
eucscol() returns the screen column width of the EUC string pointed to by \texttt{str}.

For the \texttt{euclen()} and \texttt{euccol()}, routines, \texttt{s} points to the first byte of the character. This byte is examined to determine its codeset. The character type table for the current \texttt{locale} is used for codeset byte length and display width information.

ATTRIBUTES

See \texttt{attributes(5)} for descriptions of the following attributes:

\begin{tabular}{|c|c|}
\hline
ATTRIBUTE TYPE & ATTRIBUTE VALUE \\
\hline
MT-Level & MT-Safe with exceptions \\
\hline
\end{tabular}

SEE ALSO

getwidth(3C), setlocale(3C), attributes(5)

NOTES

These functions will only work with EUC locales.

These functions can be used safely in multi-thread applications, as long as \texttt{setlocale(3C)} is not called to change the locale.
NAME
exit, _exithandle – terminate process

SYNOPSIS
#include <stdlib.h>
void exit(int status);
void _exithandle(void);

DESCRIPTION
The exit() function terminates a process by first calling _exithandle() and then _exit.
The _exithandle() function calls any functions registered through the atexit(3C) function
in the reverse order of their registration. This action includes executing all finalization
code from the .fini sections of all objects that are part of the process.
The _exithandle() function is intended for use only with _exit(), and allows for specialized
processing such as dldump(3X) to be performed. Normal process execution should
not be continued after a call to _exithandle() has occurred, as internal data structures
may have been torn down due to atexit() or .fini processing.
The symbols EXIT_SUCCESS and EXIT_FAILURE are defined in the header <stdlib.h> and
may be used as the value of status to indicate successful or unsuccessful termination,
respectively.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
exit(2), atexit(3C), dldump(3X), attributes(5)
NAME  exp – exponential function

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

DESCRIPTION  The exp() function computes the exponential of x, defined as $e^x$.

RETURN VALUES  Upon successful completion, exp() returns the exponential of x.
If the correct value would cause overflow, exp() returns HUGE_VAL and sets errno to ERANGE.
If the correct value would cause underflow to zero, exp() returns 0 and may set errno to ERANGE.
If x is NaN, NaN is returned.
For exceptional cases, matherr(3M) tabulates the values to be returned as dictated by Standards other than XPG4.

ERRORS  The exp() function will fail if:
ERANGE   The result overflows.
The exp() function may fail if:
ERANGE   The result underflows.

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

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  isnan(3M), log(3M), matherr(3M), mp(3M), attributes(5), standards(5)

NOTES  Prior to Solaris 2.6, there was a conflict between the pow() function in this library and the pow() function in the libmp library. This conflict was resolved by prepending mp_ to all functions in the libmp library. See mp(3M) for details.

modified 29 Dec 1996  SunOS 5.6  3M-505
NAME  expm1 – computes exponential functions

SYNOPSIS  cc [ flag ... ] file ... -lm [ library ... ]
           #include <math.h>
           double expm1(double x);

DESCRIPTION  The expm1() function computes $e^x - 1.0$.

RETURN VALUES  If $x$ is NaN, then the function returns NaN.
If $x$ is positive infinity, expm1() returns positive infinity.
If $x$ is negative infinity, expm1() returns $-1.0$.
If the value overflows, expm1() returns HUGE_VAL.

ERRORS  No errors will occur.

USAGE  The value of expm1($x$) may be more accurate than exp($x$)$-1.0$ for small values of $x$.

The expm1() and log1p(3M) functions are useful for financial calculations of

\[
\frac{(1+x)^n - 1}{x}
\]

when $x$ is very small (for example, when performing calculations with a small daily
interest rate). These functions also simplify writing accurate inverse hyperbolic functions.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  exp(3M), ilogb(3M), log1p(3M), attributes(5)
NAME          fabs – absolute value function

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

DESCRIPTION    The fabs() function computes the absolute value of x, |x|.

RETURN VALUES  Upon successful completion, fabs() returns the absolute value of x.
                If x is NaN, NaN is returned.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO       isnan(3M), attributes(5)
NAME
fattach – attach a STREAMS-based file descriptor to an object in the file system name space

SYNOPSIS
int fattach(int fildes, const char *path);

DESCRIPTION
The fattach() function attaches a STREAMS-based file descriptor to an object in the file system name space, effectively associating a name with fildes. fildes must be a valid open file descriptor representing a STREAMS file. path 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 path will operate on the STREAMS file until the STREAMS file is detached from the node. fildes can be attached to more than one path, that is, a stream can have several names associated with it.

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 path, the number of links is set to 1, and the size and device identifier are set to those of the streams device associated with fildes. If any attributes of the named stream are subsequently changed (for example, chmod(2)), the attributes of the underlying object are not affected.

RETURN VALUES
If successful, fattach() returns 0; otherwise it returns −1 and sets errno to indicate an error.

ERRORS
Under the following conditions, the function fattach() fails and sets errno to:

- EACCES The user is the owner of path but does not have write permissions on path or fildes is locked.
- EBADF The fildes argument is not a valid open file descriptor.
- EBUSY The path argument is currently a mount point or has a STREAMS file descriptor attached it.
- EINVAL The path argument is a file in a remotely mounted directory.
- EINVAL The fildes argument does not represent a STREAMS file.
- ELOOP Too many symbolic links were encountered in translating path.
- ENAMETOOLONG The size of path exceeds {PATH_MAX}, or the component of a path name is longer than {NAME_MAX} while {_POSIX_NO_TRUNC} is in effect.
- ENOENT The path argument does not exist.
- ENOTDIR A component of a path prefix is not a directory.
- EPERM The effective user ID is not the owner of path or a user with the appropriate privileges.
ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO

fdetach(1M), chmod(2), mount(2), stat(2), fdetach(3C), isastream(3C), attributes(5), streamio(7I)

STREAMS Programming Guide
NAME
fclose – close a stream

SYNOPSIS
#include <stdio.h>

int fclose(FILE *stream);

DESCRIPTION
The `fclose()` function causes the stream pointed to by `stream` to be flushed and the associated file to be closed. Any unwritten buffered data for the stream is written to the file; any unread buffered data is discarded. The stream is disassociated from the file. If the associated buffer was automatically allocated, it is deallocated. It marks for update the `st_ctime` and `st_mtime` fields of the underlying file, if the stream was writable, and if buffered data had not been written to the file yet. The `fclose()` function will perform a `close(2)` on the file descriptor that is associated with the stream pointed to by `stream`. After the call to `fclose()`, any use of `stream` causes undefined behavior.

The `fclose()` function is performed automatically for all open files upon calling `exit(2)`.

RETURN VALUES
Upon successful completion, `fclose()` returns 0. Otherwise, it returns EOF and sets `errno` to indicate the error.

ERRORS
The `fclose()` function will fail if:

- **EAGAIN**: The `O_NONBLOCK` flag is set for the file descriptor underlying `stream` and the process would be delayed in the write operation.
- **EBADF**: The file descriptor underlying `stream` is not valid.
- **EFSBIG**: An attempt was made to write a file that exceeds the maximum file size or the process’ file size limit.
- **EFSBIG**: The file is a regular file and an attempt was made to write at or beyond the offset maximum associated with the corresponding stream.
- **EINTR**: The `fclose()` function was interrupted by a signal.
- **EIO**: The process is a member of a background process group attempting to write to its controlling terminal, TOSTOP is set, the process is neither ignoring nor blocking SIGTTOU and the process group of the process is orphaned.
- **ENOSPC**: There was no free space remaining on the device containing the file.
- **EPIPE**: An attempt is made to write to a pipe or FIFO that is not open for reading by any process. A `SIGPIPE` signal will also be sent to the process.

The `fclose()` function may fail if:

- **ENXIO**: A request was made of a non-existent device, or the request was beyond the limits of the device.

ATTRIBUTES
See `attributes(5)` for descriptions of the following attributes:
<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**SEE ALSO**

`close(2), exit(2), getrlimit(2), ulimit(2), fopen(3S), stdio(3S), attributes(5)`
NAME
fdatasync – synchronize a file’s data

SYNOPSIS
cc [ flag ... ] file ... -lposix4 [ library ... ]
#include <unistd.h>
int fdatasync(int fildes);

DESCRIPTION
fdatasync() forces all currently queued I/O operations associated with the file descriptor
fildes 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 fildes 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 con-
ditions defined for read(2) and write(2).

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Async-Signal-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
fcnt1(2), open(2), read(2), write(2), fsync(3C), aio_fsync(3R), attributes(5), fcntl(5)

NOTES
If fdatasync() fails, outstanding I/O operations are not guaranteed to have been com-
pleted.
NAME
fdetach – detach a name from a STREAMS-based file descriptor

SYNOPSIS
#include <stropts.h>

int fdetach(const char *path);

DESCRIPTION
The fdetach() function detaches a STREAMS-based file from the file to which it was
attached by a previous call to fattach(3C). The path argument points to the pathname of
the attached STREAMS file. The process must have appropriate privileges or be the owner
of the file. A successful call to fdetach() causes all pathnames that named the attached
STREAMS file to again name the file to which the STREAMS file was attached. All subse-
quent operations on path will operate on the underlying file and not on the STREAMS file.
All open file descriptions established while the STREAMS file was attached to the file
referenced by path, will still refer to the STREAMS file after the fdetach() has taken effect.
If there are no open file descriptors or other references to the STREAMS file, then a suc-
cessful call to fdetach() has the same effect as performing the last close(2) on the attached
file.

RETURN VALUES
Upon successful completion, fdetach() returns 0. Otherwise, it returns −1 and sets errno
to indicate the error.

ERRORS
The fdetach() function will fail if:
EACCES Search permission is denied on a component of the path prefix.
EPERM The effective user ID is not the owner of path and the process does not
have appropriate privileges.
ENOTDIR A component of the path prefix is not a directory.
ENOENT A component of path does not name an existing file or path is an empty
string.
EINVAL The path argument names a file that is not currently attached.
ENAMETOOLONG The size of a pathname exceeds PATH_MAX, or a pathname component
is longer than NAME_MAX while _POSIX_NO_TRUNC is in effect.
ELOOP Too many symbolic links were encountered in resolving path.

The fdetach() function may fail if:
ENAMETOOLONG Pathname resolution of a symbolic link produced an intermediate result
whose length exceeds PATH_MAX.

SEE ALSO
fdetach(1M), close(2), fattach(3C), streamio(7I)
STREAMS Programming Guide
NAME  fdopen – associate a stream with a file descriptor

SYNOPSIS  

```c
#include <stdio.h>
FILE *fdopen(int fildes, const char *mode);
```

DESCRIPTION  The `fdopen()` function associates a stream with a file descriptor `fildes`, whose value must be less than 255.

The `mode` argument is a character string having one of the following values:

- `r` or `rb`  open a file for reading
- `w` or `wb`  open a file for writing
- `a` or `ab`  open a file for writing at end of file
- `r+` or `rb+` or `r+b`  open a file for update (reading and writing)
- `w+` or `wb+` or `w+b`  open a file for update (reading and writing)
- `a+` or `ab+` or `a+b`  open a file for update (reading and writing) at end of file

The meaning of these flags is exactly as specified in `fopen(3S)`, except that modes beginning with `w` do not cause truncation of the file.

The mode of the stream must be allowed by the file access mode of the open file. The file position indicator associated with the new stream is set to the position indicated by the file offset associated with the file descriptor.

`fdopen()` will preserve the offset maximum previously set for the open file description corresponding to `fildes`.

The error and end-of-file indicators for the stream are cleared. The `fdopen()` function may cause the `st_atime` field of the underlying file to be marked for update.

RETURN VALUES  Upon successful completion, `fdopen()` returns a pointer to a stream. Otherwise, a null pointer is returned and `errno` is set to indicate the error.

`fdopen()` may fail and not set `errno` if there are no free `stdio` streams.

ERRORS  The `fdopen()` function may fail if:

- `EBADF`  The `fildes` argument is not a valid file descriptor.
- `EINVAL`  The `mode` argument is not a valid mode.
- `EMFILE`  `FOPEN_MAX` streams are currently open in the calling process.
- `EMFILE`  `STREAM_MAX` streams are currently open in the calling process.
- `ENOMEM`  Insufficient space to allocate a buffer.

USAGE  `STREAM_MAX` is the number of streams that one process can have open at one time. If defined, it has the same value as `FOPEN_MAX`.

File descriptors are obtained from calls like `open(2)`, `dup(2)`, `creat(2)` or `pipe(2)`, which open files but do not return streams. Streams are necessary input for almost all of the Section 3S library routines.
ATTTIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO

creat(2), dup(2), open(2), pipe(2), fclose(3S), fopen(3S), attributes(5)
NAME   ferror, feof, clearerr, fileno – stream status inquiries

SYNOPSIS   
#include <stdio.h>  
int ferror(FILE *stream);  
int feof(FILE *stream);  
void clearerr(FILE *stream);  
int fileno(FILE *stream);  

DESCRIPTION   ferror() returns non-zero when an error has previously occurred reading from or writing to the named stream (see intro(3)), otherwise zero.  
feof() returns non-zero when EOF has previously been detected reading the named input stream, otherwise zero.  
clearerr() resets the error indicator and EOF indicator to zero on the named stream.  
fileno() returns the integer file descriptor associated with the named stream; see open(2).  

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO   open(2), intro(3), fopen(3S), stdio(3S), attributes(5)
NAME    fflush – flush a stream

SYNOPSIS #include <stdio.h>

int fflush(FILE *stream);

DESCRIPTION If stream points to an output stream or an update stream in which the most recent operation was not input, fflush() causes any unwritten data for that stream to be written to the file, and the st_ctime and st_mtime fields of the underlying file are marked for update. If stream is a null pointer, fflush() performs this flushing action on all streams for which the behavior is defined above.

RETURN VALUES Upon successful completion, fflush() returns 0. Otherwise, it returns EOF and sets errno to indicate the error.

ERRORS The fflush() function will fail if:

EAGAIN The O_NONBLOCK flag is set for the file descriptor underlying stream and the process would be delayed in the write operation.

EBADF The file descriptor underlying stream is not valid.

EFBIG An attempt was made to write a file that exceeds the maximum file size or the process’ file size limit.

ENOSPC There was no free space remaining on the device containing the file.

EPIPE An attempt is made to write to a pipe or FIFO that is not open for reading by any process. A SIGPIPE signal will also be sent to the process.

The fflush() function may fail if:

ENXIO A request was made of a non-existent device, or the request was beyond the limits of the device.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
SEE ALSO  getrlimit(2), ulimit(2), attributes(5)
NAME  
ffs – find first set bit

SYNOPSIS  
#include <strings.h>
int ffs(const int i);

DESCRIPTION  
The ffs() function finds the first bit set (beginning with the least significant bit) and returns the index of that bit. Bits are numbered starting at one (the least significant bit).

RETURN VALUES  
The ffs() function returns the index of the first bit set. If i is 0, then ffs() returns 0.

ERRORS  
No errors are defined.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  
attributes(5)
NAME
fgetpos – get current file position information

SYNOPSIS
#include <stdio.h>
int fgetpos(FILE *stream, fpos_t *pos);

DESCRIPTION
The fgetpos() function stores the current value of the file position indicator for the stream
pointed to by stream in the object pointed to by pos. The value stored contains
unspecified information usable by fsetpos(3S) for repositioning the stream to its position
at the time of the call to fgetpos().

RETURN VALUES
Upon successful completion, fgetpos() returns 0. Otherwise, it returns a non-zero value
and sets errno to indicate the error.

ERRORS
The fgetpos() function may fail if:
EBADF    The file descriptor underlying stream is not valid.
ESPIPE   The file descriptor underlying stream is associated with a pipe, a FIFO, or
          a socket.
EOVERFLOW The current value of the file position cannot be represented correctly in
           an object of type fpos_t.

USAGE
The fgetpos() function has an explicit 64-bit equivalent. See interface64(5).

SEE ALSO
fopen(3S), fsetpos(3S), ftell(3S), rewind(3S), ungetc(3S), interface64(5)
NAME  fgetwc – get a wide-character code from a stream

SYNOPSIS  #include <stdio.h>
#include <wchar.h>

wint_t fgetwc(FILE *stream);

DESCRIPTION  The fgetwc() function obtains the next character (if present) from the input stream
pointed to by stream, converts that to the corresponding wide-character code and
advances the associated file position indicator for the stream (if defined).
If an error occurs, the resulting value of the file position indicator for the stream is
indeterminate.

The fgetwc() function may mark the st_atime field of the file associated with stream for
update. The st_atime field will be marked for update by the first successful execution of
fgetwc(), fgetc(3S), fgets(3S), fgetws(3S), fread(3S), fscanf(3S), getc(3S), getchar(3S),
gets(3S), or scanf(3S) using stream that returns data not supplied by a prior call to
ungetc(3S) or ungetwc(3S).

RETURN VALUES  Upon successful completion the fgetwc() function returns the wide-character code of the
character read from the input stream pointed to by stream converted to a type wint_t.
If the stream is at end-of-file, the end-of-file indicator for the stream is set and fgetwc()
returns WEOF.
If a read error occurs, the error indicator for the stream is set, fgetwc() returns WEOF and
sets errno to indicate the error.

ERRORS  The fgetwc() function will fail if data needs to be read and:

EAGAIN  The O_NONBLOCK flag is set for the file descriptor underlying
stream and the process would be delayed in the fgetwc() operation.

EBADF  The file descriptor underlying stream is not a valid file descriptor
open for reading.

EINTR  The read operation was terminated due to the receipt of a signal,
and no data was transferred.

EIO  A physical I/O error has occurred, or the process is in a back-
ground process group attempting to read from its controlling ter-
minal, and either the process is ignoring or blocking the SIGTTIN
signal or the process group is orphaned.

EOVERFLOW  The file is a regular file and an attempt was made to read at or
beyond the offset maximum associated with the corresponding
stream.
The `fgetwc()` function may fail if:

- **ENOMEM**: Insufficient storage space is available.
- **ENXIO**: A request was made of a non-existent device, or the request was outside the capabilities of the device.
- **EILSEQ**: The data obtained from the input stream does not form a valid character.

**Usage**
The `ferror(3S)` or `feof(3S)` functions must be used to distinguish between an error condition and an end-of-file condition.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe with exceptions</td>
</tr>
<tr>
<td>CSI</td>
<td>Enabled</td>
</tr>
</tbody>
</table>

**See Also**
`feof(3S), ferror(3S), fgetc(3S), fgets(3S), fgetws(3S), fopen(3S), fread(3S), fscanf(3S), getc(3S), getchar(3S), gets(3S), scanf(3S), setlocale(3C), ungetc(3S), ungetwc(3S), attributes(5)`
NAME filter – disable use of certain terminal capabilities

SYNOPSIS #include <curses.h>

void filter(void);

DESCRIPTION The filter() function changes how X/Open Curses initializes terminal capabilities that assume the terminal has more than one line. After a call to filter(), the initscr(3XC) or newterm(3XC) functions also:

- disable use of clear, cud, cud1, cup, cuu1 and vpa
- set home string to the value of cr
- set lines to 1

RETURN VALUES The filter() function does not return a value.

ERRORS None.

SEE ALSO initscr(3XC), newterm(3XC)
The `floating_to_decimal()` functions convert the floating-point value at `*px` into a decimal record at `*pd`, observing the modes specified in `*pm` and setting exceptions in `*ps`. If there are no IEEE exceptions, `*ps` will be zero.

If `*px` is zero, infinity, or NaN, then only `pd->sign` and `pd->fpclass` are set. Otherwise `pd->exponent` and `pd->ds` are also set so that

\[(pd->sign)*(pd->ds)*10^{pd->exponent}\]

is a correctly rounded approximation to `*px`. `pd->ds` 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.

`pd->ds` is correctly rounded according to the IEEE rounding modes in `pm->rd`. `*ps` has `fp_inexact` set if the result was inexact, and has `fp_overflow` set if the string result does not fit in `pd->ds` because of the limitation `DECIMAL_STRING_LENGTH`.

If `pm->df == floating_form`, then `pd->ds` always contains `pm->ndigits` significant digits. Thus if `*px == 12.34` and `pm->ndigits == 8`, then `pd->ds` will contain 12340000 and `pd->exponent` will contain \(-6\).

If `pm->df == fixed_form` and `pm->ndigits >= 0`, then `pd->ds` always contains `pm->ndigits` 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 `pd->ndigits`; if that number >= `DECIMAL_STRING_LENGTH`, then `ds` is undefined. `pd->exponent` always gets \(-pm->ndigits\). Thus if `*px == 12.34` and `pm->ndigits == 1`, then `pd->ds` gets 123, `pd->exponent` gets \(-1\), and `pd->ndigits` gets 3.

If `pm->df == fixed_form` and `pm->ndigits < 0`, then `pd->ds` always contains \(-pm->ndigits\) trailing zeros; in other words, rounding occurs \(-pm->ndigits\) to the left of the decimal point, but the digits rounded away are retained as zeros. The total number of digits required is in `pd->ndigits`. `pd->exponent` always gets 0. Thus if `*px == 12.34` and `pm->ndigits == -1`, then `pd->ds` gets 10, `pd->exponent` gets 0, and `pd->ndigits` gets 2.
pd->more is not used.

econvert(3), fconvert(3), gconvert(3), printf(3S), and sprintf(3S) all use
double_to_decimal().

ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO econvert(3), fconvert(3), gconvert(3), printf(3S), sprintf(3S), attributes(5)
NAME  
flock – apply or remove an advisory lock on an open file

SYNOPSIS  
/usr/uucb/cc [ flag ...] file ...
#include <sys/file.h>
int flock(fd, operation)
int fd, operation;

DESCRIPTION  
flock() applies or removes an advisory lock on the file associated with the file descriptor fd. The compatibility version of flock() has been implemented on top of fcntl(2) locking. It does not provide complete binary compatibility.
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).
The locking mechanism allows two types of locks: shared locks and exclusive locks.
More than one process may hold a shared lock for a file at any given time, but multiple exclusive, or both shared and exclusive, locks may not exist simultaneously on a file.
A lock is applied by specifying an operation parameter LOCK_SH for a shared lock or LOCK_EX for an exclusive lock. The operation parameter may be ORed with LOCK_NB to make the operation non-blocking. To unlock an existing lock, the operation should be LOCK_UN.
Read permission is required on a file to obtain a shared lock, and write permission is required to obtain an exclusive lock. Locking a segment that is already locked by the calling process causes the old lock type to be removed and the new lock type to take effect.
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 operation, then this will not happen; instead, the call will fail and the error EWOULDBLOCK will be returned.

RETURN VALUES  
flock() returns:
0 on success.
−1 on failure and sets errno to indicate the error.

ERRORS  
EBADF The argument fd is an invalid descriptor.
EINVAL operation is not a valid argument.
EOPNOTSUPP The argument fd refers to an object other than a file.
EWOULDBLOCK The file is locked and the LOCK_NB option was specified.

SEE ALSO  
lockd(1M), chmod(2), close(2), dup(2), exec(2), fcntl(2), fork(2), open(2), lockf(3C)

NOTES  
Use of these interfaces should be restricted to only applications written on BSD platforms. Use of these interfaces with any of the system libraries or in multi-thread applications is unsupported.
Locks are on files, not file descriptors. That is, file descriptors duplicated through `dup(2)` or `fork(2)` do not result in multiple instances of a lock, but rather multiple references to a single lock. If a process holding a lock on a file forks and the child explicitly unlocks the file, the parent will lose its lock. Locks are not inherited by a child process.

Processes blocked awaiting a lock may be awakened by signals. Mandatory locking may occur, depending on the mode bits of the file. See `chmod(2)`.

Locks obtained through the `flock()` mechanism under SunOS 4.1 were known only within the system on which they were placed. This is no longer true.
NAME  flockfile, funlockfile, ftrylockfile – acquire and release stream lock

SYNOPSIS  
#include <stdio.h>

void flockfile(FILE *stream);
void funlockfile(FILE *stream);
int ftrylockfile(FILE *stream);

DESCRIPTION  The flockfile() function acquires an internal lock of a stream stream. If the lock is already acquired by another thread, the thread calling flockfile() is suspended until it can acquire the lock. In the case that the stream lock is available, flockfile() not only acquires the lock, but keeps track of the number of times it is being called by the current thread. This implies that the stream lock can be acquired more than once by the same thread.

The funlockfile() function releases the lock being held by the current thread. In the case of recursive locking, this function must be called the same number of times flockfile() was called. After the number of funlockfile() calls is equal to the number of flockfile() calls, the stream lock is available for other threads to acquire.

The ftrylockfile() function acquires an internal lock of a stream stream, only if that object is available. In essence ftrylockfile() is a non-blocking version of flockfile().

RETURN VALUES  The ftrylockfile() function returns 0 on success and non-zero to indicate a lock cannot be acquired.

EXAMPLES  The following example prints everything out together, blocking other threads that might want to write to the same file between calls to fprintf(3S):

FILE iop;
flockfile(iop);
fprintf(iop, "hello ");
fprintf(iop, "world0);
putc(iop, 'a');
funlockfile(iop);

An unlocked interface is available in case performance is an issue. For example:

flockfile(iop);
while (!feof(iop)) {
    *c++ = getc_unlocked(iop);
}
funlockfile(iop);

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

3S-528 SunOS 5.6 modified 30 Dec 1996
SEE ALSO intro(3), ferror(3S), fprintf(3S), getc(3S), putc(3S), stdio(3S), ungetc(3S), attributes(5), standards(5)

NOTES The interfaces on this page are as specified in IEEE Std 1003.1c. See standards(5).
NAME
floor – floor function

SYNOPSIS
c
cc [ flag ... ] file ... -lm [ library ... ]
#include <math.h>

double floor(double x);

DESCRIPTION
The floor() function computes the largest integral value not greater than x.

RETURN VALUES
Upon successful completion, floor() returns the largest integral value not greater than x, expressed as a double.
If x is NaN, NaN is returned.
If x is ±Inf or ±0, x is returned.

ERRORS
No errors will occur.

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

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
ceil(3M), isnan(3M), attributes(5)
<table>
<thead>
<tr>
<th>NAME</th>
<th>flushinp – discard type-ahead characters</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYNOPSIS</td>
<td>#include &lt;curses.h&gt;</td>
</tr>
<tr>
<td></td>
<td>int flushinp(void);</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>The <code>flushinp()</code> function discards all type-ahead characters (characters typed by the user, but not yet processed by X/Open Curses).</td>
</tr>
<tr>
<td>RETURN VALUES</td>
<td>The <code>flushinp()</code> function always returns OK.</td>
</tr>
<tr>
<td>ERRORS</td>
<td>None.</td>
</tr>
</tbody>
</table>
NAME  

fmod – floating-point remainder value function

SYNOPSIS  

cc [ flag ... ] file ... -lm [ library ... ]
#include <math.h>
double fmod(double x, double y);

DESCRIPTION  

The fmod() function returns the floating-point remainder of the division of x by y.

RETURN VALUES  

The fmod() function returns the value x – i * y, for some integer i such that, if y is non-zero, the result has the same sign as x and magnitude less than the magnitude of y.
If x or y is NaN, NaN is returned.
If y is 0, NaN is returned and errno is set to EDOM.
If x is ±Inf, NaN is returned.
If y is non-zero, fmod(±0, y) returns the value of x. If x is not ±Inf, fmod(x, ±Inf) returns the value of x.

ERRORS  

The fmod() function may fail if:
EDOM  y is 0.
No other errors will occur.

USAGE  

Portable applications should not call fmod() with y equal to 0, because the result is implementation-dependent. The application should verify y is non-zero before calling fmod().
An application wishing to check for error situations should set errno to 0 before calling fmod(). If errno is non-zero on return, or the return value is NaN, an error has occurred.

ATTRIBUTES  

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  

isnan(3M), attributes(5)
fmtmsg – display a message on stderr or system console

#include <fmtmsg.h>

int fmtmsg(long classification, const char *label, int severity, const char *text,
            const char *action, const char *tag);

DESCRIPTION
Based on a message's classification component, fmtmsg() writes a formatted message to stderr, to the console, or to both.

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.

A formatted message consists of up to five standard components as defined below. The component, classification, 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.

classification
Contains identifiers from the following groups of major classifications and subclassifications. Any one identifier from a subclass may be used in combination by ORing the values together with a single identifier from a different subclass. Two or more identifiers from the same subclass should not be used together, with the exception of identifiers from the display subclass. (Both display subclass identifiers may be used so that messages can be displayed to both stderr and the system console).

- "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.

label
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 label identifies the package in which the application resides as well as the program or application name. For example, the label UX:cat indicates the UNIX System V package and the cat application.
fmtmsg (3C) C Library Functions

**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:

```text
export MSGVERB=keyword[keyword[:...]]
```

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.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(no severity is used)</td>
</tr>
<tr>
<td>1</td>
<td>HALT</td>
</tr>
<tr>
<td>2</td>
<td>ERROR</td>
</tr>
<tr>
<td>3</td>
<td>WARNING</td>
</tr>
<tr>
<td>4</td>
<td>INFO</td>
</tr>
</tbody>
</table>

`SEV_LEVEL` can be set as follows:

```sh
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.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Null-Value</th>
<th>Identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>label</td>
<td>char*</td>
<td>(char*) NULL</td>
<td>MM_NULLLBL</td>
</tr>
<tr>
<td>severity</td>
<td>int</td>
<td>0</td>
<td>MM_NULLSEV</td>
</tr>
<tr>
<td>class</td>
<td>long</td>
<td>0L</td>
<td>MM_NULLMC</td>
</tr>
<tr>
<td>text</td>
<td>char*</td>
<td>(char*) NULL</td>
<td>MM_NULLTXT</td>
</tr>
<tr>
<td>action</td>
<td>char*</td>
<td>(char*) NULL</td>
<td>MM_NULLACT</td>
</tr>
<tr>
<td>tag</td>
<td>char*</td>
<td>(char*) NULL</td>
<td>MM_NULLTAG</td>
</tr>
</tbody>
</table>

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()`:

```c
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:

```bash
MSGVERB=severity:text:action
```

and the Example 1 is used, `fmtmsg()` produces:

```
ERROR: invalid syntax
TO FIX: refer to manual
```
Example 3:
When the environment variable SEV_LEVEL is set as follows:

```
SEV_LEVEL=note,5,NOTE
```

the following call to `fmtmsg()`:

```
fmtmsg(MM_UTIL | MM_PRINT, "UX:cat", 5, "invalid syntax", "refer to manual", "UX:cat:001")
```

produces:

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

**ATTRIBUTES**

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

**SEE ALSO**

`fmtmsg(1)`, `addseverity(3C)`, `gettext(3C)`, `printf(3S)`, `attributes(5)`
NAME
fn_attr_bind – bind a reference to a name and associate attributes with named object

SYNOPSIS
#include <xfn/xfn.h>

int fn_attr_bind(FN_ctx_t *ctx, const FN_composite_name_t *name,
                 const FN_ref_t *ref, const FN_attrset_t *attrs,
                 unsigned int exclusive,
                 FN_status_t *status);

DESCRIPTION
This operation binds the supplied reference ref to the supplied composite name name relative to ctx, and associates the attributes specified in attrs with the named object. The binding is made in the target context, that is, that context named by all but the terminal atomic part of name. The operation binds the terminal atomic name to the supplied reference in the target context. The target context must already exist.

The value of exclusive determines what happens if the terminal atomic part of the name is already bound in the target context. If exclusive is nonzero and name is already bound, the operation fails. If exclusive is 0, the new binding replaces any existing binding, and, if attrs is not NULL, attrs replaces any existing attributes associated with the named object. If attrs is NULL and exclusive is 0, any existing attributes associated with the named object are left unchanged.

RETURN VALUES
fn_attr_bind() returns 1 upon success, 0 upon failure.

ERRORS
fn_attr_bind() sets status as described in FN_status_t(3N) and xfn_status_codes(3N). Of special relevance for this operation is the following status code:

FN_E_NAME_IN_USE
The supplied name is already in use.

USAGE
The value of ref cannot be NULL. If the intent is to reserve a name using fn_attr_bind(), a reference containing no address should be supplied. This reference may be name service-specific or it may be the conventional NULL reference.

If multiple sources are updating a reference or attributes associated with a named object, they must synchronize amongst each other when adding, modifying, or removing from the address list of a bound reference, or manipulating attributes associated with the named object.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
FN_composite_name_t(3N), FN_ctx_t(3N), FN_ref_t(3N), FN_status_t(3N),
fn_ctx_bind(3N), fn_ctx_lookup(3N), fn_ctx_unbind(3N), xfn_attributes(3N),
xfn_status_codes(3N), attributes(5)
NAME
fn_attr_create_subcontext – create a subcontext in a context and associate attributes with newly created context

SYNOPSIS
#include <xfn/xfn.h>
FN_ref_t *fn_attr_create_subcontext(FN_ctx_t *ctx,
    const FN_composite_name_t *name, const FN_attrset_t *attrs,
    FN_status_t *status);

DESCRIPTION
This operation creates a new XFN context of the same type as the target context, that is, that context named by all but the terminal atomic component of name, and binds it to the supplied composite name. In addition, attributes given in attrs are associated with the newly created context.

The target context must already exist. The new context is created and bound in the target context using the terminal atomic name in name. The operation returns a reference to the newly created context.

RETURN VALUES
fn_attr_create_subcontext() returns a reference to the newly created context; if the operation fails, it returns a NULL pointer.

ERRORS
fn_attr_create_subcontext() sets status as described in FN_status_t(3N) and xfn_status_codes(3N). Of special relevance for this operation is the following status code:
FN_E_NAME_IN_USE
The terminal atomic name already exists in the target context.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
FN_composite_name_t(3N), FN_ctx_t(3N), FN_ref_t(3N), FN_status_t(3N), fn_attr_bind(3N), fn_ctx_bind(3N), fn_ctx_create_subcontext(3N), fn_ctx_destroy_subcontext(3N), fn_ctx_lookup(3N), xfn_attributes(3N), xfn_status_codes(3N), attributes(5)

modified 22 Nov 1996
SunOS 5.6
3N-539
NAME

fn_attr_ext_search, FN_ext_searchlist_t, fn_ext_searchlist_next, fn_ext_searchlist_destroy

Name search for names in the specified context(s) whose attributes satisfy the filter

SYNOPSIS

#include <xfn/xfn.h>

FN_ext_searchlist_t *fn_attr_ext_search(FN_ctx_t *ctx, 
    const FN_composite_name_t *name, const FN_search_control_t *control, 
    const FN_search_filter_t *filter, FN_status_t *status);

FN_composite_name_t *fn_ext_searchlist_next(FN_ext_searchlist_t *esl, 
    FN_ref_t **returned_ref, FN_attrset_t **returned_attrs, FN_status_t *status);

void fn_ext_searchlist_destroy(FN_ext_searchlist_t *esl);

DESCRIPTION

This set of operations is used to list names of objects whose attributes satisfy the filter expression. The references to which these names are bound and specified attributes and their values may also be returned.

control encapsulates the option settings for the search. These options are:

- the scope of the search
- whether XFN links are followed
- a limit on the number of names returned
- whether references and specific attributes associated with the named objects that satisfy the filter are returned

The scope of the search is one of:

- the object named name relative to the context ctx
- the context named name relative to the context ctx
- the context named name relative to the context ctx, and its subcontexts
  or
- the context named name relative to the context ctx, and a context implementation-defined set of subcontexts

If the value of control is 0, default control option settings are used. The default settings are:

- scope is search named context
- links are not followed
- all names of objects that satisfy the filter are returned
- references and attributes are not returned

The FN_search_control_t type is described in FN_search_control_t(3N).

The filter expression filter in fn_attr_ext_search() is evaluated against the attributes of the objects bound in the scope of the search. The filter evaluates to either TRUE or FALSE. The names and, optionally, the references and attributes of objects whose attributes satisfy the filter are enumerated. If the value of filter is 0, all names within the search scope are enumerated. The FN_search_filter_t type is described in FN_search_filter_t(3N).
The call to `fn_attr_ext_search()` initiates the search process. It returns a handle to an `FN_ext_searchlist_t` object that is used to enumerate the names of the objects that satisfy the filter.

The operation `fn_ext_searchlist_next()` returns the next name in the enumeration identified by `esl`; it also updates `esl` to indicate the state of the enumeration. If the reference to which the name is bound was requested, it is returned in `returned_ref`. Requested attributes associated with the name are returned in `returned_attrs`; each attribute consists of an attribute identifier, syntax, and value(s). Successive calls to `fn_ext_searchlist_next()` using `esl` return successive names and, optionally, their references and attributes, in the enumeration; these calls further update the state of the enumeration.

The names that are returned are composite names, to be resolved relative to the starting context for the search. This starting context is the context named `name` relative to `ctx` unless the scope of the search is only the named object. If the scope of the search is only the named object, the terminal atomic name in `name` is returned.

`fn_ext_searchlist_destroy()` releases resources used during the enumeration. This may be invoked at any time to terminate the enumeration.

**RETURN VALUES**

`fn_attr_ext_search()` returns a pointer to an `FN_ext_searchlist_t` object if the search is successfully initiated; it returns a NULL pointer if the search cannot be initiated or if no named object with attributes whose values satisfy the filter expression is found.

`fn_ext_searchlist_next()` returns a pointer to an `FN_composite_name_t` object (see `FN_composite_name_t(3N)`) that is the next name in the enumeration; it returns a NULL pointer if no more names can be returned. If `returned_attrs` is a NULL pointer, no attributes are returned; otherwise, `returned_attrs` contains the attributes associated with the named object, as specified in the control parameter to `fn_attr_ext_search()`. If `returned_ref` is a NULL pointer, no reference is returned; otherwise, if control specified the return of the reference of the named object, that reference is returned in `returned_ref`.

In the case of a failure, these operations return in the `status` argument a code indicating the nature of the failure.

**ERRORS**

If successful, `fn_attr_ext_search()` returns a pointer to an `FN_ext_searchlist_t` object and sets `status` to `FN_SUCCESS`.

`fn_attr_ext_search()` returns a NULL pointer when no more names can be returned. `status` is set in the following way:

- **FN_SUCCESS**
  A named object could not be found whose attributes satisfied the filter expression.

- **FN_E_NOT_A_CONTEXT**
  The object named for the start of the search was not a context and the search scope was the given context or the given context and its sub-contexts.
FN_E_SEARCH_INVALID_FILTER
The filter could not be evaluated TRUE or FALSE, or there was some other problem with the filter.

FN_E_SEARCH_INVALID_OPTION
A supplied search control option could not be supported.

FN_E_SEARCH_INVALID_OP
An operator in the filter expression is not supported or, if the operator is an extended operator, the number of types of arguments supplied does not match the signature of the operation.

FN_E_ATTR_NO_PERMISSION
The caller did not have permission to read one or more of the attributes specified in the filter.

FN_E_INVALID_ATTR_VALUE
A value type in the filter did not match the syntax of the attribute against which it was being evaluated.

Other status codes are possible as described in FN_status_t(3N) and xfn_status_codes(3N).

Each successful call to fn_ext_searchlist_next() returns a name and, optionally, its reference in returned_ref and requested attributes in returned_attrs. status is set in the following way:

FN_SUCCESS
All requested attributes were returned successfully with the name.

FN_E_ATTR_NO_PERMISSION
The caller did not have permission to read one or more of the requested attributes.

FN_E_INVALID_ATTR_IDENTIFIER
A requested attribute identifier was not in a format acceptable to the naming system, or its contents were not valid for the format specified.

FN_E_NO_SUCH_ATTRIBUTE
The named object did not have one of the requested attributes.

FN_E_INSUFFICIENT_RESOURCES
Insufficient resources are available to return all the requested attributes and their values.

FN_E_ATTR_NO_PERMISSION
FN_E_INVALID_ATTR_IDENTIFIER
FN_E_NO_SUCH_ATTRIBUTE
FN_E_INSUFFICIENT_RESOURCES
These indicate that some of the requested attributes may have been returned in returned_attrs but one or more of them could not be returned. Use fn_attr_get(3N) or fn_attr_multi_get(3N) to discover why these attributes could not be returned.
If `fn_ext_searchlist_next()` returns a name, it can be called again to get the next name in the enumeration.

`fn_ext_searchlist_next()` returns a NULL pointer if no more names can be returned. `status` is set in the following way:

- **FN_SUCCESS**
  The search has completed successfully.

- **FN_E_PARTIAL_RESULT**
  The enumeration is not yet complete but cannot be continued.

- **FN_E_ATTR_NO_PERMISSION**
  The caller did not have permission to read one or more of the attributes specified in the filter.

- **FN_E_INVALID_ENUM_HANDLE**
  The supplied enumeration handle was not valid. Possible reasons could be that the handle was from another enumeration, or the context being enumerated no longer accepts the handle (due to such events as handle expiration or updates to the context).

Other status codes are possible as described in `FN_status_t(3N)` and `xfn_status_codes(3N)`.

**USAGE**

The search performed by `fn_attr_ext_search()` is not ordered in any way, including the traversal of subcontexts. The names enumerated using `fn_ext_searchlist_next()` are not ordered in any way. Furthermore, there is no guarantee that any two series of enumerations with the same arguments to `fn_attr_ext_search()` will return the names in the same order.

XFN links encountered during the resolution of `name` are followed, regardless of the follow links control setting, and the search starts at the final named object or context.

If `control` specifies that the search should follow links, XFN link names encountered during the search are followed and the terminal named object is searched. If the terminal named object is bound to a context and the scope of the search includes subcontexts, that context and its subcontexts are also searched. For example, if `aname` is bound to an XFN link, `lname`, in a context within the scope of the search, and `aname` is returned by `fn_ext_searchlist_next()`, this means that the object identified by `lname` satisfied the filter expression. `aname` is returned instead of `lname` because `aname` can always be named relative to the starting context for the search.

If `control` specifies that the search should not follow links, the attributes associated with the names of XFN links are searched. For example, if `aname` is bound to an XFN link, `lname`, in a context within the scope of the search, and `aname` is returned by `fn_ext_searchlist_next()`, this means that the object identified by `aname` satisfied the filter expression.

When following XFN links, `fn_attr_ext_search()` may search contexts outside of `scope`. In addition, if the link name's terminal atomic name is bound in a context within `scope`, the operation may return the same object more than once.

modified 22 Nov 1996

SunOS 5.6

3N-543
XFN does not specify how control affects the following of native naming system links during the search.

**EXAMPLES**

The following code fragment illustrates how the `fn_attr_ext_search()` operation may be used. The code consists of three parts: preparing the arguments for the search, performing the search, and cleaning up.

The first part involves getting the name of the context to start the search and constructing the search filter that named objects in the context must satisfy. This is done in the declarations part of the code and by the routine `get_search_query`. See `FN_search_filter_t(3N)` for the description of `sfilter` and the filter creation operation.

The next part involves doing the search and enumerating the results of the search. This is done by first getting a context handle to the Initial Context, and then passing that handle along with the name of the target context and search filter to `fn_attr_ext_search()`. This particular call to `fn_attr_ext_search()` uses the default search control options (by passing in 0 as the `control` argument). This means that the search will be performed in the context named by `target_name` and that no reference or attributes will be returned. In addition, any XFN links encountered will not be followed and all named objects that satisfy the search filter will be returned (that is, no limit). If successful, `fn_attr_ext_search()` returns `esl`, a handle for enumerating the results of the search. The results of the search are enumerated using calls to `fn_ext_searchlist_next()`, which returns the name of the object. (The arguments `returned_ref` and `returned attrs` to `fn_ext_searchlist_next()` are 0 because the default search control used in `fn_attr_ext_search()` did not request them to be returned.)

The last part of the code involves cleaning up the resources used during the search and enumeration. The call to `fn_ext_searchlist_destroy()` releases resources reserved for this enumeration. The other calls release the context handle, name, filter, and status objects created earlier.

```c
/* Declarations */
FN_ctx_t *ctx;
FN_ext_searchlist_t *esl;
FN_composite_name_t *name;
FN_status_t *status = fn_status_create();
FN_composite_name_t *target_name = get_name_from_user_input();
FN_search_filter_t *sfilter = get_search_query();

/* Get context handle to Initial Context */
ctx = fn_ctx_handle_from_initial(status);

/* error checking on 'status' */
```
/* Initiate search */
if ((esl = fn_attr_ext_search(ctx, target_name,
    /* default controls */ 0, sfilter, status)) == 0) {
    /* report 'status', cleanup, and exit */
}

/* Enumerate names requested */
while (name = fn_ext_searchlist_next(esl, 0, 0, status)) {
    /* do something with 'name' */
    fn_composite_destroy(name);
}

/* check 'status' for reason for end of enumeration */

/* Clean up */
fn_ext_searchlist_destroy(esl);
fn_search_filter_destroy(sfilter);
fn_ctx_handle_destroy(ctx);
fn_composite_name_destroy(target_name);
fn_status_destroy(status);

/* Procedure for constructing the filter object for search:
   "age" attribute is greater than or equal to 17 AND
   less than or equal to 25
   AND the "student" attribute is present.
   */

FN_search_filter_t *
get_search_query()
{
    extern FN_attribute_t *attr_age;
    extern FN_attribute_t *attr_student;
    FN_search_filter_t *sfilter;
    unsigned int filter_status;

    sfilter = fn_search_filter_create(
        &filter_status,
        "(%a >= 17) and (%a <= 25) and %a",
        attr_age, attr_age, attr_student);

    /* error checking on 'filter_status' */

    return (sfilter);
}
ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO

FN_attrset_t(3N), FN_composite_name_t(3N), FN_ctx_t(3N), FN_ref_t(3N),
FN_search_control_t(3N), FN_search_filter_t(3N), FN_status_t(3N), fn_attr_get(3N),
fn_attr_multi_get(3N), xfn_status_codes(3N), attributes(5)
NAME  
fn_attr_get – return specified attribute associated with name

SYNOPSIS  
cc [ flag ...] file ... -lxfn [ library ...]
#include <xfn/xfn.h>
FN_attribute_t *fn_attr_get(FN_ctx_t *ctx, const FN_composite_name_t *name,
        const FN_identifier_t *attribute_id, FN_status_t *status);

DESCRIPTION  
This operation returns the identifier, syntax and values of a specified attribute for the
object named name relative to ctx. If name is empty, the attribute associated with ctx is
returned.

RETURN VALUES  
fn_attr_get returns a pointer to an FN_attribute_t object if the operation succeeds; it
returns a NULL pointer (0) if the operation fails.

ERRORS  
fn_attr_get() sets status as described in FN_status_t(3N) and xfn_status_codes(3N).

USAGE  
fn_attr_get_values() and its related operations are used for getting individual values of
an attribute. They should be used if the combined size of all the values are expected to be
too large to be returned in a single invocation of fn_attr_get().

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  
FN_attribute_t(3N), FN_composite_name_t(3N), FN_ctx_t(3N), FN_identifier_t(3N),
FN_status_t(3N), fn_attr_get_values(3N), xfn(3N), xfn_attributes(3N),
xfn_status_codes(3N), attributes(5)

NOTES  
The implementation of XFN in this Solaris release is based on the X/Open preliminary
specification. It is likely that there will be minor changes to these interfaces to reflect
changes in the final version of this specification. The next minor release of Solaris will
offer binary compatibility for applications developed using the current interfaces. As the
interfaces evolve toward standardization, it is possible that future releases of Solaris will
require minor source code changes to applications that have been developed against the
preliminary specification.

modified 13 Dec 1996          SunOS 5.6             3N-547
fn_attr_get_ids (3N) Network Functions

NAME fn_attr_get_ids − get a list of the identifiers of all attributes associated with named object

SYNOPSIS

cc [ flag . . . ] file . . . −lxfn [ library . . . ]

#include <xfn/xfn.h>

FN_attrset_t *fn_attr_get_ids(FN_ctx_t *ctx, const FN_composite_name_t *name, FN_status_t *status);

DESCRIPTION

This operation returns a list of the attribute identifiers of all attributes associated with the object named by name relative to the context ctx. If name is empty, the attribute identifiers associated with ctx are returned.

RETURN VALUES

This operation returns a pointer to an object of type FN_attrset_t; if the operation fails, a NULL pointer (0) is returned.

ERRORS

This operation sets status as described in FN_status_t(3N) and xfn_status_codes(3N).

USAGE

The attributes in the returned set do not contain the syntax or values of the attributes, only their identifiers.

ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO

FN_attribute_t(3N), FN_attrset_t(3N), FN_composite_name_t(3N), FN_ctx_t(3N), FN_status_t(3N), fn_attr_get(3N), fn_attr_multi_get(3N), xfn(3N), xfn_attributes(3N), xfn_status_codes(3N), attributes(5)

NOTES

The implementation of XFN in this Solaris release is based on the X/Open preliminary specification. It is likely that there will be minor changes to these interfaces to reflect changes in the final version of this specification. The next minor release of Solaris will offer binary compatibility for applications developed using the current interfaces. As the interfaces evolve toward standardization, it is possible that future releases of Solaris will require minor source code changes to applications that have been developed against the preliminary specification.
NAME | fn_attr_get_values, FN_valuelist_t, fn_valuelist_next, fn_valuelist_destroy – return values of an attribute

SYNOPSIS | cc [ flag ... ] file ... -lxfn [ library ... ]
#include <xfn/xfn.h>
FN_valuelist_t *fn_attr_get_values(FN_ctx_t *ctx, const FN_composite_name_t *name, const FN_identifier_t *attribute_id, FN_status_t *status);
FN_attrvalue_t *fn_valuelist_next(FN_valuelist_t *vl, FN_identifier_t **attr_syntax, FN_status_t *status);
void fn_valuelist_destroy(FN_valuelist_t *vl, FN_status_t *status);

DESCRIPTION | This set of operations is used to obtain the values of a single attribute, identified by attribute_id, associated with the object named name, resolved in the context ctx. If name is empty, the attribute values associated with ctx are obtained.
The operation fn_attr_get_values() initiates the enumeration process. It returns a handle to an FN_valuelist_t object that can be used to enumerate the values of the specified attribute.
The operation fn_valuelist_next() returns a new FN_attrvalue_t object containing the next value in the attribute and may be called multiple times until all values are retrieved. The syntax of the attribute is returned in attr_syntax.
The operation fn_valuelist_destroy() is used to release the resources used during the enumeration. This may be invoked before the enumeration has completed to terminate the enumeration.
These operations work in a fashion similar to the fn_ctx_list_names() operations.

RETURN VALUES | fn_attr_get_values() returns a pointer to an FN_valuelist_t object if the enumeration process is successfully initiated; it returns a NULL pointer if the process failed.
fn_valuelist_next() returns a NULL pointer if no more attribute values can be returned. In the case of a failure, these operations set status to indicate the nature of the failure.

ERRORS | Each successful call to fn_valuelist_next() returns an attribute value. status is set to FN_SUCCESS.
When fn_valuelist_next() returns a NULL pointer, it indicates that no more values can be returned. status is set in the following way:
FN_SUCCESS
The enumeration has completed successfully.

modified 13 Dec 1996
FN_E_INVALID_ENUM_HANDLE
The given enumeration handle is not valid. Possible reasons could be that the handle was from another enumeration, or the context being enumerated no longer accepts the handle (due to such events as handle expiration or updates to the context).

FN_E_PARTIAL_RESULT
The enumeration is not yet complete but cannot be continued.

In addition to these status codes, other status codes are also possible in calls to these operations. In such cases, status is set as described in FN_status_t(3N) and xfn_status_codes(3N).

USAGE
This interface should be used instead of fn_attr_get() if the combined size of all the values is expected to be too large to be returned by fn_attr_get().

There may be a relationship between the ctx argument supplied to fn_attr_get_values() and the FN_valuelist_t object it returns. For example, some implementations may store the context handle ctx within the FN_valuelist_t object for subsequent fn_valuelist_next() calls. In general, an fn_ctx_handle_destroy(3N) should not be invoked on ctx until the enumeration has terminated.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
FN_attribute_t(3N), FN_attrvalue_t(3N), FN_composite_name_t(3N), FN_ctx_t(3N), FN_identifier_t(3N), FN_status_t(3N), fn_attr_get(3N), fn_ctx_handle_destroy(3N), fn_ctx_list_names(3N), xfn(3N), xfn_attributes(3N), xfn_status_codes(3N), attributes(5)

NOTES
The implementation of XFN in this Solaris release is based on the X/Open preliminary specification. It is likely that there will be minor changes to these interfaces to reflect changes in the final version of this specification. The next minor release of Solaris will offer binary compatibility for applications developed using the current interfaces. As the interfaces evolve toward standardization, it is possible that future releases of Solaris will require minor source code changes to applications that have been developed against the preliminary specification.
NAME

FN_attribute_t, fn_attribute_create, fn_attribute_destroy, fn_attribute_copy,
fn_attribute_assign, fn_attribute_identifier, fn_attribute_syntax, fn_attribute_valuecount,
fn_attribute_first, fn_attribute_next, fn_attribute_add, fn_attribute_remove – an XFN attribute

SYNOPSIS

cc [ flag ... ] file ... -lxfn [ library ... ]
#include <xfn/xfn.h>
FN_attribute_t *fn_attribute_create(const FN_identi®er_t *attribute_id,
const FN_identi®er_t *attribute_syntax);
void fn_attribute_destroy(FN_attribute_t *attr);
FN_attribute_t *fn_attribute_copy(const FN_attribute_t *attr);
FN_attribute_t *fn_attribute_assign(FN_attribute_t *dst,
const FN_attribute_t *src);
const FN_identi®er_t *fn_attribute_identifier(const FN_attribute_t *attr);
const FN_identi®er_t *fn_attribute_syntax(const FN_attribute_t *attr);
unsigned int fn_attribute_valuecount(const FN_attribute_t *attr);
const FN_attrvalue_t *fn_attribute_first(const FN_attribute_t *attr,
void **iter_pos);
const FN_attrvalue_t *fn_attribute_next(const FN_attribute_t *attr,
void **iter_pos);
int fn_attribute_add(FN_attribute_t *attr, const FN_attrvalue_t *attribute_value,
unsigned int exclusive);
int fn_attribute_remove(FN_attribute_t *attr, const FN_attrvalue_t *attribute_value);

DESCRIPTION

An attribute has an attribute identifier, a syntax, and a set of distinct values. Each value
is a sequence of octets. The operations associated with objects of type FN_attribute_t
allow the construction, destruction, and manipulation of an attribute and its value set.

The attribute identifier and its syntax are specified using an FN_identi®er_t.
fn_attribute_create() creates a new attribute object with the given identifier and syntax,
and an empty set of values. fn_attribute_destroy() releases the storage associated with
attr. fn_attribute_copy() returns a copy of the object pointed to by attr.
fn_attribute_assign() makes a copy of the attribute object pointed to by src and assigns it
to dst, releasing any old contents of dst. A pointer to the same object as dst is returned.

fn_attribute_identifier() returns the attribute identifier of attr. fn_attribute_syntax()
returns the attribute syntax of attr. fn_attribute_valuecount() returns the number of
attribute values in attr.

fn_attribute_first() and fn_attribute_next() are used to enumerate the values of an attribute.
Enumeration of the values of an attribute may return the values in any order.
fn_attribute_first() returns an attribute value from attr and sets the iteration marker
iter_pos. Subsequent calls to fn_attribute_next() returns the next attribute value.
identified by \textit{iter_pos} and advances \textit{iter_pos}. Adding or removing values from an attribute invalidates any iteration markers that the caller holds.

\textbf{fn_attribute_add()} adds a new value \textit{attribute_value} to \textit{attr}. The operation succeeds (but no change is made) if \textit{attribute_value} is already in \textit{attr} and \textit{exclusive} is 0; the operation fails if \textit{attribute_value} is already in \textit{attr} and \textit{exclusive} is non-zero.

\textbf{fn_attribute_remove()} removes \textit{attribute_value} from \textit{attr}. The operation succeeds even if \textit{attribute_value} is not amongst \textit{attr}'s values.

**RETURN VALUES**

\textbf{fn_attribute_first()} returns 0 if the attribute contains no values. \textbf{fn_attribute_next()} returns 0 if there are no more values to be returned in the attribute (as identified by the iteration marker) or if the iteration marker is invalid.

\textbf{fn_attribute_add()} and \textbf{fn_attribute_remove()} return 1 if the operation succeeds, 0 if it fails.

**USAGE**

Manipulation of attributes using the operations described in this manual page does not affect their representation in the underlying naming system. Changes to attributes in the underlying naming system can only be effected through the use of the interfaces described in \texttt{xfn\_attributes(3N)}.

**ATTRIBUTES**

See \texttt{attributes(5)} for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**SEE ALSO**

\texttt{FN\_attrset\_t(3N)}, \texttt{FN\_attrvalue\_t(3N)}, \texttt{FN\_identifier\_t(3N)}, \texttt{fn\_attr\_get(3N)}, \texttt{fn\_attr\_modify(3N)}, \texttt{xfn(3N)}, \texttt{xfn\_attributes(3N)}, \texttt{attributes(5)}

**NOTES**

The implementation of XFN in this Solaris release is based on the X/Open preliminary specification. It is likely that there will be minor changes to these interfaces to reflect changes in the final version of this specification. The next minor release of Solaris will offer binary compatibility for applications developed using the current interfaces. As the interfaces evolve toward standardization, it is possible that future releases of Solaris will require minor source code changes to applications that have been developed against the preliminary specification.
NAME fn_attr_modify – modify specified attribute associated with name

SYNOPSIS cc [ flag ...] file ... -lxfn [ library ...]
#include <xfn/xfn.h>
int fn_attr_modify(FN_ctx_t *ctx, const FN_composite_name_t *name,
unsigned int mod_op, const FN_attribute_t *attr, FN_status_t *status);

DESCRIPTION This operation modifies according to mod_op the attribute attr associated with the object named name relative to ctx. If name is empty, the attribute associated with ctx is modified. The modification is made on the attribute identified by the attribute identifier of attr. The syntax and values of attr are used according to the modification operation. The modification operations are as follows:

FN_ATTR_OP_ADD Add an attribute with given attribute identifier and set of values. If an attribute with this identifier already exists, replace the set of values with those in the given set. The set of values may be empty if the target naming system permits.

FN_ATTR_OP_ADD_EXCLUSIVE Add an attribute with the given attribute identifier and set of values. The operation fails if an attribute with this identifier already exists. The set of values may be empty if the target naming system permits.

FN_ATTR_OP_REMOVE Remove the attribute with the given attribute identifier and all of its values. The operation succeeds even if the attribute does not exist. The values of the attribute supplied with this operation are ignored.

FN_ATTR_OP_ADD_VALUES Add the given values to those of the given attribute (resulting in the attribute having the union of its prior value set with the set given). Create the attribute if it does not exist already. The set of values may be empty if the target naming system permits.

FN_ATTR_OP_REMOVE_VALUES Remove the given values from those of the given attribute (resulting in the attribute having the set difference of its prior value set and the set given). This succeeds even if some of the given values are not in the set of values that
the attribute has. In naming systems that require an attribute to have at least one value, removing the last value will remove the attribute as well.

**RETURN VALUES**

1  Successful operation.
0  Operation failed.

**ERRORS**

fn_attr_modify() sets status as described in FN_status_t(3N) and xfn_status_codes(3N).

**ATTRIBUTES**

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**SEE ALSO**

FN_attribute_t(3N), FN_composite_name_t(3N), FN_ctx_t(3N), FN_status_t(3N), fn_attr_multi_modify(3N), xfn(3N), xfn_attributes(3N), xfn_status_codes(3N), attributes(5)

**NOTES**

The implementation of XFN in this Solaris release is based on the X/Open preliminary specification. It is likely that there will be minor changes to these interfaces to reflect changes in the final version of this specification. The next minor release of Solaris will offer binary compatibility for applications developed using the current interfaces. As the interfaces evolve toward standardization, it is possible that future releases of Solaris will require minor source code changes to applications that have been developed against the preliminary specification.
NAME  FN_attrmodlist_t, fn_attrmodlist_create, fn_attrmodlist_destroy, fn_attrmodlist_copy,
        fn_attrmodlist_assign, fn_attrmodlist_count, fn_attrmodlist_first, fn_attrmodlist_next,
        fn_attrmodlist_add – a list of attribute modifications

SYNOPSIS  cc [ flag . . . ] file . . . -lxfn [ library . . . ]
          #include <xfn/xfn.h>
          FN_attrmodlist_t *fn_attrmodlist_create(void);
          void fn_attrmodlist_destroy(FN_attrmodlist_t *modlist);
          FN_attrmodlist_t *fn_attrmodlist_copy(const FN_attrmodlist_t *modlist);
          FN_attrmodlist_t *fn_attrmodlist_assign(FN_attrmodlist_t *dst,
                                                  const FN_attrmodlist_t *src);
          unsigned int fn_attrmodlist_count(const FN_attrmodlist_t *modlist);
          const FN_attribute_t *fn_attrmodlist_first(const FN_attrmodlist_t *modlist,
                                                      void **iter_pos, unsigned int *first_mod_op);
          const FN_attribute_t *fn_attrmodlist_next(const FN_attrmodlist_t *modlist,
                                                      void **iter_pos, unsigned int *mod_op);
          int fn_attrmodlist_add(FN_attrmodlist_t *modlist, unsigned int mod_op,
                                  const FN_attribute_t *attr);

DESCRIPTION  An attribute modification list allows for multiple modification operations to be made on
               the attributes associated with a single named object. It is used in the
               fn_attr_multi_modify(3N) operation.
               An attribute modification list is a list of attribute modification specifiers. An attribute
               modification specifier consists of an attribute object and an operation specifier. The
               attribute’s identifier indicates the attribute that is to be operated upon. The attribute’s
               values are used in a manner depending on the operation. The operation specifier is an
               unsigned int that must have one of the values:
               
               FN_ATTR_OP_ADD
               FN_ATTR_OP_ADD_EXCLUSIVE
               FN_ATTR_OP_REMOVE
               FN_ATTR_OP_ADD_VALUES
               or
               FN_ATTR_OP_REMOVE_VALUES

               (See fn_attr_modify(3N) for detailed descriptions of these specifiers.) The operations are
               to be performed in the order in which they appear in the modification list.
               fn_attrmodlist_create() creates an empty attribute modification list.
               fn_attrmodlist_destroy() releases the storage associated with modlist.
               fn_attrmodlist_copy() returns a copy of the attribute modification list modlist.
               fn_attrmodlist_assign() makes a copy of src and assigns it to dst, releasing any old con-
               tents of dst. It returns a pointer to the same object as dst.

modified 13 Dec 1996       SunOS 5.6       3N-555
fn_attrmodlist_count() returns the number attribute modification items in the attribute modification list.

The iterators fn_attrmodlist_first() and fn_attrmodlist_next() return a handle to the attribute part of the modification and return the operation specifier part through an unsigned int * parameter. fn_attrmodlist_first() returns the attribute of the first modification item from modlist and sets mod_op to be the code of the modification operation of that item; iter_pos is set after the first modification item.

fn_attrmodlist_next() returns the attribute of the next modification item from modlist after iter_pos and advances iter_pos; mod_op is set to the code of the modification operation of that item. The order of the items returned during an enumeration is the same as the order by which the items were added to the modification list.

fn_attrmodlist_add() adds a new item consisting of the given modification operation code mod_op and attribute attr to the end of the modification list modlist. attr's identifier indicates the attribute that is to be operated upon. attr's values are used in a manner depending on the operation.

RETURN VALUES

fn_attrmodlist_first() returns 0 if the modification list is empty. fn_attrmodlist_next() returns 0 if there are no more items on the modification list to be enumerated or if the iteration marker is invalid.

fn_attrmodlist_add() returns 1 if the operation succeeds, 0 if the operation fails.

USAGE

Manipulation of attributes using the operations described in this manual page does not affect their representation in the underlying naming system. Changes to attributes in the underlying naming system can only be effected through the use of the interfaces described in xfn_attributes(3N).

ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO

FN_attribute_t(3N), FN_attrset_t(3N), FN_identifier_t(3N), fn_attr_modify(3N), fn_attr_multi_modify(3N), xfn(3N), xfn_attributes(3N), attributes(5)

NOTES

The implementation of XFN in this Solaris release is based on the X/Open preliminary specification. It is likely that there will be minor changes to these interfaces to reflect changes in the final version of this specification. The next minor release of Solaris will offer binary compatibility for applications developed using the current interfaces. As the interfaces evolve toward standardization, it is possible that future releases of Solaris will require minor source code changes to applications that have been developed against the preliminary specification.
Network Functions

fn_attr_multi_get (3N)

NAME
fn_attr_multi_get, FN_multigetlist_t, fn_multigetlist_next, fn_multigetlist_destroy – return multiple attributes associated with named object

SYNOPSIS
cc [flag ...] file ... -lxfn [ library ... ]
#include <xfn/xfn.h>
FN_multigetlist_t *fn_attr_multi_get(FN_ctx_t *ctx,
    const FN_composite_name_t *name, const FN_attrset_t *attr_ids,
    FN_status_t *status);
FN_attribute_t *fn_multigetlist_next(FN_multigetlist_t *ml, FN_status_t *status);
void fn_multigetlist_destroy(FN_multigetlist_t *ml, FN_status_t *status);

DESCRIPTION
This set of operations returns one or more attributes associated with the object named by
name relative to the context ctx. If name is empty, the attributes associated with ctx are
returned.
The attributes returned are those specified in attr_ids. If the value of attr_ids is 0, all attri-
butes associated with the named object are returned. Any attribute values in attr_ids pro-
vided by the caller are ignored; only the attribute identifiers are relevant for this opera-
tion. Each attribute (identifier, syntax, values) is returned one at a time using an
enumeration scheme similar to that for listing a context.

fn_attr_multi_get() initiates the enumeration process. It returns a handle to an
FN_multigetlist_t object that can be used for the enumeration.
The operation fn_multigetlist_next() returns a new FN_attribute_t object containing the
next attribute (identifiers, syntaxes, and values) requested and updates ml to indicate the
state of the enumeration.
The operation fn_multigetlist_destroy() releases the resources used during the enumera-
tion. It may be invoked before the enumeration has completed to terminate the enumera-
tion.

RETURN VALUES
fn_attr_multi_get() returns a pointer to an FN_multigetlist_t object if the enumeration
has been initiated successfully; a NULL pointer (0) is returned if it failed.
fn_multigetlist_next() returns a pointer to an FN_attribute_t object if an attribute was
returned, a NULL pointer (0) if no attribute was returned.
In the case of a failure, these operations set status to indicate the nature of the failure.

ERRORS
Each call to fn_multigetlist_next() sets status as follows:
FN_SUCCESS
    If an attribute was returned, there are more attributes to be enumerated. If no
    attribute was returned, the enumeration has completed successfully.
FN_E_ATTR_NO_PERMISSION
    The caller did not have permission to read this attribute.

modified 13 Dec 1996 SunOS 5.6 3N-557
FN_E_INSUFFICIENT_RESOURCES
Insufficient resources are available to return the attribute’s values.

FN_E_INVALID_ATTR_IDENTIFIER
This attribute identifier was not in a format acceptable to the naming system, or its contents was not valid for the format specified for the identifier.

FN_E_INVALID_ENUM_HANDLE
(No attribute should be returned with this status code). The given enumeration handle is not valid. Possible reasons could be that the handle was from another enumeration, or the object being processed no longer accepts the handle (due to such events as handle expiration or updates to the object’s attribute set).

FN_E_NO_SUCH_ATTRIBUTE
The object did not have an attribute with the given identifier.

FN_E_PARTIAL_RESULT
(No attribute should be returned with this status code). The enumeration is not yet complete but cannot be continued.

For FN_E_ATTR_NO_PERMISSION, FN_E_INVALID_ATTR_IDENTIFIER, FN_E_INSUFFICIENT_RESOURCES, or FN_E_NO_SUCH_ATTRIBUTE, the returned attribute contains only the attribute identifier (no value or syntax). For these four status codes and FN_SUCCESS (when an attribute was returned), fn_multigetlist_next() can be called again to return another attribute. All other status codes indicate that no more attributes can be returned by fn_multigetlist_next().

Other status codes, such as FN_E_COMMUNICATION_FAILURE, are also possible, in which case, no attribute is returned. In such cases, status is set as described in FN_status_t(3N) and xfn_status_codes(3N).

USAGE
Implementations are not required to return all attributes requested by attr_ids. Some may choose to return only the attributes found successfully, followed by a status of FN_E_PARTIAL_RESULT; such implementations may not necessarily return attributes identifying those that could not be read. Implementations are not required to return the attributes in any order.

There may be a relationship between the ctx argument supplied to fn_attr_multi_get() and the FN_multigetlist_t object it returns. For example, some implementations may store the context handle ctx within the FN_multigetlist_t object for subsequent fn_multigetlist_next() calls. In general, a fn_ctx_handle_destroy() should not be invoked on ctx until the enumeration has terminated.

EXAMPLES
The following code fragment illustrates to obtain all attributes associated with a given name using the fn_attr_multi_get() operations.

/* list all attributes associated with given name */

extern FN_string_t *input_string;
FN_ctx_t *ctx;
FN_composite_name_t *target_name = fn_composite_name_from_string(input_string);
FN_multigetlist_t *ml;
FN_status_t *status = fn_status_create();
FN_attribute_t *attr;
int done = 0;

ctx = fn_ctx_handle_from_initial(status);
/* error checking on 'status' */

/* attr_ids == 0 indicates all attributes are to be returned */
if ((ml=fn_attr_multi_get(ctx, target_name, 0, status)) == 0) {
    /* report 'status' and exit */
}

while ((attr=fn_multigetlist_next(ml, status)) && !done) {
    switch (fn_status_code(status)) {
    case FN_SUCCESS:
        /* do something with 'attr' */
        break;
    case FN_E_ATTR_NO_PERMISSION:
    case FN_E_ATTR_INVALID_ATTR_IDENTIFIER:
    case FN_E_NO_SUCH_ATTRIBUTE:
        /* report error using identifier in 'attr' */
        break;
    default:
        /* other error handling */
        done = 1;
        }
    if (attr)
        fn_attribute_destroy(attr);
    }
/* check 'status' for reason for end of enumeration and report if necessary */

/* clean up */
fn_multigetlist_destroy(ml, status);

/* report 'status' */

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
FN_attribute_t(3N), FN_attrset_t(3N), FN_composite_name_t(3N), FN_ctx_t(3N),
FN.identifier_t(3N), FN_status_t(3N), fn_attr_get(3N), fn_ctx_handle_destroy(3N),
fn_ctx_list_names(3N), xfn(3N), xfn_attributes(3N), xfn_status_codes(3N), attributes(5)

modified 13 Dec 1996 SunOS 5.6 3N-559
NOTES

The implementation of XFN in this Solaris release is based on the X/Open preliminary specification. It is likely that there will be minor changes to these interfaces to reflect changes in the final version of this specification. The next minor release of Solaris will offer binary compatibility for applications developed using the current interfaces. As the interfaces evolve toward standardization, it is possible that future releases of Solaris will require minor source code changes to applications that have been developed against the preliminary specification.
Network Functions

NAME
fn_attr_multi_modify – modify multiple attributes associated with named object

SYNOPSIS
cc [ flag ...] file ... -lxfn [ library ...]
#include <xfn/xfn.h>

int fn_attr_multi_modify(FN_ctx_t *ctx, const FN_composite_name_t *name,
const FN_attrmodlist_t *mods, FN_attrmodlist_t **unexecuted_mods,
FN_status_t *status);

DESCRIPTION
This operation modifies the attributes associated with the object named name relative to ctx. If name is empty, the attributes associated with ctx are modified.

In the mods parameter, the caller specifies a sequence of modifications that are to be done in order on the attributes. Each modification in the sequence specifies a modification operation code (see fn_attr_modify(3N)) and an attribute on which to operate.

The FN_attrmodlist_t type is described in FN_attrmodlist_t(3N).

RETURN VALUES
fn_attr_multi_modify() returns 1 if all the modification operations were performed successfully. The function returns 0 if any error occurs. If the operation fails, status and unexecuted_mods are set as described below.

ERRORS
If an error is encountered while performing the list of modifications, status indicates the type of error and unexecuted_mods is set to a list of unexecuted modifications. The contents of unexecuted_mods do not share any state with mods; items in unexecuted_mods are copies of items in mods and appear in the same order in which they were originally supplied in mods. The first operation in unexecuted_mods is the first one that failed and the code in status applies to this modification operation in particular. If status indicates failure and a NULL pointer (0) is returned in unexecuted_mods, that indicates no modifications were executed.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
FN_attrmodlist_t(3N), FN_composite_name_t(3N), FN_ctx_t(3N), FN_status_t(3N), fn_attr_modify(3N), xfn(3N), xfn_attributes(3N), xfn_status_codes(3N), attributes(5)

NOTES
The implementation of XFN in this Solaris release is based on the X/Open preliminary specification. It is likely that there will be minor changes to these interfaces to reflect changes in the final version of this specification. The next minor release of Solaris will offer binary compatibility for applications developed using the current interfaces. As the interfaces evolve toward standardization, it is possible that future releases of Solaris will require minor source code changes to applications that have been developed against the preliminary specification.

modified 13 Dec 1996
SunOS 5.6
3N-561
fn_attr_search (3N)  

NAME fn_attr_search, FN_searchlist_t, fn_searchlist_next, fn_searchlist_destroy – search for the atomic name of objects with the specified attributes in a single context

SYNOPSIS #include <xfn/xfn.h>

FN_searchlist_t *fn_attr_search(FN_ctx_t *ctx, const FN_composite_name_t *name, const FN_attrset_t *match_attrs, unsigned int return_ref, const FN_attrset_t *return_attr_ids, FN_status_t *status);

FN_string_t *fn_searchlist_next(FN_searchlist_t *sl, FN_ref_t **returned_ref, FN_attrset_t **returned_attrs, FN_status_t *status);

void fn_searchlist_destroy(FN_searchlist_t *sl);

DESCRIPTION This set of operations is used to enumerate names of objects bound in the target context named name relative to the context ctx with attributes whose values match all those specified by match_attrs.

The attributes specified by match_attrs form a conjunctive AND expression against which the attributes of each named object in the target context are evaluated. For multi-valued attributes, the list order of values is ignored and attribute values not specified in match_attrs are ignored. If no value is specified for an attribute in match_attrs, the presence of the attribute is tested. If the value of match_attrs is 0, all names in the target context are enumerated.

If a non-zero value of return_ref is passed to fn_attr_search(), the reference bound to the name is returned in the returned_ref argument to fn_searchlist_next(). Attribute identifiers and values associated with named objects that satisfy match_attrs may be returned by fn_searchlist_next(). The attributes returned are those listed in the return_attr_ids argument to fn_attr_search(). If the value of return_attr_ids is 0, all attributes are returned. If return_attr_ids is an empty FN_attrset_t(3N) object, no attributes are returned. Any attribute values in return_attr_ids are ignored; only the attribute identifiers are relevant for return_attr_ids.

The call to fn_attr_search() initiates the enumeration process. It returns a handle to an FN_searchlist_t object that is used to enumerate the names of the objects whose attributes match the attributes specified by match_attrs.

The operation fn_searchlist_next() returns the next name in the enumeration identified by the sl. The reference of the name is returned in returned_ref if return_ref was set in the call to fn_attr_search(). The attributes specified by return_attr_ids are returned in returned_attrs. fn_searchlist_next() also updates sl to indicate the state of the enumeration. Successive calls to fn_searchlist_next() using sl return successive names, and optionally, references and attributes, in the enumeration; these calls further update the state of the enumeration.
fn_attr_search() returns a pointer to an FN_searchlist_t object if the enumeration is successfully initiated; it returns a NULL pointer if the enumeration cannot be initiated or if no named object with attributes whose values match those specified in match_attrs is found.

fn_searchlist_next() returns a pointer to an FN_string_t object; it returns a NULL pointer if no more names can be returned in the enumeration. If returned_ref is a NULL pointer, or if the return_ref parameter to fn_attr_search was 0, no reference is returned; otherwise, returned_ref contains the reference bound to the name. If returned_attrs is NULL, no attributes are returned; otherwise, returned_attrs contains the attributes associated with the named object, as specified by the return_attr_ids parameter to fn_attr_search().

In the case of a failure, these operations return in the status argument a code indicating the nature of the failure.

Errors fn_attr_search() returns a NULL pointer if the enumeration could not be initiated. The status argument is set in the following way:

**FN_SUCCESS**
A named object could not be found whose attributes satisfied the implied filter of equality and conjunction.

**FN_E_ATTR_NO_PERMISSION**
The caller did not have permission to read one or more of the specified attributes.

**FN_E_INVALID_ATTR_VALUE**
A value type in the specified attributes did not match the syntax of the attribute against which it was being evaluated.

Other status codes are possible as described in FN_status_t(3N) and xfn_status_codes(3N).

Each successful call to fn_searchlist_next() returns a name and, optionally, the reference and requested attributes. status is set in the following way:

**FN_SUCCESS**
All requested attributes were returned successfully with the name.

**FN_E_ATTR_NO_PERMISSION**
The caller did not have permission to read one or more of the requested attributes.

**FN_E_INVALID_ATTR_IDENTIFIER**
A requested attribute identifier was not in a format acceptable to the naming system, or its contents was not valid for the format specified.

**FN_E_NO_SUCH_ATTRIBUTE**
The named object did not have one of the requested attributes.
FN_E_INSUFFICIENT_RESOURCES
Insufficient resources are available to return all the requested attributes and their values.

FN_E_ATTR_NO_PERMISSION
FN_E_INVALID_ATTR_IDENTIFIER
FN_E_NO_SUCH_ATTRIBUTE
FN_E_INSUFFICIENT_RESOURCES
These indicate that some of the requested attributes may have been returned in returned_attrs but one or more of them could not be returned. Use fn_attr_get(3N) or fn_attr_multi_get(3N) to discover why these attributes could not be returned.

fn_searchlist_next() returns a NULL pointer if no more names can be returned. The status argument is set in the following way:

FN_SUCCESS
The search has completed successfully.

FN_E_PARTIAL_RESULT
The enumeration is not yet complete but cannot be continued.

FN_E_ATTR_NO_PERMISSION
The caller did not have permission to read one or more of the specified attributes.

FN_E_INVALID_ENUM_HANDLE
The supplied enumeration handle was not valid. Possible reasons could be that the handle was from another enumeration, or the context being enumerated no longer accepts the handle (due to such events as handle expiration or updates to the context).

Other status codes are possible as described in FN_status_t(3N) and xfn_status_codes(3N).

USAGE
The names enumerated using fn_searchlist_next() are not ordered in any way. Furthermore, there is no guarantee that any two series of enumerations on the same context with identical match_attrs will return the names in the same order.

EXAMPLES
The following code fragment illustrates how the fn_attr_search() operation may be used. The code consists of three parts: preparing the arguments for the search, performing the search, and cleaning up.

The first part involves getting the name of the context to start the search and constructing the set of attributes that named objects in the context must satisfy. This is done in the declarations part of the code and by the routine get_search_query.

The next part involves doing the search and enumerating the results of the search. This is done by first getting a context handle to the Initial Context, and then passing that handle along with the name of the target context and matching attributes to fn_attr_search. This particular call to fn_attr_search() is requesting that no reference be returned (by passing in 0 for return_ref), and that all attributes associated with the named object be
Network Functions

returned (by passing in 0 as the return_attr_ids argument). If successful, \texttt{fn_attr_search()} returns sl, a handle for enumerating the results of the search. The results of the search are enumerated using calls to \texttt{fn_searchlist_next()}, which returns the name of the object and the attributes associated with the named object in returned_attrs.

The last part of the code involves cleaning up the resources used during the search and enumeration. The call to \texttt{fn_searchlist_destroy()} releases resources reserved for this enumeration. The other calls release the context handle, name, attribute set, and status objects created earlier.

\begin{verbatim}
/* Declarations */
FN_ctx_t *ctx;
FN_searchlist_t *sl;
FN_string_t *name;
FN_attrset_t *returned_attrs;
FN_status_t *status = fn_status_create();
FN_composite_name_t *target_name = get_name_from_user_input();
FN_attrset_t *match_attrs = get_search_query();

/* Get context handle to Initial Context */
ctx = fn_ctx_handle_from_initial(status);

/* error checking on 'status' */

/* Initiate search */
if ((sl=fn_attr_search(ctx, target_name, match_attrs,
    /* no reference */ 0, /* return all attrs */ 0, status)) == 0) {
    /* report 'status', cleanup, and exit */
}

/* Enumerate names and attributes requested */
while (name=fn_searchlist_next(sl, 0, &returned_attrs, status)) {
    /* do something with 'name' and 'returned_attrs'*/
    fn_string_destroy(name);
    fn_attrset_destroy(returned_attrs);
}

/* check 'status' for reason for end of enumeration */

/* Clean up */
fn_searchlist_destroy(sl); /* Free resources of 'sl' */
fn_status_destroy(status);
fn_attrset_destroy(match_attrs);
fn_ctx_handle_destroy(ctx);
\end{verbatim}

modified 22 Nov 1996 SunOS 5.6 3N-565
fn_composite_name_destroy(target_name);

/*
 * Procedure for constructing attribute set containing
 * attributes to be matched:
 *      "zip_code" attribute value is "02158"
 * AND "employed" attribute is present.
 */

FN_attrset_t *
get_search_query()
{
    /* Zip code and employed attribute identifier, syntax */
    extern FN_attribute_t    *attr_zip_code;
    extern FN_attribute_t    *attr_employed;

    FN_attribute_t *zip_code = fn_attribute_copy(attr_zip_code);
    FN_attribute_t *employed = fn_attribute_copy(attr_employed);

    FN_attr_value_t zc_value = {5, "02158"};
    FN_attrset_t *match_attrs = fn_attrset_create();

    fn_attribute_add(zip_code, &zc_value, 0);
    fn_attrset_add(match_attrs, zip_code, 0);
    fn_attrset_add(match_attrs, attr_employed, 0);

    return (match_attrs);
}

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
FN_attribute_t(3N), FN_attrset_t(3N), FN_attrvalue_t(3N), FN_composite_name_t(3N),
FN_ctx_t(3N), FN_status_t(3N), FN_string_t(3N), fn_attr_ext_search(3N),
fn_attr_get(3N), fn_attr_multi_get(3N), fn_ctx_list_names(3N), xfn_status_codes(3N),
attributes(5)
Network Functions

NAME

FN_attrset_t, fn_attrset_create, fn_attrset_destroy, fn_attrset_copy, fn_attrset_assign,
fn_attrset_get, fn_attrset_count, fn_attrset_first, fn_attrset_next, fn_attrset_add,
fn_attrset_remove — a set of XFN attributes

SYNOPSIS

cc [ flag ... ] file ... -lxfn [ library ... ]
#include <xfn/xfn.h>
FN_attrset_t *fn_attrset_create(void);
void fn_attrset_destroy(FN_attrset_t *aset);
FN_attrset_t *fn_attrset_copy(const FN_attrset_t *aset);
FN_attrset_t *fn_attrset_assign(FN_attrset_t *dst, const FN_attrset_t *src);
const FN_attribute_t *fn_attrset_get(const const FN_attrset_t *aset,
 const FN_identifier_t *attr_id);
unsigned int fn_attrset_count(const FN_attrset_t *aset);
const FN_attribute_t *fn_attrset_first(const FN_attrset_t *aset, void **iter_pos);
const FN_attribute_t *fn_attrset_next(const FN_attrset_t *aset, void **iter_pos);
int fn_attrset_add(FN_attrset_t *aset, const FN_attribute_t *attr,
 unsigned int exclusive);
int fn_attrset_remove(FN_attrset_t *aset, const FN_identifier_t *attr_id);

DESCRIPTION

An attribute set is a set of attribute objects with distinct identifiers. The
fn_attr_multi_get(3N) operation takes an attribute set as parameter and returns an attribute
set. The fn_attr_get_ids(3N) operation returns an attribute set containing the
identifiers of the attributes.

Attribute sets are represented by the type FN_attrset_t. The following operations are
defined for manipulating attribute sets.

fn_attrset_create() creates an empty attribute set. fn_attrset_destroy() releases the
storage associated with the attribute set aset. fn_attrset_copy() returns a copy of the
attribute set aset. fn_attrset_assign() makes a copy of the attribute set src and assigns it
to dst, releasing any old contents of dst. A pointer to the same object as dst is returned.

fn_attrset_get() returns the attribute with the given identifier attr_id from aset.
fn_attrset_count() returns the number attributes found in the attribute set aset.

fn_attrset_first() and fn_attrset_next() are functions that can be used to return an
enumeration of all the attributes in an attribute set. The attributes are not ordered in any
way. There is no guaranteed relation between the order in which items are added to an
attribute set and the order of the enumeration. The specification does guarantee that any
two enumerations will return the members in the same order, provided that no
fn_attrset_add() or fn_attrset_remove() operation was performed on the object in
between or during the two enumerations. fn_attrset_first() returns the first attribute
from the set and sets iter_pos after the first attribute. fn_attrset_next() returns the attribute
following iter_pos and advances iter_pos.

modified 13 Dec 1996 SunOS 5.6 3N-567
**fn_attrset_add()** adds the attribute `attr` to the attribute set `aset`, replacing the attribute’s values if the identifier of `attr` is not distinct in `aset` and `exclusive` is 0. If `exclusive` is non-zero and the identifier of `attr` is not distinct in `aset`, the operation fails.

**fn_attrset_remove()** removes the attribute with the identifier `attr_id` from `aset`. The operation succeeds even if no such attribute occurs in `aset`.

**RETURN VALUES**
- `fn_attrset_first()` returns 0 if the attribute set is empty. `fn_attrset_next()` returns 0 if there are no more attributes in the set.
- `fn_attrset_add()` and `fn_attrset_remove()` return 1 if the operation succeeds, and 0 if the operation fails.

**USAGE**
Manipulation of attributes using the operations described in this manual page does not affect their representation in the underlying naming system. Changes to attributes in the underlying naming system can only be effected through the use of the interfaces described in `xfn_attributes(3N)`.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**SEE ALSO**
- `FN_attribute_t(3N)`, `FN_attrvalue_t(3N)`, `FN_identifier_t(3N)`, `fn_attr_get_ids(3N)`, `fn_attr_multi_get(3N)`, `xfn(3N)`, `xfn_attributes(3N)`, `attributes(5)`

**NOTES**
The implementation of XFN in this Solaris release is based on the X/Open preliminary specification. It is likely that there will be minor changes to these interfaces to reflect changes in the final version of this specification. The next minor release of Solaris will offer binary compatibility for applications developed using the current interfaces. As the interfaces evolve toward standardization, it is possible that future releases of Solaris will require minor source code changes to applications that have been developed against the preliminary specification.
NAME FN_attrvalue_t – an XFN attribute value

SYNOPSIS cc [ flag ... ] file ... -lxfn [ library ... ]
#include <xfn/xfn.h>

DESCRIPTION The type FN_attrvalue_t is used to represent the contents of a single attribute value, within an attribute of type FN_attribute_t.
The representation of this structure is defined by XFN as follows:

typedef struct {
    size_t length;
    void *contents;
} FN_attrvalue_t;

SEE ALSO FN_attribute_t(3N), fn_attr_get_values(3N), xfn(3N)

modified 4 Nov 1994 SunOS 5.6 3N-569
NAME

FN_composite_name_t, fn_composite_name_create, fn_composite_name_destroy,
fn_composite_name_from_str, fn_composite_name_from_string,
fn_string_from_composite_name, fn_composite_name_copy, fn_composite_name_assign,
fn_composite_name_is_empty, fn_composite_name_count, fn_composite_name_first,
fn_composite_name_next, fn_composite_name_prev, fn_composite_name_last,
fn_composite_name_prefix, fn_composite_name_suffix, fn_composite_name_is_equal,
fn_composite_name_is_prefix, fn_composite_name_is_suffix,
fn_composite_name_prepend_comp, fn_composite_name_append_comp,
fn_composite_name_insert_comp, fn_composite_name_delete_comp,
fn_composite_name_prepend_name, fn_composite_name_append_name,
fn_composite_name_insert_name – a sequence of component names spanning multiple
naming systems

SYNOPSIS

cc [ flag ... ] file ... -lxfn [ library ... ]
#include <xfn/xfn.h>
FN_composite_name_t *fn_composite_name_create(void);
void fn_composite_name_destroy(FN_composite_name_t *name);
FN_composite_name_t *fn_composite_name_from_str( const unsigned char *cstr);
FN_composite_name_t *fn_composite_name_from_string( const FN_string_t *str);
FN_string_t *fn_string_from_composite_name( const FN_composite_name_t *name,
unsigned int *status);
FN_composite_name_t *fn_composite_name_copy( const FN_composite_name_t *name);
FN_composite_name_t *fn_composite_name_assign( FN_composite_name_t *dst,
const FN_composite_name_t *src);
int fn_composite_name_is_empty( const FN_composite_name_t *name);
unsigned int fn_composite_name_count( const FN_composite_name_t *name);
const FN_string_t *fn_composite_name_first( const FN_composite_name_t *name,
void **iter_pos);
const FN_string_t *fn_composite_name_next( const FN_composite_name_t *name,
void **iter_pos);
const FN_string_t *fn_composite_name_prev( const FN_composite_name_t *name,
void **iter_pos);
const FN_string_t *fn_composite_name_last( const FN_composite_name_t *name,
void **iter_pos);
FN_composite_name_t *fn_composite_name_prefix( const FN_composite_name_t *name,
const void *iter_pos);
FN_composite_name_t *fn_composite_name_suffix( const FN_composite_name_t *name,
const void *iter_pos);
int fn_composite_name_is_equal( const FN_composite_name_t *name,
    const FN_composite_name_t *name2, unsigned int *status);

int fn_composite_name_is_prefix( const FN_composite_name_t *name,
    const FN_composite_name_t *prefix, void **iter_pos, unsigned int *status);

int fn_composite_name_is_suffix( const FN_composite_name_t *name,
    const FN_composite_name_t *suffix, void **iter_pos, unsigned int *status);

int fn_composite_name_prepend_comp( FN_composite_name_t *name,
    const FN_string_t *newcomp);

int fn_composite_name_append_comp( FN_composite_name_t *name,
    const FN_string_t *newcomp);

int fn_composite_name_insert_comp( FN_composite_name_t *name,
    void **iter_pos, const FN_string_t *newcomp);

int fn_composite_name_delete_comp( FN_composite_name_t *name, void **iter_pos);

int fn_composite_name_prepend_name( FN_composite_name_t *name,
    const FN_composite_name_t *newcomps);

int fn_composite_name_append_name( FN_composite_name_t *name,
    const FN_composite_name_t *newcomps);

int fn_composite_name_insert_name( FN_composite_name_t *name,
    void **iter_pos, const FN_composite_name_t *newcomps);

DESCRIPTION

A composite name is represented by an object of type FN_composite_name_t. Each component is a string name, of type FN_string_t, from the namespace of a single naming system. It may be an atomic name or a compound name in that namespace.

fn_composite_name_create creates an FN_composite_name_t object with zero components. Components may be subsequently added to the composite name using the modify operations described below. fn_composite_name_destroy releases any storage associated with the given FN_composite_name_t handle.

fn_composite_name_from_str() creates an FN_composite_name_t from the given null-terminated string based on the code set of the current locale setting, using the XFN composite name syntax. fn_composite_name_from_string() creates an FN_composite_name_t from the string str using the XFN composite name syntax.

fn_string_from_composite_name() returns the standard string form of the given composite name, by concatenating the components of the composite name in a left to right order, each separated by the XFN component separator.

fn_composite_name_copy() returns a copy of the given composite name object.

fn_composite_name_assign() makes a copy of the composite name object pointed to by src and assigns it to dst, releasing any old contents of dst. A pointer to the same object as dst is returned.
**fn_composite_name_is_empty()** returns 1 if the given composite name is an empty composite name (that is, it consists of a single, empty component name); otherwise, it returns 0. **fn_composite_name_count()** returns the number of components in the given composite name.

The iteration scheme is based on the exchange of an opaque `void *` argument, `iter_pos`, that serves to record the position of the iteration in the sequence. Conceptually, `iter_pos` records a position between two successive components (or at one of the extreme ends of the sequence).

The function **fn_composite_name_first()** returns a handle to the `FN_string_t` that is the first component in the name, and sets `iter_pos` to indicate the position immediately following the first component. It returns 0 if the name has no components. Thereafter, successive calls of the **fn_composite_name_next()** function return pointers to the component following the iteration marker, and advance the iteration marker. If the iteration marker is at the end of the sequence, **fn_composite_name_next()** returns 0. Similarly, **fn_composite_name_prev()** returns the component preceding the iteration pointer and moves the marker back one component. If the marker is already at the beginning of the sequence, **fn_composite_name_prev()** returns 0. The function **fn_composite_name_last()** returns a pointer to the last component of the name and sets the iteration marker immediately preceding this component (so that subsequent calls to **fn_composite_name_prev()** can be used to step through leading components of the name).

The **fn_composite_name_suffix()** function returns a composite name consisting of a copy of those components following the supplied iteration marker. The method **fn_composite_name_prefix()** returns a composite name consisting of those components that precede the iteration marker. Using these functions with an iteration marker that was not initialized using **fn_composite_name_first(), fn_composite_name_last(), fn_composite_name_is_prefix(), or fn_composite_name_is_suffix()** yields undefined and generally undesirable behavior.

The functions **fn_composite_name_is_equal(), fn_composite_name_is_prefix(), and fn_composite_name_is_suffix()** test for equality between composite names or between parts of composite names. For these functions, equality is defined as exact string equality, not name equivalence. A name’s syntactic property, such as case-insensitivity, is not taken into account by these functions.

The function **fn_composite_name_is_prefix()** tests if one composite name is a prefix of another. If so, it returns 1 and sets the iteration marker immediately following the prefix. (For example, a subsequent call to **fn_composite_name_suffix()** will return the remainder of the name.) Otherwise, it returns 0 and the value of the iteration marker is undefined. The function **fn_composite_name_is_suffix()** is similar. It tests if one composite name is a suffix of another. If so, it returns 1 and sets the iteration marker immediately preceding the suffix.

The functions **fn_composite_name_prepend_comp()** and **fn_composite_name_append_comp()** prepend and append a single component to the given composite name, respectively. These operations invalidate any iteration marker the
client holds for that object. \texttt{fn\_composite\_name\_insert\_comp()} inserts a single component before \texttt{iter\_pos} to the given composite name and sets \texttt{iter\_pos} to be immediately after the component just inserted. \texttt{fn\_composite\_name\_delete\_comp()} deletes the component located before \texttt{iter\_pos} from the given composite name and sets \texttt{iter\_pos} back one component.

The functions \texttt{fn\_composite\_name\_prepend\_name()}, \texttt{fn\_composite\_name\_append\_name()}, and \texttt{fn\_composite\_name\_insert\_name()} perform the same update functions as their \_comp counterparts, respectively, except that multiple components are being added, rather than single components. For example, \texttt{fn\_composite\_name\_insert\_name()} sets \texttt{iter\_pos} to be immediately after the name just added.

\textbf{RETURN VALUES} The functions \texttt{fn\_composite\_name\_is\_empty()}, \texttt{fn\_composite\_name\_is\_equal()}, \texttt{fn\_composite\_name\_is\_suffix()}, and \texttt{fn\_composite\_name\_is\_prefix()} return 1 if the test indicated is true; 0 otherwise.

The update functions \texttt{fn\_composite\_name\_prepend\_comp()}, \texttt{fn\_composite\_name\_append\_comp()}, \texttt{fn\_composite\_name\_insert\_comp()}, \texttt{fn\_composite\_name\_delete\_comp()}, and their _name counterparts return 1 if the update was successful; 0 otherwise.

If a function is expected to return a pointer to an object, a NULL pointer (0) is returned if the function fails.

\textbf{ERRORS} Code set mismatches that occur during the composition of the string form or during comparisons of composite names are resolved in an implementation-dependent way. \texttt{fn\_string\_from\_composite\_name()}, \texttt{fn\_composite\_name\_is\_equal()}, \texttt{fn\_composite\_name\_is\_suffix()}, and \texttt{fn\_composite\_name\_is\_prefix()} set \texttt{status} to \texttt{FN\_E\_INCOMPATIBLE\_CODE\_SETS} for composite names whose components have code sets that are determined by the implementation to be incompatible.

\textbf{ATTRIBUTES} See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

\textbf{SEE ALSO} \texttt{FN\_string\_t(3N)}, \texttt{xfn(3N)}, attributes(5)

\textbf{NOTES} The implementation of XFN in this Solaris release is based on the X/Open preliminary specification. It is likely that there will be minor changes to these interfaces to reflect changes in the final version of this specification. The next minor release of Solaris will offer binary compatibility for applications developed using the current interfaces. As the interfaces evolve toward standardization, it is possible that future releases of Solaris will require minor source code changes to applications that have been developed against the preliminary specification.
NAME
FN_compound_name_t, fn_compound_name_from_syntax_attrs,
fn_compound_name_get_syntax_attrs, fn_compound_name_destroy,
fn_string_from_compound_name, fn_compound_name_copy,
fn_compound_name_assign, fn_compound_name_count, fn_compound_name_first,
fn_compound_name_next, fn_compound_name_prev, fn_compound_name_last,
fn_compound_name_prefix, fn_compound_name_suffix, fn_compound_name_is_empty,
fn_compound_name_is_equal, fn_compound_name_is_prefix,
fn_compound_name_is_suffix, fn_compound_name_prepend_comp,
fn_compound_name_append_comp, fn_compound_name_insert_comp,
fn_compound_name_delete_comp, fn_compound_name_delete_all – an XFN compound
name

SYNOPSIS
cc [ flag … ] file … -lxfn [ library … ]
#include <xfn/xfn.h>
FN_compound_name_t *fn_compound_name_from_syntax_attrs(
    const FN_attrset_t *aset, const FN_string_t *name, FN_status_t *status);
FN_attrset_t *fn_compound_name_get_syntax_attrs(
    const FN_compound_name_t *name);
void fn_compound_name_destroy( FN_compound_name_t *name);
FN_string_t *fn_string_from_compound_name( const FN_compound_name_t *name);
FN_compound_name_t *fn_compound_name_copy( const FN_compound_name_t *name);
FN_compound_name_t *fn_compound_name_assign( FN_compound_name_t *dst,
    const FN_compound_name_t *src);
unsigned int fn_compound_name_count( const FN_compound_name_t *name);
const FN_string_t *fn_compound_name_first( const FN_compound_name_t *name,
    void **iter_pos);
const FN_string_t *fn_compound_name_next( const FN_compound_name_t *name,
    void **iter_pos);
const FN_string_t *fn_compound_name_prev( const FN_compound_name_t *name,
    void **iter_pos);
const FN_string_t *fn_compound_name_last( const FN_compound_name_t *name,
    void **iter_pos);
FN_compound_name_t *fn_compound_name_prefix( const FN_compound_name_t *name, const void *iter_pos);
FN_compound_name_t *fn_compound_name_suffix( const FN_compound_name_t *name, const void *iter_pos);
int fn_compound_name_is_empty( const FN_compound_name_t *name);
int fn_compound_name_is_equal( const FN_compound_name_t *name1,
            const FN_compound_name_t *name2, unsigned int *status);

int fn_compound_name_is_prefix( const FN_compound_name_t *name,
            const FN_compound_name_t *prefix, void **iter_pos, unsigned int *status);

int fn_compound_name_is_suffix( const FN_compound_name_t *name,
            const FN_compound_name_t *suffix, void **iter_pos, unsigned int *status);

int fn_compound_name_prepend_comp( FN_compound_name_t *name,
            const FN_string_t *atomic_comp, unsigned int *status);

int fn_compound_name_append_comp( FN_compound_name_t *name,
            const FN_string_t *atomic_comp, unsigned int *status);

int fn_compound_name_insert_comp( FN_compound_name_t *name, void **iter_pos,
            const FN_string_t *atomic_comp, unsigned int *status);

int fn_compound_name_delete_comp( FN_compound_name_t *name,
            void **iter_pos);

int fn_compound_name_delete_all( FN_compound_name_t *name);

DESCRIPTION

Most applications treat names as opaque data. Hence, the majority of clients of the XFN interface will not need to parse names. Some applications, however, such as browsers, need to parse names. For these applications, XFN provides support in the form of the FN_compound_name_t object.

Each naming system in an XFN federation potentially has its own naming conventions. The FN_compound_name_t object has associated operations for applications to process compound names that conform to the XFN model of expressing compound name syntax. The XFN syntax model for compound names covers a large number of specific name syntaxes and is expressed in terms of syntax properties of the naming convention. See xfn_compound_names(3N).

An FN_compound_name_t object is constructed by the operation fn_compound_name_from_syntax_attrs, using a string name and an attribute set containing the "fn_syntax_type" (with identifier format FN_ID_STRING) attribute identifying the namespace syntax of the string name. The value "standard" (with identifier format FN_ID_STRING) in the "fn_syntax_type" specifies a syntax model that is by default supported by the FN_compound_name_t object. An implementation may support other syntax types instead of the XFN standard syntax model, in which case the value of the "fn_syntax_type" attribute would be set to an implementation-specific string.

fn_compound_name_get_syntax_attrs() returns an attribute set containing the syntax attributes that describes the given compound name. fn_compound_name_destroy() releases the storage associated with the given compound name.

fn_string_from_compound_name() returns the string form of the given compound name. fn_compound_name_copy() returns a copy of the given compound name.

fn_compound_name_assign() makes a copy of the compound name src and assigns it to dst, releasing any old contents of dst. A pointer to the object pointed to by dst is returned.

fn_compound_name_count() returns the number of atomic components in the given compound name.
The function `fn_compound_name_first()` returns a handle to the `FN_string_t` that is the first atomic component in the compound name, and sets `iter_pos` to indicate the position immediately following the first component. It returns 0 if the name has no components. Thereafter, successive calls of the `fn_compound_name_next()` function return pointers to the component following the iteration marker, and advance the iteration marker. If the iteration marker is at the end of the sequence, `fn_compound_name_next()` returns 0. Similarly, `fn_compound_name_prev()` returns the component preceding the iteration pointer and moves the marker back one component. If the marker is already at the beginning of the sequence, `fn_compound_name_prev()` returns 0. The function `fn_compound_name_last()` returns a pointer to the last component of the name and sets the iteration marker immediately preceding this component (so that subsequent calls to `fn_compound_name_prev()` can be used to step through trailing components of the name).

The `fn_compound_name_suffix()` function returns a compound name consisting of a copy of those components following the supplied iteration marker. The function `fn_compound_name_prefix()` returns a compound name consisting of those components that precede the iteration marker. Using these functions with an iteration marker that was not initialized with the use of `fn_compound_name_first()`, `fn_compound_name_last()`, `fn_compound_name_is_prefix()`, or `fn_compound_name_is_suffix()` yields undefined and generally undesirable behavior.

The functions `fn_compound_name_is_equal()`, `fn_compound_name_is_prefix()`, and `fn_compound_name_is_suffix()` test for equality between compound names or between parts of compound names. For these functions, equality is defined as name equivalence. A name’s syntactic property, such as case-insensitivity, is taken into account by these functions.

The function `fn_compound_name_is_prefix()` tests if one compound name is a prefix of another. If so, it returns 1 and sets the iteration marker immediately following the prefix. (For example, a subsequent call to `fn_compound_name_suffix()` will return the remainder of the name.) Otherwise, it returns 0 and value of the iteration marker is undefined. The function `fn_compound_name_is_suffix()` is similar. If tests if one compound name is a suffix of another. If so, it returns 1 and sets the iteration marker immediately preceding the suffix.

The functions `fn_compound_name_prepend_comp()` and `fn_compound_name_append_comp()` prepend and append a single atomic component to the given compound name, respectively. These operations invalidate any iteration marker the client holds for that object. `fn_compound_name_insert_comp()` inserts an atomic component before `iter_pos` to the given compound name and sets `iter_pos` to be immediately after the component just inserted. `fn_compound_name_delete_comp()` deletes the atomic component located before `iter_pos` from the given compound name and sets `iter_pos` back one component. `fn_compound_name_delete_all()` deletes all the atomic components from `name`. 

3N-576 SunOS 5.6 modified 13 Dec 1996
RETURN VALUES

The following test functions return 1 if the test indicated is true; otherwise, they return 0:

- `fn_compound_name_is_empty`
- `fn_compound_name_is_equal`
- `fn_compound_name_is_suffix`
- `fn_compound_name_is_prefix`

The following update functions return 1 if the update was successful; otherwise, they return 0:

- `fn_compound_name_prepend_comp`
- `fn_compound_name_append_comp`
- `fn_compound_name_insert_comp`
- `fn_compound_name_delete_comp`
- `fn_compound_name_delete_all`

If a function is expected to return a pointer to an object, a NULL pointer (0) is returned if the function fails.

ERRORS

When the function `fn_compound_name_from_syntax_attrs` fails, it returns a status code in `status`. The possible status codes are:

- **FN_E_ILLEGAL_NAME**
  The name supplied to the operation was not a well-formed XFN compound name, or one of the component names was not well-formed according to the syntax of the naming system(s) involved in its resolution.

- **FN_E_INCOMPATIBLE_CODE_SETS**
  The code set of the given string is incompatible with that supported by the compound name.

- **FN_E_INVALID_SYNTAX_ATTRS**
  The syntax attributes supplied are invalid or insufficient to fully specify the syntax.

- **FN_E_SYNTAX_NOT_SUPPORTED**
  The syntax type specified is not supported.

The following functions may return in `status` the status code `FN_E_INCOMPATIBLE_CODE_SETS` when the code set of the given string is incompatible with that of the compound name:

- `fn_compound_name_is_equal`
- `fn_compound_name_is_suffix`
- `fn_compound_name_is_prefix`
- `fn_compound_name_prepend_comp`
- `fn_compound_name_append_comp`
- `fn_compound_name_insert_comp`

ATTRIBUTES

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

---

modified 13 Dec 1996

SunOS 5.6

3N-577
<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**SEE ALSO**
FN_attribute_t(3N), FN_attrset_t(3N), FN_composite_name_t(3N), FN_status_t(3N), FN_string_t(3N), fn_ctx_get_syntax_attrs(3N), xfn(3N), xfn_compound_names(3N), attributes(5)

**NOTES**
The implementation of XFN in this Solaris release is based on the X/Open preliminary specification. It is likely that there will be minor changes to these interfaces to reflect changes in the final version of this specification. The next minor release of Solaris will offer binary compatibility for applications developed using the current interfaces. As the interfaces evolve toward standardization, it is possible that future releases of Solaris will require minor source code changes to applications that have been developed against the preliminary specification.
NAME
fn_ctx_bind – bind a reference to a name

SYNOPSIS
cc [ flag ...] file ... -lxfn [ library ...]
#include <xfn/xfn.h>

int fn_ctx_bind(FN_ctx_t *ctx, const FN_composite_name_t *name,
    const FN_ref_t *ref, unsigned int exclusive, FN_status_t *status);

DESCRIPTION
This operation binds the supplied reference \textit{ref} to the supplied composite name \textit{name} relative to \textit{ctx}. The binding is made in the target context, that is, the context named by all but the terminal atomic part of \textit{name}. The operation binds the terminal atomic name to the supplied reference in the target context. The target context must already exist.

The value of \textit{exclusive} determines what happens if the terminal atomic part of the name is already bound in the target context. If \textit{exclusive} is nonzero and \textit{name} is already bound, the operation fails. If \textit{exclusive} is 0, the new binding replaces any existing binding.

RETURN VALUES
When the bind operation is successful it returns 1; on error it returns 0.

ERRORS
\textbf{fn_ctx_bind} sets \textit{status} as described in \texttt{FN_status_t(3N)} and \texttt{xfn_status_codes}. Of special relevance for this operation is the status code \texttt{FN_E_NAME_IN_USE}, which indicates that the supplied name is already in use.

USAGE
The value of \textit{ref} cannot be \texttt{NULL}. If the intent is to reserve a name using \textbf{fn_ctx_bind()}, a reference containing no address should be supplied. This reference may be name service-specific or it may be the conventional \texttt{NULL} reference defined in the X/Open registry (see \texttt{fns_references}(5)).

If multiple sources are updating a reference, they must synchronize amongst each other when adding, modifying, or removing from the address list of a bound reference.

ATTRIBUTES
See \texttt{attributes}(5) for descriptions of the following attributes:

\begin{table}[h]
\centering
\begin{tabular}{|c|c|}
\hline
\textbf{ATTRIBUTE TYPE} & \textbf{ATTRIBUTE VALUE} \\
\hline
MT-Level & MT-Safe \\
\hline
\end{tabular}
\end{table}

SEE ALSO
\texttt{FN_composite_name_t(3N)}, \texttt{FN_ctx_t(3N)}, \texttt{FN_ref_t(3N)}, \texttt{FN_status_t(3N)}, \texttt{fn_ctx_lookup(3N)}, \texttt{fn_ctx_unbind(3N)}, \texttt{xfn(3N)}, \texttt{xfn_status_codes(3N)}, \texttt{attributes(5)}, \texttt{fns_references(5)}

NOTES
The implementation of XFN in this Solaris release is based on the X/Open preliminary specification. It is likely that there will be minor changes to these interfaces to reflect changes in the final version of this specification. The next minor release of Solaris will offer binary compatibility for applications developed using the current interfaces. As the interfaces evolve toward standardization, it is possible that future releases of Solaris will require minor source code changes to applications that have been developed against the preliminary specification.

modified 13 Dec 1996
SunOS 5.6

3N-579
NAME fn_ctx_create_subcontext – create a subcontext in a context

SYNOPSIS cc [ flag ...] file ... -lxfn [ library ...]
#include <xfn/xfn.h>
FN_ref_t* fn_ctx_create_subcontext(FN_ctx_t* ctx,
          const FN_composite_name_t* name, FN_status_t* status);

DESCRIPTION This operation creates a new XFN context of the same type as the target context — that named by all but the terminal atomic component of name — and binds it to the supplied composite name.

As with fn_ctx_bind(), the target context must already exist. The new context is created and bound in the target context using the terminal atomic name in name. The operation returns a reference to the newly created context.

RETURN VALUE fn_ctx_create_subcontext() returns a reference to the newly created context; if the operation fails, it returns a NULL pointer (0).

ERRORS fn_ctx_create_subcontext() sets status as described in FN_status_t(3N) and xfn_status_codes(3N). Of special relevance for this operation is the following status code:

FN_E_NAME_IN_USE
The terminal atomic name already exists in the target context.

APPLICATION USAGE The new subcontext is an XFN context and is created in the same naming system as the target context. The new subcontext also inherits the same syntax attributes as the target context. XFN does not specify any further properties of the new subcontext. The target context and its naming system determine these.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe.</td>
</tr>
</tbody>
</table>

SEE ALSO FN_composite_name_t(3N), FN_ctx_t(3N), FN_ref_t(3N), FN_status_t(3N), fn_ctx_bind(3N), fn_ctx_lookup(3N), fn_ctx_destroy_subcontext(3N), xfn_status_codes(3N), xfn(3N), attributes(5)
Network Functions

fn_ctx_destroy_subcontext (3N)

NAME
fn_ctx_destroy_subcontext – destroy the named context and remove its binding from the parent context

SYNOPSIS
cc [ flag ... ] file ... -lxfn [ library ... ]
#include <xfn/xfn.h>
int fn_ctx_destroy_subcontext(FN_ctx_t *ctx, const FN_composite_name_t *name,
   FN_status_t *status);

DESCRIPTION
This operation destroys the subcontext named by name relative to ctx, and unbinds the name.
As with fn_ctx_unbind(), this operation succeeds even if the terminal atomic name is not bound in the target context — the context named by all but the terminal atomic name in name.

RETURN VALUE
fn_ctx_destroy_subcontext() returns 1 on success and 0 on failure.

ERRORS
fn_ctx_destroy_subcontext() sets status as described in FN_status_t(3N) and xfn_status_codes(3N). Of special relevance for fn_ctx_destroy_subcontext() are the following status codes:
FN_E_CTX_NOT_A_CONTEXT
name does not name a context.
FN_E_CTX_NOT_EMPTY
The naming system being asked to do the destroy does not support removal of a context that still contains bindings.

APPLICATION USAGE
Some aspects of this operation are not specified by XFN, but are determined by the target context and its naming system. For example, XFN does not specify what happens if the named subcontext is non-empty when the operation is invoked.
In naming systems that support attributes, and store the attributes along with names or contexts, this operation removes the name, the context, and its associated attributes.
Normal resolution always follows links. In a fn_ctx_destroy_subcontext() operation, resolution of name continues to the target context; the terminal atomic name is not resolved. If the terminal atomic name is bound to a link, the link is not followed and the operation fails with FN_E_CTX_NOT_A_CONTEXT because the name is not bound to a context.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe.</td>
</tr>
</tbody>
</table>

modified 30 Dec 1996  SunOS 5.6  3N-581
SEE ALSO

FN_ctx_t(3N), FN_composite_name_t(3N), FN_status_t(3N), fn_ctx_create_subcontext(3N), fn_ctx_unbind(3N), xfn(3N), xfn_status_codes(3N), attributes(5)
Network Functions

fn_ctx_equivalent_name(3N)

NAME
fn_ctx_equivalent_name – construct an equivalent name in same context

SYNOPSIS
#include <xfn/xfn.h>

FN_composite_name_t *fn_ctx_equivalent_name(FN_ctx_t *ctx,
               const FN_composite_name_t *name, const FN_string_t *leading_name,
               FN_status_t *status);

DESCRIPTION
Given the name of an object name relative to the context ctx, this operation returns an
equivalent name for that object, relative to the same context ctx, that has leading_name as
its initial atomic name. Two names are said to be equivalent if they have prefixes that
resolve to the same context, and the parts of the names immediately following the
prefixes are identical.

The existence of a binding for leading_name in ctx does not guarantee that a name
equivalent to name can be constructed. The failure may be because such equivalence is
not meaningful, or due to the inability of the system to construct a name with the
equivalence. For example, supplying _thishost as leading_name when name starts with
_myself to fn_ctx_equivalent_name() in the Initial Context would not be meaningful;
this results in the return of the error code FN_E_NO_EQUIVALENT_NAME.

RETURN VALUES
If an equivalent name cannot be constructed, the value 0 is returned and status is set
appropriately.

ERRORS
fn_ctx_equivalent_name() sets status as described in FN_status_t(3N) and
xfn_status_codes(3N). The following status code is especially relevant for this operation:

FN_E_NO_EQUIVALENT_NAME
No equivalent name can be constructed, either because there is no mean-
ingful equivalence between name and leading_name, or the system does
not support constructing the requested equivalent name, for
implementation-specific reasons.

EXAMPLES
In the Initial Context supporting XFN enterprise policies, a user jsmith is able to name
one of her files relative to this context in several ways:

_myself/_fs/map.ps
_user/jsmith/_fs/map.ps
_orgunit/finance/_user/jsmith/_fs/map.ps

The first of these may be appealing to the user jsmith in her day-to-day operations. This
name is not, however, appropriate for her to use when referring the file in an electronic
mail message sent to a colleague. The second of these names would be appropriate if the
colleague were in the same organizational unit, and the third appropriate for anyone in
the same enterprise.

When the following sequence of instructions is executed by the user jsmith in the organi-
zational unit finance, enterprise_wide_name would contain the composite name
_orgunit/finance/_user/jsmith/_fs/map.ps:

modified 22 Nov 1996 SunOS 5.6 3N-583
FN_string_t* namestr =
    fn_string_from_str((const unsigned char*)"_myself/_fs/map.ps");
FN_composite_name_t* name = fn_composite_name_from_string(namestr);
FN_string_t* org_lead =
    fn_string_from_str((const unsigned char*)"_orgunit");
FN_status_t* status = fn_status_create();
FN_composite_name_t* enterprise_wide_name =
    fn_composite_name_from_string(namestr);
FN_ctx_t* init_ctx = fn_ctx_handle_from_initial(status);
    /* check status of from_initial() */
enterprise_wide_name = fn_ctx_equivalent_name(init_ctx, name, org_lead, status);

When the following sequence of instructions is executed by the user jsmith in the organi-
zational unit finance, shortest_name would contain the composite name
_myself/_fs/map.ps:
FN_string_t* namestr =
    fn_string_from_str((const unsigned char*)
    "_orgunit/finance/_user_jsmith/_fs/map.ps");
FN_composite_name_t* name = fn_composite_name_from_string(namestr);
FN_string_t* mylead = fn_string_from_str((const unsigned char*)"_myself");
FN_status_t* status = fn_status_create();
FN_composite_name_t* shortest_name =
    fn_composite_name_from_string(namestr);
FN_ctx_t* init_ctx = fn_ctx_handle_from_initial(status);
    /* check status of from_initial() */
shortest_name = fn_ctx_equivalent_name(init_ctx, name, mylead, status);

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
FN_composite_name_t(3N), FN_ctx_t(3N), FN_status_t(3N), FN_string_t(3N),
xfn_status_codes(3N), attributes(5)
NAME fn_ctx_get_ref — return a context’s reference

SYNOPSIS cc [ flag ...] file ... -lxfn [ library ...]
#include <xfn/xfn.h>
FN_ref_t *fn_ctx_get_ref(const FN_ctx_t *ctx, FN_status_t *status);

DESCRIPTION This operation returns a reference to the supplied context object.

RETURN VALUE fn_ctx_get_ref() returns a pointer to an FN_ref_t object if the operation succeeds, it returns 0 if the operation fails.

ERRORS fn_ctx_get_ref() sets status as described in FN_status_t(3N) and xfn_status_codes(3N). The following status code is of particular relevance to this operation:

FN_E_OPERATION_NOT_SUPPORTED
Using the fn_ctx_get_ref() operation on the Initial Context returns this status code.

APPLICATION USAGE fn_ctx_get_ref() cannot be used on the Initial Context. fn_ctx_get_ref() can be used on contexts bound in the Initial Context (in other words, the bindings in the Initial Context have references).

If the context handle was created earlier using the fn_ctx_handle_from_ref() operation, the reference returned by the fn_ctx_get_ref() operation may not necessarily be exactly the same in content as that originally supplied. For example, fn_ctx_handle_from_ref() may construct the context handle from one address from the list of addresses. The context implementation may return with a call to fn_ctx_get_ref() only that address, or a more complete list of addresses than what was supplied in fn_ctx_handle_from_ref().

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level Safe.</td>
<td>Safe.</td>
</tr>
</tbody>
</table>

SEE ALSO FN_ctx_t(3N), FN_ref_t(3N), FN_status_t(3N), fn_ctx_handle_from_initial(3N), fn_ctx_handle_from_ref(3N), xfn_status_codes(3N), xfn(3N), attributes(5)
NAME
fn_ctx_get_syntax_attrs – return syntax attributes associated with named context

SYNOPSIS
cc [ flag ...] file ... -lxfn [ library ...]
#include <xfn/xfn.h>
FN_attrset_t *fn_ctx_get_syntax_attrs(FN_ctx_t *ctx,
    const FN_composite_name_t *name, FN_status_t *status);

DESCRIPTION
Each context has an associated set of syntax-related attributes. This operation returns the syntax attributes associated with the context named by name relative to the context ctx.
The attributes must contain the attribute fn_syntax_type (FN_ID_STRING format). If the context supports a syntax that conforms to the XFN standard syntax model, fn_syntax_type is set to "standard" (ASCII attribute syntax) and the attribute set contains the rest of the relevant syntax attributes described in xfn_compound_names(3N).
This operation is different from other XFN attribute operations in that these syntax attributes could be obtained directly from the context. Attributes obtained through other XFN attribute operations may not necessarily be associated with the context; they may be associated with the reference of context, rather than the context itself (see xfn_attributes(3N)).

RETURN VALUE
fn_ctx_get_syntax_attrs() returns an attribute set if successful; it returns a NULL pointer (0) if the operation fails.

ERRORS
fn_ctx_get_syntax_attrs() sets status as described in FN_status_t(3N) and xfn_status_codes(3N).

APPLICATION USAGE
Implementations may choose to support other syntax types in addition to, or in place of, the XFN standard syntax model, in which case, the value of the fn_syntax_type attribute would be set to an implementation-specific string, and different or additional syntax attributes will be in the set.
Syntax attributes of a context may be generated automatically by a context, in response to fn_ctx_get_syntax_attrs(), or they may be created and updated using the base attribute operations. This is implementation-dependent.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe.</td>
</tr>
</tbody>
</table>

SEE ALSO
FN_attrset_t(3N), FN_composite_name_t(3N), FN_compound_name_t(3N), FN_ctx_t(3N), FN_status_t(3N), fn_attr_get(3N), fn_attr_multi_get(3N), xfn_compound_names(3N), xfn_attributes(3N), xfn_status_codes(3N), xfn(3N), attributes(5)
NAME  fn_ctx_handle_destroy – release storage associated with context handle

SYNOPSIS  cc [ flag … ] file … −lxfn [ library … ]
#include <xfn/xfn.h>

void fn_ctx_handle_destroy(FN_ctx_t *ctx);

DESCRIPTION  This operation destroys the context handle ctx and allows the implementation to free resources associated with the context handle. This operation does not affect the state of the context itself.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe.</td>
</tr>
</tbody>
</table>

SEE ALSO  FN_ctx_t(3N), fn_ctx_handle_from_initial(3N), fn_ctx_handle_from_ref(3N), xfn(3N), attributes(5)
NAME  fn_ctx_handle_from_initial — return a handle to the Initial Context

SYNOPSIS  
```cc [ flag ...] file ... -lxfn [ library ...]  
#include <xfn/xfn.h>  
FN_ctx_t *fn_ctx_handle_from_initial( unsigned int authoritative,  
FN_status_t *status);```

DESCRIPTION  This operation returns a handle to the caller’s Initial Context. On successful return, the handle points to a context which meets the specification of the XFN Initial Context (see \texttt{fns_initial_context(5)}).

\textit{authoritative} specifies whether the handle to the context returned should be authoritative with respect to information the context obtains from the naming service. When the flag is non-zero, subsequent operations on the context will access the most authoritative information. When \textit{authoritative} is 0, the handle to the context returned need not be authoritative.

RETURN VALUES  \texttt{fn_ctx_handle_from_initial()} returns a pointer to an \texttt{FN_ctx_t} object if the operation succeeds; it returns a NULL pointer (0) otherwise.

ERRORS  \texttt{fn_ctx_handle_from_initial()} sets only the status code portion of the status object \texttt{status}.

USAGE  Authoritiveness is determined by specific naming services. For example, in a naming service that supports replication using a master/slave model, the source of authoritative information would come from the master server. In some naming systems, bypassing the naming service cache may reach servers which provide the most authoritative information. The availability of an authoritative context might be lower due to the lower number of servers offering this service. For the same reason, it might also provide poorer performance than contexts that need not be authoritative.

Applications set \textit{authoritative} to 0 for typical day-to-day operations. Applications only set \textit{authoritative} to a non-zero value when they require access to the most authoritative information, possibly at the expense of lower availability and/or poorer performance.

It is implementation-dependent whether authoritiveness is transferred from one context to the next as composite name resolution proceeds. Getting an authoritative context handle to the Initial Context means that operations on bindings in the Initial Context are processed using the most authoritative information. Contexts referenced implicitly through an authoritative Initial Context (for example, through the use of composite names) may not necessarily themselves be authoritative.
ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO

FN_ctx_t(3N), FN_status_t(3N), fn_ctx_get_ref(3N), fn_ctx_handle_from_ref(3N), xfn(3N), xfn_status_codes(3N), attributes(5), fns_initial_context(5)

NOTES

The implementation of XFN in this Solaris release is based on the X/Open preliminary specification. It is likely that there will be minor changes to these interfaces to reflect changes in the final version of this specification. The next minor release of Solaris will offer binary compatibility for applications developed using the current interfaces. As the interfaces evolve toward standardization, it is possible that future releases of Solaris will require minor source code changes to applications that have been developed against the preliminary specification.
fn_ctx_handle_from_ref (3N)  
Network Functions

NAME  
fn_ctx_handle_from_ref — construct a handle to a context object using the given reference

SYNOPSIS  
cc [ flag ... ] file ... -lxfn [ library ... ]

#include <xfn/xfn.h>

FN_ctx_t *fn_ctx_handle_from_ref(const FN_ref_t *ref, unsigned int authoritative,
FN_status_t *status);

DESCRIPTION  
This operation creates a handle to an FN_ctx_t object using an FN_ref_t object for that context.

authoritative specifies whether the handle to the context returned should be authoritative with respect to information the context obtains from the naming service. When the flag is non-zero, subsequent operations on the context will access the most authoritative information. When authoritative is 0, the handle to the context returned need not be authoritative.

RETURN VALUES  
This operation returns a pointer to an FN_ctx_t object if the operation succeeds; otherwise, it returns a NULL pointer (0).

ERRORS  
fn_ctx_handle_from_ref() sets status as described in FN_status_t(3N) and xfn_status_codes(3N). The following status code is of particular relevance to this operation:

FN_E_NO_SUPPORTED_ADDRESS  
A context object could not be constructed from a particular reference. The reference contained no address type over which the context interface was supported.

USAGE  
Authoritativeness is determined by specific naming services. For example, in a naming service that supports replication using a master/slave model, the source of authoritative information would come from the master server. In some naming systems, bypassing the naming service cache may reach servers which provide the most authoritative information. The availability of an authoritative context might be lower due to the lower number of servers offering this service. For the same reason, it might also provide poorer performance than contexts that need not be authoritative.

Applications set authoritative to 0 for typical day-to-day operations. Applications only set authoritative to a non-zero value when they require access to the most authoritative information, possibly at the expense of lower availability and/or poorer performance.

To control the authoritativeness of the target context, the application first resolves explicitly to the target context using fn_ctx_lookup(3N). It then uses fn_ctx_handle_from_ref() with the appropriate authoritative argument to obtain a handle to the context. This returns a handle to a context with the specified authoritativeness. The application then uses the XFN operations, such as lookup and list, with this context handle.
It is implementation-dependent whether authoritativeness is transferred from one context to the next as composite name resolution proceeds. The application should use the approach recommended above to achieve the desired level of authoritativeness on a per context basis.

**ATTRIBUTES**

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**SEE ALSO**

FN_ctx_t(3N), FN_ref_t(3N), FN_status_t(3N), fn_ctx_get_ref(3N), fn_ctx_handle_destroy(3N), fn_ctx_lookup(3N), xfn(3N), xfn_status_codes(3N), attributes(5), fns_references(5)

**NOTES**

The implementation of XFN in this Solaris release is based on the X/Open preliminary specification. It is likely that there will be minor changes to these interfaces to reflect changes in the final version of this specification. The next minor release of Solaris will offer binary compatibility for applications developed using the current interfaces. As the interfaces evolve toward standardization, it is possible that future releases of Solaris will require minor source code changes to applications that have been developed against the preliminary specification.
NAME  fn_ctx_list_bindings, FN_bindinglist_t, fn_bindinglist_next, fn_bindinglist_destroy – list the atomic names and references bound in a context

SYNOPSIS  cc [ flag .. ] file .. –lxfn [ library .. ]
#include <xfn/xfn.h>
FN_bindinglist_t *fn_ctx_list_bindings(FN_ctx_t *ctx,
    const FN_composite_name_t *name, FN_status_t *status);
FN_string_t *fn_bindinglist_next(FN_bindinglist_t *bl, FN_ref_t **ref,
    FN_status_t *status);
void fn_bindinglist_destroy(FN_bindinglist_t *bl, FN_status_t *status);

DESCRIPTION  This set of operations is used to list the names and bindings in the context named by name relative to the context ctx. Note that name must name a context. If the intent is to list the contents of ctx, name should be an empty composite name.

The semantics of these operations are similar to those for listing names (see fn_ctx_list_names(3N)). In addition to a name string being returned, fn_bindinglist_next() also returns the reference of the binding for each member of the enumeration.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  FN_composite_name_t(3N), FN_ctx_t(3N), FN_ref_t(3N), FN_status_t(3N), FN_string_t(3N), fn_ctx_list_names(3N), xfn(3N), xfn_status_codes(3N), attributes(5)

NOTES  The implementation of XFN in this Solaris release is based on the X/Open preliminary specification. It is likely that there will be minor changes to these interfaces to reflect changes in the final version of this specification. The next minor release of Solaris will offer binary compatibility for applications developed using the current interfaces. As the interfaces evolve toward standardization, it is possible that future releases of Solaris will require minor source code changes to applications that have been developed against the preliminary specification.
NAME
fn_ctx_list_names, FN_namelist_t, fn_namelist_next, fn_namelist_destroy – list the atomic names bound in a context

SYNOPSIS
cc [flag …] file … -lxfn [ library …]
#include <xfn/xfn.h>

FN_namelist_t *fn_ctx_list_names(FN_ctx_t *ctx, const FN_composite_name_t *name, FN_status_t *status);
FN_string_t *fn_namelist_next(FN_namelist_t *nl, FN_status_t *status);
void fn_namelist_destroy(FN_namelist_t *nl, FN_status_t *status);

DESCRIPTION
This set of operations is used to list the names bound in the target context named name relative to the context ctx. Note that name must name a context. If the intent is to list the contents of ctx, name should be an empty composite name.

The call to fn_ctx_list_names() initiates the enumeration process. It returns a handle to an FN_namelist_t object that can be used to enumerate the names in the target context.

The operation fn_namelist_next() returns the next name in the enumeration identified by nl and updates nl to indicate the state of the enumeration. Successive calls to fn_namelist_next() using nl return successive names in the enumeration and further update the state of the enumeration. fn_namelist_next() returns a NULL pointer (0) when the enumeration has been completed.

fn_namelist_destroy() is used to release resources used during the enumeration. This may be invoked at any time to terminate the enumeration.

RETURN VALUES
fn_ctx_list_names() returns a pointer to an FN_namelist_t object if the enumeration is successfully initiated; otherwise it returns a NULL pointer (0).

fn_namelist_next() returns a NULL pointer (0) if no more names can be returned in the enumeration.

In the case of a failure, these operations return in status a code indicating the nature of the failure.

ERRORS
Each successful call to fn_namelist_next() returns a name and sets status to FN_SUCCESS.

When fn_namelist_next() returns a NULL pointer (0), it indicates that no more names can be returned. status is set in the following way:

FN_SUCCESS
The enumeration has completed successfully.

modified 13 Dec 1996 SunOS 5.6 3N-593
**FN_E_INVALID_ENUM_HANDLE**
The supplied enumeration handle is not valid. Possible reasons could be that the handle was from another enumeration, or the context being enumerated no longer accepts the handle (due to such events as handle expiration or updates to the context).

**FN_E_PARTIAL_RESULT**
The enumeration is not yet complete but cannot be continued.

Other status codes, such as FN_E_COMMUNICATION_FAILURE, are also possible in calls to `fn_ctx_list_names()`, `fn_namelist_next()`, and `fn_namelist_destroy()`. These functions set `status` for these other status codes as described in `FN_status_t(3N)` and `xfn_status_codes(3N)`.

**USAGE**
The names enumerated using `fn_namelist_next()` are not ordered in any way. There is no guaranteed relation between the order in which names are added to a context and the order of names obtained by enumeration. The specification does not guarantee that any two series of enumerations will return the names in the same order.

When a name is added to or removed from a context, this may or may not invalidate the enumeration handle that the client holds for that context. If the enumeration handle becomes invalid, the status code FN_E_INVALID_ENUM_HANDLE is returned in `status`. If the enumeration handle remains valid, the update may or may not be visible to the client.

In addition, there may be a relationship between the `ctx` argument supplied to `fn_ctx_list_names()` and the FN_namelist_t object it returns. For example, some implementations may store the context handle `ctx` within the FN_namelist_t object for subsequent `fn_namelist_next()` calls. In general, a `fn_ctx_handle_destroy(3N)` should not be invoked on `ctx` until the enumeration has terminated.

**EXAMPLES**
The following code fragment illustrates how the list names operations may be used:

```c
extern FN_string_t *user_input;
FN_ctx_t *ctx;
FN_composite_name_t *target_name = fn_composite_name_from_string(user_input);
FN_status_t *status = fn_status_create();
FN_string_t *name;
FN_namelist_t *nl;
ctx = fn_ctx_handle_from_initial(status);
/* error checking on 'status' */
if ((nl=fn_ctx_list_names(ctx, target_name, status)) == 0) {
    /* report 'status' and exit */
}
while (name=fn_namelist_next(nl, status)) {
    /* do something with 'name' */
    fn_string_destroy(name);
}
```

3N-594 SunOS 5.6 modified 13 Dec 1996
}  
/* check 'status' for reason for end of enumeration and report if necessary */

/* clean up */
fn_namelist_destroy(nl, status);

/* report 'status' */

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  
FN_composite_name_t(3N), FN_ctx_t(3N), FN_status_t(3N), FN_string_t(3N), fn_ctx_handle_destroy(3N), xfn(3N), xfn_status_codes(3N), attributes(5)

NOTES  
The implementation of XFN in this Solaris release is based on the X/Open preliminary specification. It is likely that there will be minor changes to these interfaces to reflect changes in the final version of this specification. The next minor release of Solaris will offer binary compatibility for applications developed using the current interfaces. As the interfaces evolve toward standardization, it is possible that future releases of Solaris will require minor source code changes to applications that have been developed against the preliminary specification.
NAME    fn_ctx_lookup – look up name in context

SYNOPSIS cc [ flag ...] file ... -lxfn [ library ...]
#include <xfn/xfn.h>
FN_ref_t *fn_ctx_lookup(FN_ctx_t *ctx, const FN_composite_name_t *name,
                        FN_status_t *status);

DESCRIPTION This operation returns the reference bound to name relative to the context ctx.

RETURN VALUE If the operation succeeds, the fn_ctx_lookup() function returns a handle to the reference bound to name. Otherwise, 0 is returned and status is set appropriately.

ERRORS fn_ctx_lookup() sets status as described FN_status_t(3N) and xfn_status_codes(3N).

APPLICATION USAGE Some naming services may not always have reference information for all names in their contexts; for such names, such naming services may return a special reference whose type indicates that the name is not bound to any address. This reference may be name service specific or it may be the conventional NULL reference defined in the X/Open registry. See fns_references(5).

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe.</td>
</tr>
</tbody>
</table>

SEE ALSO FN_composite_name_t(3N), FN_ctx_t(3N), FN_ref_t(3N), FN_status_t(3N),
            fns_references(5), xfn_status_codes(3N), xfn(3N), attributes(5)
NAME  fn_ctx_lookup_link – look up the link reference bound to a name

SYNOPSIS  cc [ flag ...] file ... -lxfn [ library ...]

  #include <xfn/xfn.h>
  FN_ref_t *fn_ctx_lookup_link(FN_ctx_t *ctx, const FN_composite_name_t *name,
                               FN_status_t *status);

DESCRIPTION  This operation returns the XFN link bound to name. The terminal atomic part of name must be bound to an XFN link.

  The normal fn_ctx_lookup(3N) operation follows all links encountered, including any bound to the terminal atomic part of name. This operation differs from the normal lookup in that when the terminal atomic part of name is an XFN link, this link is not followed, and the operation returns the link.

RETURN VALUES  If fn_ctx_lookup_link() fails, a NULL pointer (0) is returned.

ERRORS  fn_ctx_lookup_link() sets status as described in FN_status_t(3N) and xfn_status_codes(3N). Of special relevance for fn_ctx_lookup_link() is the following status code:

  FN_E_MALFORMED_LINK

  name resolved to a reference that was not a link.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  FN_composite_name_t(3N), FN_ctx_t(3N), FN_ref_t(3N), FN_status_t(3N), fn_ctx_lookup(3N), xfn(3N), xfn_links(3N), xfn_status_codes(3N), attributes(5)

NOTES  The implementation of XFN in this Solaris release is based on the X/Open preliminary specification. It is likely that there will be minor changes to these interfaces to reflect changes in the final version of this specification. The next minor release of Solaris will offer binary compatibility for applications developed using the current interfaces. As the interfaces evolve toward standardization, it is possible that future releases of Solaris will require minor source code changes to applications that have been developed against the preliminary specification.

modified 13 Dec 1996  SunOS 5.6  3N-597
fn_ctx_rename(3N) Network Functions

NAME
fn_ctx_rename – rename the name of a binding

SYNOPSIS
cc [ flag ...] file ... -lxfn [ library ...]
#include <xfn/xfn.h>

int fn_ctx_rename(FN_ctx_t *ctx, const FN_composite_name_t *oldname,
const FN_composite_name_t *newname, unsigned int exclusive,
FN_status_t *status);

DESCRIPTION
The fn_ctx_rename() operation binds the reference currently bound to oldname relative to ctx, to the name newname, and unbinds oldname. newname is resolved relative to the target context (that named by all but the terminal atomic part of oldname).

If exclusive is 0, the operation overwrites any old binding of newname. If exclusive is nonzero, the operation fails if newname is already bound.

RETURN VALUES
fn_ctx_rename() returns 1 if the operation is successful, 0 otherwise.

ERRORS
fn_ctx_rename() sets status as described FN_status_t(3N) and xfn_status_codes(3N).

USAGE
The only restriction that XFN places on newname is that it be resolved relative to the target context. XFN does not specify further restrictions on newname. For example, in some implementations, newname might be restricted to be a name in the same naming system as the terminal component of oldname. In another implementation, newname might be restricted to be an atomic name.

Normal resolution always follows links. In an fn_ctx_rename() operation, resolution of oldname continues to the target context; the terminal atomic name is not resolved. If the terminal atomic name is bound to a link, the link is not followed and the operation binds newname to the link and unbinds the terminal atomic name of oldname.

In naming systems that support attributes and store the attributes along with the names, the unbind of the terminal atomic name of oldname also removes its associated attributes. It is implementation-dependent whether these attributes become associated with newname.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
FN_composite_name_t(3N), FN_ctx_t(3N), FN_ref_t(3N), FN_status_t(3N), fn_ctx_bind(3N), fn_ctx_unbind(3N), xfn(3N), xfn_status_codes(3N), attributes(5)

NOTES
The implementation of XFN in this Solaris release is based on the X/Open preliminary specification. It is likely that there will be minor changes to these interfaces to reflect changes in the final version of this specification. The next minor release of Solaris will offer binary compatibility for applications developed using the current interfaces. As the
interfaces evolve toward standardization, it is possible that future releases of Solaris will require minor source code changes to applications that have been developed against the preliminary specification.
NAME  FN_ctx_t — an XFN context

SYNOPSIS  cc [ flag ...] file ... -llxfn [ library ...]
#include <xfn/xfn.h>
FN_ctx_t *fn_ctx_handle_from_initial( unsigned int authoritative,
   FN_status_t *status);
FN_ctx_t *fn_ctx_handle_from_ref(const FN_ref_t *ref, unsigned int authoritative,
   FN_status_t *status);
FN_ref_t *fn_ctx_get_ref( const FN_ctx_t *ctx, FN_status_t *status);
void fn_ctx_handle_destroy(FN_ctx_t *ctx);
FN_ref_t *fn_ctx_lookup( FN_ctx_t *ctx, const FN_composite_name_t *name,
   FN_status_t *status);
FN_namelist_t *fn_ctx_list_names( FN_ctx_t *ctx, const FN_composite_name_t *name,
   FN_status_t *status);
FN_string_t *fn_namelist_next( FN_namelist_t *nl, FN_status_t *status);
void fn_namelist_destroy( FN_namelist_t *nl, FN_status_t *status);
FN_bindinglist_t *fn_ctx_list_bindings( FN_ctx_t *ctx,
   const FN_composite_name_t *name, FN_status_t *status);
FN_string_t *fn_bindinglist_next( FN_bindinglist_t *iter, FN_ref_t **ref,
   FN_status_t *status);
void fn_bindinglist_destroy( FN_bindinglist_t *iter_pos, FN_status_t *status);
int fn_ctx_bind( FN_ctx_t *ctx, const FN_composite_name_t *name,
   const FN_ref_t *ref, unsigned int exclusive, FN_status_t *status);
int fn_ctx_unbind( FN_ctx_t *ctx, const FN_composite_name_t *name,
   FN_status_t *status);
int fn_ctx_rename( FN_ctx_t *ctx, const FN_composite_name_t *oldname,
   const FN_composite_name_t *newname, unsigned int exclusive,
   FN_status_t *status);
FN_ref_t *fn_ctx_create_subcontext( FN_ctx_t *ctx,
   const FN_composite_name_t *name, FN_status_t *status);
int fn_ctx_destroy_subcontext( FN_ctx_t *ctx, const FN_composite_name_t *name,
   FN_status_t *status);
FN_ref_t *fn_ctx_lookup_link( FN_ctx_t *ctx, const FN_composite_name_t *name,
   FN_status_t *status);
FN_attrset_t *fn_ctx_get_syntax_attrs( FN_ctx_t *ctx,
   const FN_composite_name_t *name, FN_status_t *status);
DESCRIPTION

An XFN context consists of a set of name to reference bindings. An XFN context is represented by the type FN_ctx_t in the client interface. The operations for manipulating an FN_ctx_t object are described in detail in separate reference manual pages.

The following contains a brief summary of these operations:

- fn_ctx_handle_from_initial() returns a pointer to an Initial Context that provides a starting point for resolution of composite names. fn_ctx_handle_from_ref() returns a handle to an FN_ctx_t object using the given reference ref. fn_ctx_get_ref() returns the reference of the context ctx. fn_ctx_handle_destroy() releases the resources associated with the FN_ctx_t object ctx; it does not affect the state of the context itself.

- fn_ctx_lookup() returns the reference bound to name resolved relative to ctx.

- fn_ctx_list_names() is used to enumerate the atomic names bound in the context named by name resolved relative to ctx. fn_ctx_list_bindings() is used to enumerate the atomic names and their references in the context named by name resolved relative to ctx.

- fn_ctx_bind() binds the composite name name to a reference ref resolved relative to ctx.

- fn_ctx_unbind() unbinds name resolved relative to ctx. fn_ctx_rename() binds newname to the reference bound to oldname and unbinds oldname. oldname is resolved relative to ctx; newname is resolved relative to the target context.

- fn_ctx_create_subcontext() creates a new context with the given composite name name resolved relative to ctx. fn_ctx_destroy_subcontext() destroys the context named by name resolved relative to ctx.

Normal resolution always follows links. fn_ctx_lookup_link() looks up name relative to ctx, following links except for the last atomic part of name, which must be bound to an XFN link.

- fn_ctx_get_syntax_attrs() returns an attribute set containing attributes that describe a context’s syntax. name must name a context.

ERRORS

In each context operation, the caller supplies an FN_status_t object as a parameter. The called function sets this status object as described in FN_status_t(3N) and xfn_status_codes(3N).

USAGE

In most of the operations of the base context interface, the caller supplies a context and a composite name. The supplied name is always interpreted relative to the supplied context.

The operation may eventually be effected on a different context called the operation’s target context. Each operation has an initial resolution phase that conveys the operation to its target context, and the operation is then applied. The effect (but not necessarily the implementation) is that of doing a lookup on that portion of the name that represents the target context, and then invoking the operation on the target context. The contexts involved only in the resolution phase are called intermediate contexts.

Normal resolution of names in context operations always follows XFN links.
ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO

FN_attrset_t(3N), FN_composite_name_t(3N), FN_ref_t(3N), FN_status_t(3N),
fn_ctx_bind(3N), fn_ctx_create_subcontext(3N), fn_ctx_destroy_subcontext(3N),
fn_ctx_get_ref(3N), fn_ctx_get_syntax_attrs(3N), fn_ctx_handle_destroy(3N),
fn_ctx_handle_from_initial(3N), fn_ctx_handle_from_ref(3N),
fn_ctx_list_bindings(3N), fn_ctx_list_names(3N), fn_ctx_lookup(3N),
fn_ctx_lookup_link(3N), fn_ctx_rename(3N), fn_ctx_unbind(3N), xfn(3N),
xfn_links(3N), xfn_status_codes(3N), attributes(5)

NOTES

The implementation of XFN in this Solaris release is based on the X/Open preliminary
specification. It is likely that there will be minor changes to these interfaces to reflect
changes in the final version of this specification. The next minor release of Solaris will
offer binary compatibility for applications developed using the current interfaces. As the
interfaces evolve toward standardization, it is possible that future releases of Solaris will
require minor source code changes to applications that have been developed against the
preliminary specification.
Network Functions

NAME
fn_ctx_unbind – unbind a name from a context

SYNOPSIS
cc [ flag ...] file ... -lxfn [ library ...]
#include <xfn/xfn.h>

int fn_ctx_unbind(FN_ctx_t *ctx, const FN_composite_name_t *name, FN_status_t *status);

DESCRIPTION
This operation removes the terminal atomic name in name from the the target context — that named by all but the terminal atomic part of name.
This operation is successful even if the terminal atomic name was not bound in target context, but fails if any of the intermediate names are not bound. fn_ctx_unbind() is idempotent.

RETURN VALUE
The operation returns 1 if successful, and 0 otherwise.

ERRORS
fn_ctx_unbind() sets status as described in FN_status_t and xfn_status_codes(3N).
Certain naming systems may disallow unbinding a name if the name is bound to an existing context in order to avoid orphan contexts that cannot be reached via any name. In such situations, the status code FN_E_OPERATION_NOT_SUPPORTED is returned.

APPLICATION USAGE
In naming systems that support attributes, and store the attributes along with the names, the unbind operation removes the name and its associated attributes.
Normal resolution always follows links. In an fn_ctx_unbind() operation, resolution of name continues to the target context; the terminal atomic name is not resolved. If the terminal atomic name is bound to a link, the link is not followed and the link itself is unbound from the terminal atomic name.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe.</td>
</tr>
</tbody>
</table>

SEE ALSO
FN_composite_name_t(3N), FN_ctx_t(3N), FN_ref_t(3N), FN_status_t(3N), fn_ctx_bind(3N), fn_ctx_lookup(3N), xfn_status_codes(3N), xfn(3N), attributes(5)

modified 30 Dec 1996

SunOS 5.6

3N-603
NAME  FN_identifier_t – an XFN identifier

DESCRIPTION  Identifiers are used to identify reference types and address types in an XFN reference, and to identify attributes and their syntax in the attribute operations. An XFN identifier consists of an unsigned int, which determines the format of identifier, and the actual identifier, which is expressed as a sequence of octets. The representation of this structure is defined by XFN as follows:

typedef struct {
    unsigned int format;
    size_t length;
    void *contents;
} FN_identifier_t;

XFN defines a small number of standard forms for identifiers:

FN_ID_STRING  The identifier is an ASCII string (ISO 646).
FN_ID_DCE_UUID  The identifier is an OSF DCE UUID in string representation. (See the X/Open DCE RPC.)
FN_ID_ISO_OID_STRING  The identifier is an ISO OID in ASN.1 dot-separated integer list string format. (See the ISO ASN.1.)
FN_ID_ISO_OID_BER  The identifier is an ISO OID in ASN.1 Basic Encoding Rules (BER) format. (See the ISO BER.)

FILES  #include <xfn/xfn.h>

SEE ALSO  FN_attribute_t(3N), FN_ref_addr_t(3N), FN_ref_t(3N), xfn(3N)

NOTES  The implementation of XFN in this Solaris release is based on the X/Open preliminary specification. It is likely that there will be minor changes to these interfaces to reflect changes in the final version of this specification. The next minor release of Solaris will offer binary compatibility for applications developed using the current interfaces. As the interfaces evolve toward standardization, it is possible that future releases of Solaris will require minor source code changes to applications that have been developed against the preliminary specification.
### NAME
fnmatch – match filename or path name

### SYNOPSIS
```c
#include <fnmatch.h>
int fnmatch(const char *pattern, const char *string, int flags);
```

### DESCRIPTION
The `fnmatch()` function matches patterns as described on the `fnmatch(5)` manual page. It checks the `string` argument to see if it matches the `pattern` argument.

The `flags` argument modifies the interpretation of `pattern` and `string`. It is the bitwise inclusive OR of zero or more of the following flags defined in the header `<fnmatch.h>`.

- **FNM_PATHNAME**
  - If set, a slash (/) character in `string` will be explicitly matched by a slash in `pattern`; it will not be matched by either the asterisk (*) or question-mark (?) special characters, nor by a bracket ([ ]) expression.
  - If not set, the slash character is treated as an ordinary character.

- **FNM_NOESCAPE**
  - If not set, a backslash character (\) in `pattern` followed by any other character will match that second character in `string`. In particular, “\" will match a backslash in `string`.
  - If set, a backslash character will be treated as an ordinary character.

- **FNM_PERIOD**
  - If set, a leading period in `string` will match a period in `pattern`; where the location of “leading” is indicated by the value of `FNM_PATHNAME`:
    - If `FNM_PATHNAME` is set, a period is “leading” if it is the first character in `string` or if it immediately follows a slash.
    - If `FNM_PATHNAME` is not set, a period is “leading” only if it is the first character of `string`.
  - If not set, no special restrictions are placed on matching a period.

### RETURN VALUES
The following values are returned:

- **0**
  - `string` matches the pattern specified by `pattern`.
- **FNM_NOMATCH**
  - there is no match. **FNM_NOMATCH** is defined in the header `<fnmatch.h>`.
- **non-zero**
  - an error has occurred.
The `fnmatch()` function has two major uses. It could be used by an application or utility that needs to read a directory and apply a pattern against each entry. The `find(1)` utility is an example of this. It can also be used by the `pax(1)` utility to process its *pattern* operands, or by applications that need to match strings in a similar manner.

The name `fnmatch()` is intended to imply *filename* match, rather than *pathname* match. The default action of this function is to match filenames, rather than path names, since it gives no special significance to the slash character. With the `FNM_PATHNAME` flag, `fnmatch()` does match path names, but without tilde expansion, parameter expansion, or special treatment for period at the beginning of a filename.

### Attributes

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

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe with exceptions</td>
</tr>
<tr>
<td>CSI</td>
<td>Enabled</td>
</tr>
</tbody>
</table>

### See Also

`find(1)`, `pax(1)`, `glob(3C)`, `setlocale(3C)`, `wordexp(3C)`, `attributes(5)`, `fnmatch(5)`

### Notes

`fnmatch()` can be used safely in multi-threaded applications as long as `setlocale(3C)` is not being called to change the locale.
Network Functions

NAME

FN_ref_addr_t, fn_ref_addr_create, fn_ref_addr_destroy, fn_ref_addr_copy, fn_ref_addr_assign, fn_ref_addr_type, fn_ref_addr_length, fn_ref_addr_data, fn_ref_addr_description – an address in an XFN reference

SYNOPSIS

cc [ flag ... ] file ... -lxfn [ library ... ]
#include <xfn/xfn.h>
FN_ref_addr_t *fn_ref_addr_create(const FN_identifier_t *type, size_t length, const void *data);
void fn_ref_addr_destroy(FN_ref_addr_t *addr);
FN_ref_addr_t *fn_ref_addr_copy(const FN_ref_addr_t *addr);
FN_ref_addr_t *fn_ref_addr_assign(FN_ref_addr_t *dst, const FN_ref_addr_t *src);
const FN_identifier_t *fn_ref_addr_type(const FN_ref_addr_t *addr);
size_t fn_ref_addr_length(const FN_ref_addr_t *addr);
const void * fn_ref_addr_data(const FN_ref_addr_t *addr);
FN_string_t *fn_ref_addr_description(const FN_ref_addr_t *addr, unsigned int detail, unsigned int *more_detail);

DESCRIPTION

An XFN reference is represented by the type FN_ref_t. An object of this type contains a reference type and a list of addresses. Each address in the list is represented by an object of type FN_ref_addr_t. An address consists of an opaque data buffer and a type field, of type FN_identifier_t.

fn_ref_addr_create() creates and returns an address with the given type and data. length indicates the size of the data. fn_ref_addr_destroy() releases the storage associated with the given address. fn_ref_addr_copy() returns a copy of the given address object. fn_ref_addr_assign() makes a copy of the address pointed to by src and assigns it to dst, releasing any old contents of dst. A pointer to the same object as dst is returned.

fn_ref_addr_type() returns the type of the given address. fn_ref_addr_length() returns the size of the address in bytes. fn_ref_addr_data() returns the contents of the address.

fn_ref_addr_description() returns the implementation-defined textual description of the address. It takes as arguments a number, detail, and a pointer to a number, more_detail. detail specifies the level of detail for which the description should be generated; the higher the number, the more detail is to be provided. If more_detail is 0, it is ignored. If more_detail is non-zero, it is set by the description operation to indicate the next level of detail available, beyond that specified by detail. If no higher level of detail is available, more_detail is set to detail.

USAGE

The address type of an FN_ref_addr_t object is intended to identify the mechanism that should be used to reach the object using that address. The client must interpret the contents of the opaque data buffer of the address based on the type of the address, and on the type of the reference that the address is in. However, this interpretation is intended to occur below the application layer. Most applications developers should not have to manipulate the contents of either address or reference objects themselves. These

modified 13 Dec 1996

SunOS 5.6

3N-607
interfaces would generally be used within service libraries. Multiple addresses in a single reference are intended to identify multiple communication endpoints for the same conceptual object. Multiple addresses may arise for various reasons, such as the object offering interfaces over more than one communication mechanism.

Manipulation of addresses using the operations described in this manual page does not affect their representation in the underlying naming system. Changes to addresses in the underlying naming system can only be effected through the use of the interfaces described in \texttt{FN_ctx_t(3N)}.

\textbf{ATTRIBUTES}\ See \texttt{attributes(5)} for descriptions of the following attributes:

\begin{tabular}{|l|l|}
\hline
ATTRIBUTE TYPE & ATTRIBUTE VALUE \\
\hline
MT-Level & MT-Safe \\
\hline
\end{tabular}

\textbf{SEE ALSO} \ \texttt{FN_ctx_t(3N)}, \texttt{FN_identifier_t(3N)}, \texttt{FN_ref_t(3N)}, \texttt{FN_string_t(3N)}, \texttt{xfn(3N)}, \texttt{attributes(5)}

\textbf{NOTES} \ The implementation of XFN in this Solaris release is based on the X/Open preliminary specification. It is likely that there will be minor changes to these interfaces to reflect changes in the final version of this specification. The next minor release of Solaris will offer binary compatibility for applications developed using the current interfaces. As the interfaces evolve toward standardization, it is possible that future releases of Solaris will require minor source code changes to applications that have been developed against the preliminary specification.
**NAME**

FN_ref_t, fn_ref_create, fn_ref_destroy, fn_ref_copy, fn_ref_assign, fn_ref_type,
fn_ref_addrcount, fn_ref_first, fn_ref_next, fn_ref-prepend_addr, fn_ref_append_addr,
fn_ref_insert_addr, fn_ref_delete_addr, fn_ref_delete_all, fn_ref_create_link,
fn_ref_is_link, fn_ref_link_name, fn_ref_description – an XFN reference

**SYNOPSIS**

cc [ flag ... ] file ... -lxfn [ library ... ]

#include <xfn/xfn.h>

FN_ref_t *fn_ref_create(const FN_identifier_t *ref_type);
void fn_ref_destroy(FN_ref_t *ref);
FN_ref_t *fn_ref_copy(const FN_ref_t *ref);
FN_ref_t *fn_ref_assign(FN_ref_t *dst, const FN_ref_t *src);
const FN_identifier_t *fn_ref_type(const FN_ref_t *ref);
unsigned int fn_ref_addrcount(const FN_ref_t *ref);
const FN_ref_addr_t *fn_ref_first(const FN_ref_t *ref, void **iter_pos);
const FN_ref_addr_t *fn_ref_next(const FN_ref_t *ref, void **iter_pos);
int fn_ref-prepend_addr(FN_ref_t *ref, const FN_ref_addr_t *addr);
int fn_ref_append_addr(FN_ref_t *ref, const FN_ref_addr_t *addr);
int fn_ref_insert_addr(FN_ref_t *ref, void **iter_pos, const FN_ref_addr_t *addr);
int fn_ref_delete_addr(FN_ref_t *ref, void **iter_pos);
int fn_ref_delete_all(FN_ref_t *ref);
FN_ref_t *fn_ref_create_link( const FN_composite_name_t *link_name);
int fn_ref_is_link(const FN_ref_t *ref);
FN_composite_name_t *fn_ref_link_name( const FN_ref_t *link_ref);
FN_string_t *fn_ref_description(const FN_ref_t *ref, unsigned int detail,
  unsigned int *more_detail);

**DESCRIPTION**

An XFN reference is represented by the type FN_ref_t. An object of this type contains a
reference type and a list of addresses. The ordering in this list at the time of binding
might not be preserved when the reference is returned upon lookup.

The reference type is represented by an object of type FN_identifier_t. The reference
type is intended to identify the class of object referenced. XFN does not dictate the precise
use of this.

Each address is represented by an object of type FN_ref_addr_t.

fn_ref_create() creates a reference with no address, using ref_type as its reference type.
Addresses can be added later to the reference using the functions described below.
fn_ref_destroy() releases the storage associated with ref. fn_ref_copy() creates a copy of
ref and returns it. fn_ref_assign() creates a copy of src and assigns it to dst, releasing any
old contents of dst. A pointer to the same object as dst is returned.

modified 13 Dec 1996 SunOS 5.6 3N-609
fn_ref_addrcount() returns the number of addresses in the reference ref.
fn_ref_first() returns the first address in ref and sets iter_pos to be after the address. It returns 0 if there is no address in the list. fn_ref_next() returns the address following iter_pos in ref and sets iter_pos to be after the address. If the iteration marker iter_pos is at the end of the sequence, fn_ref_next() returns 0.

fn_ref_prepend_addr() adds addr to the front of the list of addresses in ref.
fn_ref_append_addr() adds addr to the end of the list of addresses in ref.
fn_ref_insert_addr() adds addr to ref before iter_pos and sets iter_pos to be immediately after the new reference added. fn_ref_delete_addr() deletes the address located before iter_pos in the list of addresses in ref and sets iter_pos back one address.
fn_ref_delete_all() deletes all addresses in ref.

fn_ref_create_link() creates a reference using the given composite name link_name as an address. fn_ref_is_link() tests if ref is a link. It returns 1 if it is; 0 if it is not.

fn_ref_link_name() returns the composite name stored in a link reference. It returns 0 if link_ref is not a link.

fn_ref_description() returns a string description of the given reference. It takes as argument an integer, detail, and a pointer to an integer, more_detail. detail specifies the level of detail for which the description should be generated; the higher the number, the more detail is to be provided. If more_detail is 0, it is ignored. If more_detail is non-zero, it is set by the description operation to indicate the next level of detail available, beyond that specified by detail. If no higher level of detail is available, more_detail is set to detail.

RETURN VALUES
The following operations return 1 if the operation succeeds, 0 if the operation fails:

fn_ref_prepend_addr()
fn_ref_append_addr()
fn_ref_insert_addr()
fn_ref_delete_addr()
fn_ref_delete_all()

USAGE
The reference type is intended to identify the class of object referenced. XFN does not dictate the precise use of this.

Multiple addresses in a single reference are intended to identify multiple communication endpoints for the same conceptual object. Multiple addresses may arise for various reasons, such as the object offering interfaces over more than one communication mechanism.

The client must interpret the contents of a reference based on the type of the addresses and the type of the reference. However, this interpretation is intended to occur below the application layer. Most applications developers should not have to manipulate the contents of either address or reference objects themselves. These interfaces would generally be used within service libraries.

Manipulation of references using the operations described in this manual page does not affect their representation in the underlying naming system. Changes to references in the underlying naming system can only be effected through the use of the interfaces.
described in FN_ctx_t(3N).

**ATTRIBUTES**

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**SEE ALSO**

FN_composite_name_t(3N), FN_ctx_t(3N), FN_identifier_t(3N), FN_ref_addr_t(3N), FN_string_t(3N), fn_ctx_lookup(3N), fn_ctx_lookup_link(3N), xfn(3N), xfn_links(3N), attributes(5)

**NOTES**

The implementation of XFN in this Solaris release is based on the X/Open preliminary specification. It is likely that there will be minor changes to these interfaces to reflect changes in the final version of this specification. The next minor release of Solaris will offer binary compatibility for applications developed using the current interfaces. As the interfaces evolve toward standardization, it is possible that future releases of Solaris will require minor source code changes to applications that have been developed against the preliminary specification.
NAME
FN_search_control_t, fn_search_control_create, fn_search_control_destroy,
fn_search_control_copy, fn_search_control_assign, fn_search_control_scope,
fn_search_control_follow_links, fn_search_control_max_names,
fn_search_control_return_ref, fn_search_control_return_attr_ids – options for attribute search

SYNOPSIS
#include <xfn/xfn.h>

FN_search_control_t *fn_search_control_create(unsigned int scope,
    unsigned int follow_links,
    unsigned int max_names,
    unsigned int return_ref,
    const FN_attrset_t *return_attr_ids,
    unsigned int *status);

void fn_search_control_destroy(FN_search_control_t *scontrol);

FN_search_control_t *fn_search_control_copy(const FN_search_control_t *scontrol);

FN_search_control_t *fn_search_control_assign(FN_search_control_t *dst,
    const FN_search_control_t *src);

unsigned int fn_search_control_scope(const FN_search_control_t *scontrol);

unsigned int fn_search_control_follow_links(const FN_search_control_t *scontrol);

unsigned int fn_search_control_max_names(const FN_search_control_t *scontrol);

unsigned int fn_search_control_return_ref(const FN_search_control_t *scontrol);

const FN_attrset_t *fn_search_control_return_attr_ids(
    const FN_search_control_t *scontrol);

DESCRIPTION
The FN_search_control_t object is used to specify options for the attribute search operation fn_attr_ext_search(3N).

fn_search_control_create() creates an FN_search_control_t object using information in
scope, follow_links, max_names, return_ref, and return_attr_ids to set the search options. If
the operation succeeds, fn_search_control_create() returns a pointer to an
FN_search_control_t object; otherwise, it returns a NULL pointer.

The scope of the search, scope, is either the named object, the named context, the named
context and its subcontexts, or the named context and a context implementation defined
set of subcontexts. The values for scope are:

FN_SEARCH_NAMED_OBJECT
    Search just the given named object.

FN_SEARCH_ONE_CONTEXT
    Search just the given context.
**FN_SEARCH_SUBTREE**
Search given context and all its subcontexts.

**FN_SEARCHCONSTRAINED_SUBTREE**
Search given context and its subcontexts as constrained by the context-specific policy in place at the named context.

`follow_links` further defines the scope and nature of the search. If `follow_links` is nonzero, the search follows XFN links. If `follow_links` is 0, XFN links are not followed. See `fn_attr_ext_search(3N)` for more detail about how XFN links are treated.

`max_names` specifies the maximum number of names to return in an `FN_ext_searchlist_t(3N)` enumeration (see `fn_attr_ext_search(3N)`). The names of all objects whose attributes satisfy the filter are returned when `max_names` is 0.

If `return_ref` is non-zero, the reference bound to the named object is returned with the object’s name by `fn_ext_searchlist_next(3N)` (see `fn_attr_ext_search(3N)`). If `return_ref` is 0, the reference is not returned.

Attribute identifiers and values associated with named objects that satisfy the filter may be returned by `fn_ext_searchlist_next(3N)`. The attributes returned are those listed in `return_attr_ids`. If the value of `return_attr_ids` is 0, all attributes are returned. If `return_attr_ids` is an empty `FN_attrset_t` object (see `FN_attrset_t(3N)`), no attributes are returned. Any attribute values in `return_attr_ids` are ignored; only the attribute identifiers are relevant for this operation.

`fn_attr_ext_search(3N)` interprets a value of 0 for the search control argument as a default search control which has the following option settings:

- `scope` = `FN_SEARCH_ONE_CONTEXT`
- `follow_links` = 0 (do not follow links)
- `max_names` = 0 (return all named objects that match filter)
- `return_ref` = 0 (do not return the reference of the named object)
- `return_attr_ids` = an empty `FN_attrset_t` object (do not return any attributes of the named object)

`fn_search_control_destroy()` releases the storage associated with `scontrol`.

`fn_search_control_copy()` returns a copy of the search control `scontrol`.

`fn_search_control_assign()` makes a copy of the search control `src` and assigns it to `dst`, releasing the old contents of `dst`. A pointer to the same object as `dst` is returned.

`fn_search_control_scope()` returns the scope for the search.

`fn_search_control_follow_links()` returns non-zero if links are followed; 0 if not.

`fn_search_control_max_names()` returns the maximum number of names.

`fn_search_control_return_ref()` returns nonzero if the reference is returned; 0 if not.

`fn_search_control_return_attr_ids()` returns a pointer to the list of attributes; a NULL pointer indicates that all attributes and values are returned.
ERRORS

*fn_search_control_create()* returns a NULL pointer if the operation fails and sets status as follows:

**FN_E_SEARCH_INVALID_OPTION**
A supplied search option was invalid or inconsistent.

Other status codes are possible (see `xfn_status_codes(3N)`).

ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO

`FN_attrset_t(3N)`, `fn_attr_ext_search(3N)`, `xfn_status_codes(3N)`, `attributes(5)`
NAME
FN_search_filter_t, fn_search_filter_create, fn_search_filter_destroy,
fn_search_filter_copy, fn_search_filter_assign, fn_search_filter_expression,
fn_search_filter_arguments – filter expression for attribute search

SYNOPSIS
#include <xfn/xfn.h>
FN_search_filter_t *fn_search_filter_create(unsigned int *status,
const unsigned char *estr, ...);
void fn_search_filter_destroy(FN_search_filter_t *sfilter);
FN_search_filter_t *fn_search_filter_copy(const FN_search_filter_t *sfilter);
FN_search_filter_t *fn_search_filter_assign(FN_search_filter_t *dst,
const FN_search_filter_t *src);
const char *fn_search_filter_expression(const FN_search_filter_t *sfilter);
const void **fn_search_filter_arguments(const FN_search_filter_t *sfilter,
size_t *number_of_arguments);

DESCRIPTION
The FN_search_filter_t type is an expression that is evaluated against the attributes of
named objects bound in the scope of the search operation fn_attr_ext_search(3N). The
filter evaluates to TRUE or FALSE. If the filter is empty, it evaluates to TRUE. Names of
objects whose attribute values satisfy the filter expression are returned by the search
operation.

If the identifier in any subexpression of the filter does not exist as an attribute of an object,
then the innermost logical expression containing that identifier is FALSE. A subexpres-
sion that is only an attribute tests for the presence of the attribute; the subexpression
evaluates to TRUE if the attribute has been defined for the object and FALSE otherwise.

fn_search_filter_create() creates a search filter from the expression string estr and the
remaining arguments.

fn_search_filter_destroy() releases the storage associated with the search filter sfilter.

fn_search_filter_copy() returns a copy of the search filter sfilter.

fn_search_filter_assign() makes a copy of the search filter src and assigns it to dst, releas-
ing the old contents of dst. A pointer to the same object as dst is returned.

fn_search_filter_expression() returns the filter expression of sfilter.

fn_search_filter_arguments() returns an array of pointers to arguments supplied to the
filter constructor. number_of_arguments is set to the size of this array. The types of the
arguments are determined by the substitution tokens in the expression in sfilter.

BNF of Filter
Expression
<FilterExpr> ::= [ <Expr> ]
<Expr> ::= <Expr> "or" <Expr>
<Expr> "and" <Expr>
| "not" <Expr>
| "(" <Expr> ")"

modified 22 Nov 1996 SunOS 5.6 3N-615
### FN_search_filter_t (3N)

#### Specification of Filter Expression

The arguments to `fn_search_filter_create()` are a return status, an expression string, and a list of arguments. The string contains the filter expression with substitution tokens for the attributes, attribute values, strings, and identifiers that are part of the expression. The remaining list of arguments contains the attributes and values in the order of appearance of their corresponding substitution tokens in the expression. The arguments are of types `FN_attribute_t*`, `FN_attrvalue_t*`, `FN_string_t*`, or `FN_identifier_t*`. Any attribute values in an `FN_attribute_t*` type of argument are ignored; only the attribute identifier and attribute syntax are relevant. The argument type expected by each substitution token are listed in the following table.

<table>
<thead>
<tr>
<th>Token</th>
<th>Argument Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>%a</td>
<td>FN_attribute_t*</td>
</tr>
<tr>
<td>%v</td>
<td>FN_attrvalue_t*</td>
</tr>
<tr>
<td>%s</td>
<td>FN_string_t*</td>
</tr>
<tr>
<td>%i</td>
<td>FN_identifier_t*</td>
</tr>
</tbody>
</table>

#### Precedence

The following precedence relations hold in the absence of parentheses, in the order of lowest to highest:

### Network Functions

3N-616 SunOS 5.6 modified 22 Nov 1996
or
and
not
relational operators

These boolean and relational operators are left associative.

**Relational Operators**

Comparisons and ordering are specific to the syntax and/or rules of the supplied attribute.

Locale (code set, language, or territory) mismatches that occur during string comparisons and ordering operations are resolved in an implementation-dependent way. Relational operations that have ordering semantics may be used for strings of code sets in which ordering is meaningful, but is not of general use in internationalized environments.

An attribute that occurs in the absence of any relational operator tests for the presence of the attribute.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>==</td>
<td>The sub-expression is <strong>TRUE</strong> if at least one value of the specified attribute is equal to the supplied value.</td>
</tr>
<tr>
<td>!=</td>
<td>The sub-expression is <strong>TRUE</strong> if no values of the specified attribute equal the supplied value.</td>
</tr>
<tr>
<td>&gt;=</td>
<td>The sub-expression is <strong>TRUE</strong> if at least one value of the attribute is greater than or equal to the supplied value.</td>
</tr>
<tr>
<td>&gt;</td>
<td>The sub-expression is <strong>TRUE</strong> if at least one value of the attribute is greater than the supplied value.</td>
</tr>
<tr>
<td>&lt;=</td>
<td>The sub-expression is <strong>TRUE</strong> if at least one value of the attribute is less than or equal to the supplied value.</td>
</tr>
<tr>
<td>&lt;</td>
<td>The sub-expression is <strong>TRUE</strong> if at least one value of the attribute is less than the supplied value.</td>
</tr>
<tr>
<td>~</td>
<td>The sub-expression is <strong>TRUE</strong> if at least one value of the specified attribute matches the supplied value according to some context-specific approximate matching criterion. This criterion must subsume strict equality.</td>
</tr>
</tbody>
</table>

**Wildcarded Strings**

A wildcarded string consists of a sequence of alternating wildcard specifiers and strings. The sequence can start with either a wildcard specifier or a string, and end with either a wildcard specifier or a string.

The wildcard specifier is denoted by the asterisk character (`*`) and means zero or more occurrences of any character.

Wildcarded strings can be used to specify substring matches. The following are examples of wildcarded strings and what they mean:
Extended Operations

In addition to the relational operators, extended operators can be specified. All extended operators return either TRUE or FALSE. A filter expression can contain both relational and extended operations.

Extended operators are specified using an identifier (see FN_identifier_t(3N)) or a string. If the operator is specified using a string, the string is used to construct an identifier of format FN_ID_STRING. Identifiers of extended operators and signatures of the corresponding extended operations, as well as their suggested semantics, are registered with X/Open Company Ltd.

The following three extended operations are currently defined:

'name'(<Wildcarded String>)
The identifier for this operation is 'name' (FN_ID_STRING). The argument to this operation is a wildcard string. The operation returns TRUE if the name of the object matches the supplied wildcard string.

'reftype'(%i)
The identifier for this operation is 'reftype' (FN_ID_STRING). The argument to this operation is an identifier. The operation returns TRUE if the reference type of the object is equal to the supplied identifier.

'addrtype'(%i)
The identifier for this operation is 'addrtype' (LM FN_ID_STRING). The argument to the operation is an identifier. The operation returns TRUE if any of the address types in the reference of the object is equal to the supplied identifier.

Support and exact semantics of extended operations are context-specific. If a context does not support an extended operation, or if the filter expression supplies the extended operation with either an incorrect number or type of arguments, the error

---

<table>
<thead>
<tr>
<th>Wildcarded String</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>Any string</td>
</tr>
<tr>
<td>'tom'</td>
<td>The string tom</td>
</tr>
<tr>
<td>'harv'*</td>
<td>Any string starting with harv</td>
</tr>
<tr>
<td>*'ing'</td>
<td>Any string ending with ing</td>
</tr>
<tr>
<td>'a'*'b'</td>
<td>Any string starting with a and ending with b</td>
</tr>
<tr>
<td>'a*b'</td>
<td>The string a*b</td>
</tr>
<tr>
<td>'jo'='ph'*'ne'='er'</td>
<td>Any string starting with jo, and containing the substring ph, and which contains the substring ne in the portion of the string following ph, and which ends with er</td>
</tr>
<tr>
<td>%s*</td>
<td>Any string starting with the supplied string</td>
</tr>
<tr>
<td>'bix'='%s'</td>
<td>Any string starting with bix and ending with the supplied string</td>
</tr>
</tbody>
</table>

String matches involving strings of different locales (code set, language, or territory) are resolved in an implementation-dependent way.
FN_E_SEARCH_INVALID_OP is returned. (Note: FN_E_OPERATION_NOT_SUPPORTED is returned when fn_attr_ext_search(3N) is not supported.)

The following are examples of filter expressions that contain extended operations:

<table>
<thead>
<tr>
<th>Expression</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>'name'('bill'* )</td>
<td>Evaluates to TRUE if the name of the object starts with bill.</td>
</tr>
<tr>
<td>%i(%a, %v)</td>
<td>Evaluates to result of applying the specified operation to the supplied arguments.</td>
</tr>
<tr>
<td>(%a == %v) and 'name'('joe'* )</td>
<td>Evaluates to TRUE if the specified attribute has the given value and if the name of the object starts with joe.</td>
</tr>
</tbody>
</table>

RETURN VALUES

fn_search_filter_create() returns a pointer to an FN_search_filter_t object if the operation succeeds; otherwise it returns a NULL pointer.

ERRORS

fn_search_filter_create() returns a NULL pointer if the operation fails and sets status in the following way:

FN_E_SEARCH_INVALID_FILTER
The filter expression had a syntax error or some other problem.

FN_E_SEARCH_INVALID_OP
An operator in the filter expression is not supported or, if the operator is an extended operator, the number of types of arguments supplied does not match the signature of the operation.

FN_E_INVALID_ATTR_IDENTIFIER
The left hand side of an operator expression was not an attribute.

FN_E_INVALID_ATTR_VALUE
The right hand side of an operator expression was not an integer, attribute value, or (wildcarded) string.

Other status codes are possible as described in the reference manual pages for FN_status_t(3N) and xfn_status_codes(3N).

EXAMPLES

The following examples illustrate how to create three different filters. The first example shows how to construct a filter involving substitution tokens and literals in the same filter expression. This example creates a filter for named objects whose color attribute contains a string value of red, blue, or white. The first two values are specified using substitution tokens; the last value, white, is specified as a literal in the expression.

```c
unsigned int status;
extern FN_attribute_t *attr_color;
FN_string_t *red = fn_string_from_str((unsigned char *)"red");
FN_string_t *blue = fn_string_from_str((unsigned char *)"blue");
FN_search_filter_t *sfilter;
```
sfilter = fn_search_filter_create(
    &status,
    "(%a == %s) or (%a == %s) or (%a == 'white')",
    attr_color, red, attr_color, blue,
    attr_color);

The second example illustrates how to construct a filter involving a wildcarded string. This example creates a filter for searching for named objects whose last_name attribute has a value that begins with the character m.

unsigned int status;
extern FN_attribute_t *attr_last_name;
FN_search_filter_t *sfilter;
sfilter = fn_search_filter_create(
    &status, "%a == 'm'\n", attr_last_name);

The third example illustrates how to construct a filter involving extended operations. This example creates a filter for finding all named objects whose name ends with ton.

unsigned int status;
FN_search_filter_t *sfilter;
sfilter = fn_search_filter_create(&status, "'name'('ton')");

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO FN_attribute_t(3N), FN_attrvalue_t(3N), FN_identifier_t(3N), FN_status_t(3N), FN_string_t(3N), fn_attr_ext_search(3N), xfn_status_codes(3N), attributes(5)
NAME
FN_status_t, fn_status_create, fn_status_destroy, fn_status_copy, fn_status_assign,
fn_status_code, fn_status_remaining_name, fn_status_resolved_name,
fn_status_resolved_ref, fn_status_diagnostic_message, fn_status_link_code,
fn_status_link_remaining_name, fn_status_link_resolved_name,
fn_status_link_resolved_ref, fn_status_link_diagnostic_message, fn_status_is_success,
fn_status_set_success, fn_status_set, fn_status_set_code, fn_status_set_remaining_name,
fn_status_set_resolved_name, fn_status_set_resolved_ref,
fn_status_set_diagnostic_message, fn_status_set_link_code,
fn_status_set_link_remaining_name, fn_status_set_link_resolved_name,
fn_status_set_link_resolved_ref, fn_status_set_link_diagnostic_message,
fn_status_append_resolved_name, fn_status_append_remaining_name,
fn_status_advance_by_name, fn_status_description – an XFN status object

SYNOPSIS
cc [ flag ... ] file ... -lxfn [ library ... ]
#include <xfn/xfn.h>
FN_status_t *fn_status_create(void);
void fn_status_destroy(FN_status_t *stat);
FN_status_t *fn_status_copy(const FN_status_t *stat);
FN_status_t *fn_status_assign(FN_status_t *dst, const FN_status_t *src);
unsigned int fn_status_code(const FN_status_t *stat);
const FN_composite_name_t *fn_status_remaining_name(const FN_status_t *stat);
const FN_composite_name_t *fn_status_resolved_name(const FN_status_t *stat);
const FN_ref_t *fn_status_resolved_ref(const FN_status_t *stat);
const FN_string_t *fn_status_diagnostic_message(const FN_status_t *stat);
unsigned int fn_status_link_code(const FN_status_t *stat);
const FN_composite_name_t *fn_status_link_remaining_name(const FN_status_t *stat);
const FN_composite_name_t *fn_status_link_resolved_name(const FN_status_t *stat);
const FN_ref_t *fn_status_link_resolved_ref(const FN_status_t *stat);
const FN_string_t *fn_status_link_diagnostic_message(const FN_status_t *stat);
int fn_status_is_success(const FN_status_t *stat);
int fn_status_set_success(FN_status_t *stat);
int fn_status_set(FN_status_t *stat, unsigned int code, const FN_ref_t *resolved_ref,
    const FN_composite_name_t *resolved_name,
    const FN_composite_name_t *remaining_name);
int fn_status_set_code(FN_status_t *stat, unsigned int code);
int fn_status_set_remaining_name(FN_status_t *stat,
    const FN_composite_name_t *name);
int fn_status_set_resolved_name(FN_status_t *stat,
   const FN_composite_name_t *name);

int fn_status_set_resolved_ref(FN_status_t *stat, const FN_ref_t *ref);

int fn_status_set_diagnostic_message(FN_status_t *stat,
   const FN_string_t *msg);

int fn_status_set_link_code(FN_status_t *stat, unsigned int code);

int fn_status_set_link_remaining_name(FN_status_t *stat,
   const FN_composite_name_t *name);

int fn_status_set_link_resolved_name(FN_status_t *stat,
   const FN_composite_name_t *name);

int fn_status_set_link_resolved_ref(FN_status_t *stat, const FN_ref_t *ref);

int fn_status_set_link_diagnostic_message(FN_status_t *stat,
   const FN_string_t *msg);

int fn_status_append_resolved_name(FN_status_t *stat,
   const FN_composite_name_t *name);

int fn_status_append_remaining_name(FN_status_t *stat,
   const FN_composite_name_t *name);

int fn_status_advance_by_name(FN_status_t *stat,
   const FN_composite_name_t *prefix, const FN_ref_t *resolved_ref);

FN_string_t *fn_status_description(const FN_status_t *stat,
   unsigned int detail, unsigned int *more_detail);

DESCRIPTION The result status of operations in the context interface and the attribute interface is encapsulated in an FN_status_t object. This object contains information about how the operation completed: whether an error occurred in performing the operation, the nature of the error, and information that helps locate where the error occurred. In the case that the error occurred while resolving an XFN link, the status object contains additional information about that error.

The context status object consists of several items of information:

primary status code An unsigned int code describing the disposition of the operation.

resolved name In the case of a failure during the resolution phase of the operation, this is the leading portion of the name that was resolved successfully. Resolution may have been successful beyond this point, but the error might not be pinpointed further.

resolved reference The reference to which resolution was successful (in other words, the reference to which the resolved name is bound).

remaining name The remaining unresolved portion of the name.

diagnostic message This contains any diagnostic message returned by the context implementation. This message provides the context implementation a way of notifying the end-user or administrator of any
implementation-specific information related to the returned error status. The diagnostic message could then be used by the end-user or administrator to take appropriate out-of-band action to rectify the problem.

**link status code**

In the case that an error occurred while resolving an XFN link, the primary status code has the value `FN_E_LINK_ERROR` and the link status code describes the error that occurred while resolving the XFN link.

**resolved link name**

In the case of a link error, this contains the resolved portion of the name in the XFN link.

**resolved link reference**

In the case of a link error, this contains the reference to which the resolved link name is bound.

**remaining link name**

In the case of a link error, this contains the remaining unresolved portion of the name in the XFN link.

**link diagnostic message**

In the case of a link error, this contains any diagnostic message related to the resolution of the link.

Both the primary status code and the link status code are values of type `unsigned int` that are drawn from the same set of meaningful values. XFN reserves the values 0 through 127 for standard meanings. The values and interpretations for the codes are determined by XFN. See `xfn_status_codes(3N)`.

*fn_status_create()* creates a status object with status `FN_SUCCESS`. *fn_status_destroy()* releases the storage associated with `stat`. *fn_status_copy()* returns a copy of the status object `stat`. *fn_status_assign()* makes a copy of the status object `src` and assigns it to `dst`, releasing any old contents of `dst`. A pointer to the same object as `dst` is returned.

*fn_status_code()* returns the status code. *fn_status_remaining_name()* returns the remaining part of name to be resolved. *fn_status_resolved_name()* returns the part of the composite name that has been resolved. *fn_status_resolved_ref()* returns the reference to which resolution was successful. *fn_status_diagnostic_message()* returns any diagnostic message set by the context implementation.

*fn_status_link_code()* returns the link status code. *fn_status_link_remaining_name()* returns the remaining part of the link name that has not been resolved.

*fn_status_link_resolved_name()* returns the part of the link name that has been resolved. *fn_status_link_resolved_ref()* returns the reference to which resolution of the link was successful. *fn_status_link_diagnostic_message()* returns any diagnostic message set by the context implementation during resolution of the link.

*fn_status_is_success()* returns 1 if the status indicates success, 0 otherwise.

*fn_status_set_success()* sets the status code to `FN_SUCCESS` and clears all other parts of `stat`. *fn_status_set()* sets the non-link contents of the status object `stat`.

*fn_status_set_code()* sets the primary status code field of the status object `stat`.

*fn_status_set_remaining_name()* sets the remaining name part of the status object `stat` to
name. \texttt{fn\_status\_set\_resolved\_name()} sets the resolved name part of the status object \texttt{stat} to \texttt{name}. \texttt{fn\_status\_set\_resolved\_ref()} sets the resolved reference part of the status object \texttt{stat} to \texttt{ref}. \texttt{fn\_status\_set\_diagnostic\_message()} sets the diagnostic message part of the status object to \texttt{msg}.

\texttt{fn\_status\_set\_link\_code()} sets the link status code field of the status object \texttt{stat} to indicate why resolution of the link failed. \texttt{fn\_status\_set\_link\_remaining\_name()} sets the remaining link name part of the status object \texttt{stat} to \texttt{name}.

\texttt{fn\_status\_set\_link\_resolved\_name()} sets the resolved link name part of the status object \texttt{stat} to \texttt{name}. \texttt{fn\_status\_set\_link\_resolved\_ref()} sets the resolved link reference part of the status object \texttt{stat} to \texttt{ref}. \texttt{fn\_status\_set\_link\_diagnostic\_message()} sets the link diagnostic message part of the status object to \texttt{msg}.

\texttt{fn\_status\_append\_resolved\_name()} appends as additional components \texttt{name} to the resolved name part of the status object \texttt{stat}. \texttt{fn\_status\_append\_remaining\_name()} appends as additional components \texttt{name} to the remaining name part of the status object \texttt{stat}. \texttt{fn\_status\_advance\_by\_name()} removes \texttt{prefix} from the remaining name, and appends it to the resolved name. The resolved reference part is set to \texttt{resolved\_ref}. This operation returns \texttt{1} on success, \texttt{0} if the \texttt{prefix} is not a prefix of the remaining name.

RETURN VALUES The \texttt{fn\_status\_set\_\ast()} operations return \texttt{1} if the operation succeeds, \texttt{0} if the operation fails.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO \texttt{FN\_composite\_name\_t(3N)}, \texttt{FN\_ref\_t(3N)}, \texttt{FN\_string\_t(3N)}, \texttt{xfn(3N)}, \texttt{xfn\_status\_codes(3N)}, attributes(5)

NOTES The implementation of XFN in this Solaris release is based on the X/Open preliminary specification. It is likely that there will be minor changes to these interfaces to reflect changes in the final version of this specification. The next minor release of Solaris will offer binary compatibility for applications developed using the current interfaces. As the interfaces evolve toward standardization, it is possible that future releases of Solaris will require minor source code changes to applications that have been developed against the preliminary specification.
NAME
FN_string_t, fn_string_create, fn_string_destroy, fn_string_from_str,
fn_string_from_str_n, fn_string_str, fn_string_from_contents, fn_string_code_set,
fn_string_charcount, fn_string_bytecount, fn_string_contents, fn_string_copy,
fn_string_assign, fn_string_from_strings, fn_string_from_substring, fn_string_is_empty,
fn_string_compare, fn_string_compare_substring, fn_string_next_substring,
fn_string_prev_substring – a character string

SYNOPSIS
c c [ flag ... ] file ... -lxfn [ library ... ]
#include <xfn/xfn.h>
FN_string_t *fn_string_create(void);
void fn_string_destroy(FN_string_t *str);
FN_string_t *fn_string_from_str(const unsigned char *cstr);
FN_string_t *fn_string_from_str_n(const unsigned char *cstr, size_t n);
const unsigned char *fn_string_str(const FN_string_t *str, unsigned int *status);
FN_string_t *fn_string_from_contents(unsigned long code_set, const void *locale_info,
size_t locale_info_len, size_t charcount, size_t bytecount, const void *contents,
unsigned int *status);
unsigned long fn_string_code_set(const FN_string_t *str, const void **locale_info,
size_t *locale_info_len);
size_t fn_string_charcount(const FN_string_t *str);
size_t fn_string_bytecount(const FN_string_t *str);
const void *fn_string_contents(const FN_string_t *str);
FN_string_t *fn_string_copy(const FN_string_t *str);
FN_string_t *fn_string_assign(FN_string_t *dst, const FN_string_t *src);
FN_string_t *fn_string_from_strings(unsigned int *status, const FN_string_t *s1,
const FN_string_t *s2, ...);
FN_string_t *fn_string_from_substring(const FN_string_t *str, int first, int last);
int fn_string_is_empty(const FN_string_t *str);
int fn_string_compare(const FN_string_t *str1, const FN_string_t *str2,
unsigned int string_case, unsigned int *status);
int fn_string_compare_substring(const FN_string_t *str1, int first, int last,
const FN_string_t *str2, unsigned int string_case, unsigned int *status);
int fn_string_next_substring(const FN_string_t *str, const FN_string_t *sub, int index,
unsigned int string_case, unsigned int *status);
int fn_string_prev_substring(const FN_string_t *str, const FN_string_t *sub, int index,
unsigned int string_case, unsigned int *status);

modified 13 Dec 1996 SunOS 5.6 3N-625
DESCRIPTION

The FN_string_t type is used to represent character strings in the XFN interface. It provides insulation from specific string representations.

The FN_string_t supports multiple code sets. It provides creation functions for character strings of the code set of the current locale setting and a generic creation function for arbitrary code sets. The degree of support for the functions that manipulate FN_string_t for arbitrary code sets is implementation-dependent. An XFN implementation is required to support the ISO 646 code set; all other code sets are optional.

fn_string_destroy() releases the storage associated with the given string.

fn_string_create() creates an empty string.

fn_string_from_str() creates an FN_string_t object from the given null terminated string based on the code set of the current locale setting. The number of characters in the string is determined by the code set of the current locale setting. fn_string_from_str_n() is like fn_string_from_str() except only n characters from the given string are used.

fn_string_str() returns the contents of the given string str in the form of a null terminated string in the code set and current locale setting.

fn_string_from_contents() creates an FN_string_t object using the specified code set code_set, locale information locale_info, and data in the given buffer contents. bytecount specifies the number of bytes in contents and charcount specifies the number of characters represented by contents.

fn_string_code_set() returns the code set associated with the given string object and, if present, the locale information in locale_info. fn_string_charcount() returns the number of characters in the given string object. fn_string_bytecount() returns the number of bytes used to represent the given string object. fn_string_contents() returns a pointer to the contents of the given string object.

fn_string_copy() returns a copy of the given string object. fn_string_assign() makes a copy of the string object src and assigns it to dst, releasing any old contents of dst. A pointer to the same object as dst is returned. fn_string_from_strings() is a function that takes a variable number of arguments (minimum of 2), the last of which must be NULL (0); it returns a new string object composed of the left to right concatenation of the given strings, in the given order. The support for strings with different code sets and/or locales as arguments to a single invocation of fn_string_from_strings() is implementation-dependent. fn_string_from_substring() returns a new string object consisting of the characters located between first and last inclusive from str. Indexing begins with 0. If last is FN_STRING_INDEX_LAST or exceeds the length of the string, the index of the last character of the string is used.

fn_string_is_empty() returns whether str is an empty string.

Comparison of two strings must take into account code set and locale information. If strings are in the same code set and same locale, case sensitivity is applied according to the case sensitivity rules applicable for the code set and locale; case sensitivity may not necessarily be relevant for all string encodings. If string_case is non-zero, case is significant and equality for strings of the same code set is defined as equality between byte-wise encoded values of the strings. If string_case is zero, case is ignored and equality for strings of the same code set is defined using the definition of case-insensitive equality.
Network Functions

for the specific code set. Support for comparison between strings of different code sets, or lack thereof, is implementation-dependent.

`fn_string_compare()` compares strings `str1` and `str2` and returns 0 if they are equal, non-zero if they are not equal. If two strings are not equal, `fn_string_compare()` returns a positive value if the difference of `str2` precedes that of `str1` in terms of byte-wise encoded value (with case-sensitivity taken into account when `string_case` is non-zero), and a negative value if the difference of `str1` precedes that of `str2`, in terms of byte-wise encoded value (with case-sensitivity taken into account when `string_case` is non-zero). Such information (positive versus negative return value) may be used by applications that use strings of code sets in which ordering is meaningful; this information is not of general use in internationalized environments. `fn_string_compare_substring()` is similar to `fn_string_compare()` except that `fn_string_compare_substring()` compares characters between first and last inclusive of `str2` with `str1`. Comparison of strings with incompatible code sets returns a negative or positive value (never 0) depending on the implementation.

`fn_string_next_substring()` returns the index of the next occurrence of `sub` at or after `index` in the string `str`. `FN_STRING_INDEX_NONE` is returned if `sub` does not occur.

`fn_string_prev_substring()` returns the index of the previous occurrence of `sub` at or before `index` in the string `str`. `FN_STRING_INDEX_NONE` is returned if `sub` does not occur.

In both of these functions, `string_case` specifies whether the search should take case-sensitivity into account.

**ERRORS**

`fn_string_str()` returns 0 and sets `status` to `FN_E_INCOMPATIBLE_CODE_SETS` if the given string’s representation cannot be converted into the code set of the current locale setting. It is implementation-dependent which code sets can be converted into the code set of the current locale.

Code set mismatches that occur during concatenation, searches, or comparisons are resolved in an implementation-dependent way. When an implementation discovers that arguments to substring searches and comparison operations have incompatible code sets, it sets `status` to `FN_E_INCOMPATIBLE_CODE_SETS`. In such cases, `fn_string_from_strings()` returns 0. The returned value for comparison operations when there is code set or locale incompatibility is either negative or positive (greater than 0); it is never 0.

`fn_string_from_contents()` returns 0 and `status` is set to `FN_E_INCOMPATIBLE_CODE_SETS` if the supplied code set and/or locale information are not supported by the XFN implementation.

**ATTRIBUTES**

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**SEE ALSO**

`xfn(3N), attributes(5)`
The implementation of XFN in this Solaris release is based on the X/Open preliminary specification. It is likely that there will be minor changes to these interfaces to reflect changes in the final version of this specification. The next minor release of Solaris will offer binary compatibility for applications developed using the current interfaces. As the interfaces evolve toward standardization, it is possible that future releases of Solaris will require minor source code changes to applications that have been developed against the preliminary specification.
NAME  fopen, freopen – open a stream

SYNOPSIS  
/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;

DESCRIPTION  fopen() opens the file named by file and associates a stream with it. If the open succeeds, fopen() returns a pointer to be used to identify the stream in subsequent operations.

file points to a character string that contains the name of the file to be opened.

mode is a character string having one of the following values:
  r open for reading
  w truncate or create for writing
  a append: open for writing at end of file, or create for writing
  r+ open for update (reading and writing)
  w+ truncate or create for update
  a+ append; open or create for update at EOF

freopen() opens the file named by file and associates the stream pointed to by iop with it. The mode argument is used just as in fopen(). The original stream is closed, regardless of whether the open ultimately succeeds. If the open succeeds, freopen() returns the original value of iop.

freopen() is typically used to attach the preopened streams associated with stdin, stdout, and stderr to other files.

When a file is opened for update, both input and output may be done on the resulting stream. However, output may not be directly followed by input without an intervening fseek(3S) or rewind(3S), and input may not be directly followed by output without an intervening fseek(3S) or rewind(3S). An input operation which encounters EOF will fail.

RETURN VALUES  fopen() and freopen() return a NULL pointer on failure.

SEE ALSO  open(2), fclose(3S), fopen(3S), freopen(3S), fseek(3S), malloc(3C), rewind(3S)

NOTES  Use of these interfaces should be restricted to only applications written on BSD platforms. Use of these interfaces with any of the system libraries or in multi-thread applications is unsupported.
In order to support the same number of open files that the system does, `fopen()` must allocate additional memory for data structures using `malloc(3C)` after 64 files have been opened. This confuses some programs which use their own memory allocators.

The interfaces of `fopen()` and `freopen()` differ from the Standard I/O Functions `fopen(3S)` and `freopen(3S)`. The Standard I/O Functions distinguish binary from text files with an additional use of ‘b’ as part of the mode. This enables portability of `fopen(3S)` and `freopen(3S)` beyond SunOS 4.X systems.
NAME  fopen – open a stream

SYNOPSIS  
#include <stdio.h>

FILE *fopen(const char *filename, const char *mode);

DESCRIPTION  The fopen() function opens the file whose pathname is the string pointed to by filename, and associates a stream with it.

The argument mode points to a string beginning with one of the following sequences:
- r or rb: open file for reading
- w or wb: truncate to zero length or create file for writing
- a or ab: append; open or create file for writing at end-of-file
- r+ or rb+ or r+b: open file for update (reading and writing)
- w+ or wb+ or w+b: truncate to zero length or create file for update
- a+ or ab+ or a+b: append; open or create file for update, writing at end-of-file

The character b has no effect, but is allowed for ISO C standard conformance. Opening a file with read mode (r as the first character in the mode argument) fails if the file does not exist or cannot be read.

Opening a file with append mode (a as the first character in the mode argument) causes all subsequent writes to the file to be forced to the then current end-of-file, regardless of intervening calls to fseek(3S). If two separate processes open the same file for append, each process may write freely to the file without fear of destroying output being written by the other. The output from the two processes will be intermixed in the file in the order in which it is written.

When a file is opened with update mode (+ as the second or third character in the mode argument), both input and output may be performed on the associated stream. However, output must not be directly followed by input without an intervening call to fflush(3S) or to a file positioning function (fseek(3S), fsetpos(3S) or rewind(3S)), and input must not be directly followed by output without an intervening call to a file positioning function, unless the input operation encounters end-of-file.

When opened, a stream is fully buffered if and only if it can be determined not to refer to an interactive device. The error and end-of-file indicators for the stream are cleared.

If mode is w, a, w+ or a+ and the file did not previously exist, upon successful completion, fopen() function will mark for update the st_atime, st_ctime and st_mtime fields of the file and the st_ctime and st_mtime fields of the parent directory.

If mode is w or w+ and the file did previously exist, upon successful completion, fopen() will mark for update the st_ctime and st_mtime fields of the file. The fopen() function will allocate a file descriptor as open(2) does.

The largest value that can be represented correctly in an object of type off_t will be established as the offset maximum in the open file description.
RETURN VALUES
Upon successful completion, `fopen()` returns a pointer to the object controlling the stream. Otherwise, a null pointer is returned, and `errno` is set to indicate the error.

`fopen()` may fail and not set `errno` if there are no free `stdio` streams.

<table>
<thead>
<tr>
<th>ERRORS</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>EACCES</strong></td>
<td>Search permission is denied on a component of the path prefix, or the file exists and the permissions specified by <code>mode</code> are denied, or the file does not exist and write permission is denied for the parent directory of the file to be created.</td>
</tr>
<tr>
<td><strong>EINTR</strong></td>
<td>A signal was caught during <code>fopen()</code>.</td>
</tr>
<tr>
<td><strong>EISDIR</strong></td>
<td>The named file is a directory and <code>mode</code> requires write access.</td>
</tr>
<tr>
<td><strong>ELOOP</strong></td>
<td>Too many symbolic links were encountered in resolving <code>path</code>.</td>
</tr>
<tr>
<td><strong>EMFILE</strong></td>
<td><code>OPEN_MAX</code> file descriptors are currently open in the calling process.</td>
</tr>
<tr>
<td><strong>ENAMETOOLONG</strong></td>
<td>The length of the <code>filename</code> exceeds <code>PATH_MAX</code> or a path name component is longer than <code>NAME_MAX</code>.</td>
</tr>
<tr>
<td><strong>ENFILE</strong></td>
<td>The maximum allowable number of files is currently open in the system.</td>
</tr>
<tr>
<td><strong>ENOENT</strong></td>
<td>A component of <code>filename</code> does not name an existing file or <code>filename</code> is an empty string.</td>
</tr>
<tr>
<td><strong>ENOSPC</strong></td>
<td>The directory or file system that would contain the new file cannot be expanded, the file does not exist, and it was to be created.</td>
</tr>
<tr>
<td><strong>ENOTDIR</strong></td>
<td>A component of the path prefix is not a directory.</td>
</tr>
<tr>
<td><strong>ENXIO</strong></td>
<td>The named file is a character special or block special file, and the device associated with this special file does not exist.</td>
</tr>
<tr>
<td><strong>EOVERFLOW</strong></td>
<td>The current value of the file position cannot be represented correctly in an object of type <code>fpos_t</code>.</td>
</tr>
<tr>
<td><strong>EROFS</strong></td>
<td>The named file resides on a read-only file system and <code>mode</code> requires write access.</td>
</tr>
</tbody>
</table>

The `fopen()` function may fail if:

<table>
<thead>
<tr>
<th>NEW ERRORS</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>EINVAL</strong></td>
<td>The value of the <code>mode</code> argument is not valid.</td>
</tr>
<tr>
<td><strong>EMFILE</strong></td>
<td><code>FOPEN_MAX</code> streams are currently open in the calling process.</td>
</tr>
<tr>
<td><strong>EMFILE</strong></td>
<td><code>STREAM_MAX</code> streams are currently open in the calling process.</td>
</tr>
<tr>
<td><strong>ENAMETOOLONG</strong></td>
<td>Pathname resolution of a symbolic link produced an intermediate result whose length exceeds <code>PATH_MAX</code>.</td>
</tr>
<tr>
<td><strong>ENOMEM</strong></td>
<td>Insufficient storage space is available.</td>
</tr>
<tr>
<td><strong>ETXTBSY</strong></td>
<td>The file is a pure procedure (shared text) file that is being executed and <code>mode</code> requires write access.</td>
</tr>
</tbody>
</table>
**USAGE**

STREAM_MAX is the number of streams that one process can have open at one time. If defined, it has the same value as FOPEN_MAX.

The fopen() function has an explicit 64-bit equivalent. See interface64(5).

**ATTRIBUTES**

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**SEE ALSO**

fclose(3S), fdopen(3S), fflush(3S), freopen(3S), fsetpos(3S), rewind(3S), attributes(5), interface64(5)
NAME  
forms – character based forms package

SYNOPSIS  
#include <form.h>

DESCRIPTION  
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.

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.

Current Default Values for Field Attributes  
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.)

Routine Name Index  
The following table lists each forms routine and the name of the manual page on which it is described.

<table>
<thead>
<tr>
<th>forms Routine Name</th>
<th>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>
<tr>
<td>field_opts_on</td>
<td>form_field_opts(3X)</td>
</tr>
<tr>
<td>field_pad</td>
<td>form_field_attributes(3X)</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>------------------------------------</td>
</tr>
<tr>
<td>field_status</td>
<td>form_field_buffer(3X)</td>
</tr>
<tr>
<td>field_term</td>
<td>form_hook(3X)</td>
</tr>
<tr>
<td>field_type</td>
<td>form_field_validation(3X)</td>
</tr>
<tr>
<td>field_userptr</td>
<td>form_field_userptr(3X)</td>
</tr>
<tr>
<td>form_driver</td>
<td>form_driver(3X)</td>
</tr>
<tr>
<td>form_fields</td>
<td>form_field(3X)</td>
</tr>
<tr>
<td>form_init</td>
<td>form_hook(3X)</td>
</tr>
<tr>
<td>form_opts</td>
<td>form_opts(3X)</td>
</tr>
<tr>
<td>form_opts_off</td>
<td>form_opts(3X)</td>
</tr>
<tr>
<td>form_opts_on</td>
<td>form_opts(3X)</td>
</tr>
<tr>
<td>form_page</td>
<td>form_page(3X)</td>
</tr>
<tr>
<td>form_sub</td>
<td>form_win(3X)</td>
</tr>
<tr>
<td>form_term</td>
<td>form_hook(3X)</td>
</tr>
<tr>
<td>form_userptr</td>
<td>form_userptr(3X)</td>
</tr>
<tr>
<td>form_win</td>
<td>form_win(3X)</td>
</tr>
<tr>
<td>free_field</td>
<td>form_field_new(3X)</td>
</tr>
<tr>
<td>free_fieldtype</td>
<td>form_fieldtype(3X)</td>
</tr>
<tr>
<td>free_form</td>
<td>form_new(3X)</td>
</tr>
<tr>
<td>link_field</td>
<td>form_field_new(3X)</td>
</tr>
<tr>
<td>link_fieldtype</td>
<td>form_fieldtype(3X)</td>
</tr>
<tr>
<td>move_field</td>
<td>form_field(3X)</td>
</tr>
<tr>
<td>new_field</td>
<td>form_field_new(3X)</td>
</tr>
<tr>
<td>new_fieldtype</td>
<td>form_fieldtype(3X)</td>
</tr>
<tr>
<td>new_form</td>
<td>form_new(3X)</td>
</tr>
<tr>
<td>new_page</td>
<td>form_new_page(3X)</td>
</tr>
<tr>
<td>pos_form_cursor</td>
<td>form_cursor(3X)</td>
</tr>
<tr>
<td>post_form</td>
<td>form_post(3X)</td>
</tr>
<tr>
<td>scale_form</td>
<td>form_win(3X)</td>
</tr>
<tr>
<td>set_current_field</td>
<td>form_page(3X)</td>
</tr>
<tr>
<td>set_field_back</td>
<td>form_field_attributes(3X)</td>
</tr>
<tr>
<td>set_field_buffer</td>
<td>form_field_buffer(3X)</td>
</tr>
<tr>
<td>set_field_fore</td>
<td>form_field_attributes(3X)</td>
</tr>
<tr>
<td>set_field_init</td>
<td>form_hook(3X)</td>
</tr>
<tr>
<td>set_field_just</td>
<td>form_field_just(3X)</td>
</tr>
<tr>
<td>set_field_opts</td>
<td>form_field_opts(3X)</td>
</tr>
<tr>
<td>set_field_pad</td>
<td>form_field_attributes(3X)</td>
</tr>
<tr>
<td>set_field_status</td>
<td>form_field_buffer(3X)</td>
</tr>
<tr>
<td>set_field_term</td>
<td>form_hook(3X)</td>
</tr>
<tr>
<td>set_field_type</td>
<td>form_field_validation(3X)</td>
</tr>
<tr>
<td>set_field_userptr</td>
<td>form_field_userptr(3X)</td>
</tr>
<tr>
<td>set_fieldtype_arg</td>
<td>form_fieldtype(3X)</td>
</tr>
<tr>
<td>set_fieldtype_choice</td>
<td>form_fieldtype(3X)</td>
</tr>
<tr>
<td>set_form_fields</td>
<td>form_field(3X)</td>
</tr>
<tr>
<td>set_form_init</td>
<td>form_hook(3X)</td>
</tr>
<tr>
<td>set_form_opts</td>
<td>form_opts(3X)</td>
</tr>
</tbody>
</table>
forms (3X)  Miscellaneous Library Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>set_form_page</td>
<td>form_page(3X)</td>
</tr>
<tr>
<td>set_form_sub</td>
<td>form_win(3X)</td>
</tr>
<tr>
<td>set_form_term</td>
<td>form_hook(3X)</td>
</tr>
<tr>
<td>set_form_userptr</td>
<td>form_userptr(3X)</td>
</tr>
<tr>
<td>set_form_win</td>
<td>form_win(3X)</td>
</tr>
<tr>
<td>set_max_field</td>
<td>form_field_buffer(3X)</td>
</tr>
<tr>
<td>set_new_page</td>
<td>form_new_page(3X)</td>
</tr>
<tr>
<td>unpost_form</td>
<td>form_post(3X)</td>
</tr>
</tbody>
</table>

**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.

**ATTRIBUTES**

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

**SEE ALSO**

curses(3X), attributes(5) and 3X pages whose names begin "form_" for detailed routine descriptions.

**NOTES**

The header <form.h> automatically includes the headers <eti.h> and <curses.h>.
NAME  
form_cursor, pos_form_cursor – position forms window cursor

SYNOPSIS  
cc [ flag ... ] file ... -lform -lcurses [ library ... ]
#include <form.h>
int pos_form_cursor(FORM *form);

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.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO  
curses(3X), forms(3X), attributes(5)

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  cc [flag ...] file ... -lform -lcurses [ library ... ]
#include <form.h>
int data_ahead(FORM *form);
int data_behind(FORM *form);

DESCRIPTION  data_ahead() returns TRUE (1) if the current field has more off-screen data ahead; otherwise it returns FALSE (0).
data_behind() returns TRUE (1) if the current field has more off-screen data behind; otherwise it returns FALSE (0).

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO  curses(3X), forms(3X), attributes(5)

NOTES  The header <form.h> automatically includes the headers <eti.h> and <curses.h>.
NAME form_driver – command processor for the forms subsystem

SYNOPSIS cc [ flag ... ] file ... -lform -lcurses [ library ... ]
#include <form.h>
int form_driver(FORM *form, int c);

DESCRIPTION form_driver() is the workhorse of the forms subsystem; it checks to
determine whether the character c 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 max-
imum value of a request listed below.
Form driver requests:
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.
ATTRIBUTES  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO  curses(3X), forms(3X), attributes(5)

NOTES  The header <form.h> automatically includes the headers <eti.h> and <curses.h>.
NAME
form_field, set_form_fields, form_fields, field_count, move_field – connect fields to forms

SYNOPSIS
cc [ flag ...] file ... -lform -lcurses [ library ...]
#include <form.h>
int set_form_fields(FORM *form, FIELD **field);
FIELD **form_fields(FORM *form);
int field_count(FORM *form);
int move_field(FIELD *field, int frow, int fcol);

DESCRIPTION
set_form_fields() changes the fields connected to form to fields. The original fields are disconnected.
form_fields() returns a pointer to the field pointer array connected to form.
field_count() returns the number of fields connected to form.
move_field() moves the disconnected field to the location frow, fcol in the forms subwindow.

RETURN VALUES
form_fields() returns NULL on error.
field_count() returns -1 on error.
set_form_fields() and move_field() 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_POSTED – The form is posted.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
curses(3X), forms(3X), attributes(5)

NOTES
The header <form.h> automatically includes the headers <eti.h> and <curses.h>.
NAME
form_field_attributes, set_field_fore, field_fore, set_field_back, field_back, set_field_pad,
field_pad – format the general display attributes of forms

SYNOPSIS
cc [ flag ... ] file ... -lform -lcurses [ library ... ]
#include <form.h>
int set_field_fore(FIELD *field, chtype attr);
cttype field_fore(FIELD *field);
int set_field_back(FIELD *field, chtype attr);
cttype field_back(FIELD *field);
int set_field_pad(FIELD *field, int pad);
cttype field_pad(FIELD *field);

DESCRIPTION
set_field_fore() sets the foreground attribute of field. The foreground attribute is the
display attribute used to display the field contents. field_fore() returns
the foreground attribute of field.
set_field_back() sets the background attribute of field. The background attribute is the
display attribute used to display the extent of the field. field_back() returns
the background attribute of field.
set_field_pad() sets the pad character of field to pad. The pad character is the character
used to fill within the field. field_pad() returns the pad character of field.

RETURN VALUES
field_fore(), field_back(), and field_pad() return default values if field is NULL. If field is
not NULL and is not a valid FIELD pointer, the return value from these routines is
undefined.
set_field_fore(), set_field_back(), and set_field_pad() return one of the following:
E_OK – The function returned successfully.
E_SYSTEM_ERROR – System error.
E_BAD_ARGUMENT – An argument is incorrect.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
curses(3X), forms(3X), attributes(5)

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

modified 31 Dec 1996		SunOS 5.6		3X-643
NAME  
form_field_buffer, set_field_buffer, field_buffer, set_field_status, field_status,  
set_max_field – set and get forms field attributes

SYNOPSIS  
cc [flag ...] file ... -lform -lcurses [ library ...]  
#include <form.h>  
int set_field_buffer(FIELD *field, int buf, char *value);  
char *field_buffer(FIELD *field, int buf);  
int set_field_status(FIELD *field, int status);  
int field_status(FIELD *field);  
int set_max_field(FIELD *field, int max);

DESCRIPTION  
set_field_buffer() sets buffer buf of field to value. 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 field buffer buf.

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 field to status. field_status() returns the  
status of field.

set_max_field() sets a maximum growth on a dynamic field, or if max=0 turns off any  
maximum growth.

RETURN VALUES  
field_buffer() returns NULL on error.
field_status() returns TRUE or FALSE.
set_field_buffer(), set_field_status(), and set_max_field() return one of the following:

E_OK  –  The function returned successfully.
E_SYSTEM_ERROR  –  System error.
E_BAD_ARGUMENT  –  An argument is incorrect.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO  
curses(3X), forms(3X), attributes(5)

NOTES  
The header <form.h> automatically includes the headers <eti.h> and <curses.h>.
NAME  
form_field_info, field_info, dynamic_field_info – get forms field characteristics

SYNOPSIS  
cc [ flag ... ] file ... -lform -lcurses [ library ... ]  
#include <form.h>
int field_info(FIELD *field, int *rows, int *cols, int *frow, int *fcoll, int *nrow, int *nbuf);
int dynamic_field_info(FIELD *field, int *drows, int *dcolls, int *max);

DESCRIPTION  
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 rows, cols, frow, fcol, nrow, and nbuf.

dynamic_field_info() returns the actual size of the field in the pointer arguments drows, dcols and returns the maximum growth allowed for field in max. If no maximum growth limit is specified for field, max will contain 0. A field can be made dynamic by turning off the field option O_STATIC.

RETURN VALUES  
These routines return one of the following:
E_OK – The function returned successfully.
E_SYSTEM_ERROR – System error.
E_BAD_ARGUMENT – An argument is incorrect.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO  
curses(3X), forms(3X), attributes(5)

NOTES  
The header <form.h> automatically includes the headers <eti.h> and <curses.h>.
NAME  form_field_just, set_field_just, field_just – format the general appearance of forms

SYNOPSIS  cc [ flag ...] file ... -lform -lcurses [ library ...]  
#include <form.h>  
int set_field_just(FIELD *field, int justification);  
int field_just(FIELD *field);

DESCRIPTION  set_field_just() sets the justification for field. Justification may be one of:  
NO_JUSTIFICATION, JUSTIFY_RIGHT, JUSTIFY_LEFT, or JUSTIFY_CENTER.

The field justification will be ignored if field is a dynamic field.

field_just() returns the type of justification assigned to field.

RETURN VALUES  field_just() returns one of the following:

NO_JUSTIFICATION, JUSTIFY_RIGHT,  
JUSTIFY_LEFT, or JUSTIFY_CENTER.

set_field_just() returns one of the following:

E_OK – The function returned successfully.
E_SYSTEM_ERROR – System error.
E_BAD_ARGUMENT – An argument is incorrect.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO  curses(3X), forms(3X), attributes(5)

NOTES  The header <form.h> automatically includes the headers <eti.h> and <curses.h>.
NAME  form_field_new, new_field, dup_field, link_field, free_field, – create and destroy forms fields

SYNOPSIS  cc [ flag ... ] file ... -lform -lcurses [ library ... ]
#include <form.h>
FIELD *new_field(int r, int c, int frow, int fcol, int nrow, int ncol);
FIELD *dup_field(FIELD *field, int frow, int fcol);
FIELD *link_field(FIELD *field, int frow, int fcol);
int free_field(FIELD *field);

DESCRIPTION  new_field() creates a new field with r rows and c columns, starting at frow, fcol, in the subwindow of a form. nrow is the number of off-screen rows and nbuf is the number of additional working buffers. This routine returns a pointer to the new field.

dup_field() duplicates field at the specified location. All field attributes are duplicated, including the current contents of the field buffers.

link_field() also duplicates field 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.

free_field() frees the storage allocated for field.

RETURN VALUES  Routines that return pointers return NULL on error. free_field() returns 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.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO  curses(3X), forms(3X), attributes(5)

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

modified 31 Dec 1996          SunOS 5.6          3X-647
form_field_opts, set_field_opts, field_opts_on, field_opts_off, field_opts – forms field option routines

SYNOPSIS

cc [-flag ...] file ... -lform -lcurses [ library ...]
#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);

DESCRIPTION

set_field_opts() turns on the named options of field and turns off all remaining options. Options are boolean values that can be OR-ed together.
field_opts_on() turns on the named options; no other options are changed.
field_opts_off() turns off the named options; no other options are changed.
field_opts() returns the options set for field.

Field Options:

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

set_field_opts, field_opts_on and field_opts_off return one of the following:

E_OK – The function returned successfully.
E_SYSTEM_ERROR – System error.
E_CURRENT – The field is the current field.
ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO curses(3X), forms(3X), attributes(5)

NOTES The header <form.h> automatically includes the headers <eti.h> and <curses.h>.
NAME  
form_fieldtype, new_fieldtype, free_fieldtype, set_fieldtype_arg, set_fieldtype_choice, 
link_fieldtype – forms fieldtype routines

SYNOPSIS  
cc [flag ...] file ... -lform -lcurses [ library ... ]

#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);

DESCRIPTION  
new_fieldtype() creates a new field type. The application programmer must write the
function field_check, which validates the field value, and the function char_check, which
validates each character. free_fieldtype() frees the space allocated for the field type.

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 mak_arg
allocates a structure for the field specific parameters to set_field_type() and returns a
pointer to the saved data. Function copy_arg duplicates the structure created by make_arg.
Function free_arg frees any storage allocated by make_arg or copy_arg.

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.

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.

RETURN VALUES  
Routines that return pointers always return NULL on error. Routines that return an
integer return one of the following:

E_OK – The function returned successfully.
E_SYSTEM_ERROR – System error.
E_BAD_ARGUMENT – An argument is incorrect.
E_CONNECTED – Type is connected to one or more fields.
ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO curses(3X), forms(3X), attributes(5)

NOTES

The header <form.h> automatically includes the headers <eti.h> and <curses.h>.
NAME  
form_field_userptr, set_field_userptr, field_userptr – associate application data with forms

SYNOPSIS  
cc [ flag ... ] file ... -lform -lcurses [ library ... ]
#include <form.h>
int set_field_userptr(FIELD *field, char *ptr);
char *field_userptr(FIELD *field);

DESCRIPTION  
Every field has an associated user pointer that can be used to store pertinent data. set_field_userptr() sets the user pointer of field. field_userptr() returns the user pointer of field.

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.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO  
curses(3X), forms(3X), attributes(5)

NOTES  
The header <form.h> automatically includes the headers <eti.h> and <curses.h>.
NAME form_field_validation, set_field_type, field_type, field_arg – forms field data type validation

SYNOPSIS cc [flag ...] file ... -lform -lcurses [library ...]
#include <form.h>
int set_field_type(FIELD *field, FIELDTYPE *type, ...);
FIELDTYPE *field_type(FIELD *field);
char *field_arg(FIELD *field);

DESCRIPTION set_field_type() associates the specified field type with field. 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_REGEXP.

field_type() returns a pointer to the field type of field. NULL is returned if no field type is assigned.

field_arg() returns a pointer to the field arguments associated with the field type of field. NULL is returned if no field type is assigned.

RETURN VALUES field_type() and field_arg() return NULL on error.

set_field_type() returns one of the following:

E_OK – The function returned successfully.
E_SYSTEM_ERROR – System error.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO curses(3X), forms(3X), attributes(5)

NOTES The header <form.h> automatically includes the headers <eti.h> and <curses.h>.
NAME
form_hook, set_form_init, form_init, set_form_term, form_term, set_field_init, field_init, set_field_term, field_term – assign application-specific routines for invocation by forms

SYNOPSIS
cc [ flag ... ] file ... -lform -lcurses [ library ... ]
#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);

DESCRIPTION
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.

set_form_init() assigns an application-defined initialization function to be called when the form is posted and just after a page change. form_init() returns a pointer to the initialization function, if any.

set_form_term() assigns an application-defined function to be called when the form is unposted and just before a page change. form_term() returns a pointer to the function, if any.

set_field_init() assigns an application-defined function to be called when the form is posted and just after the current field changes. field_init() returns a pointer to the function, if any.

set_field_term() assigns an application-defined function to be called when the form is unposted and just before the current field changes. field_term() returns a pointer to the function, if any.

RETURN VALUES
Routines that return pointers always return NULL on error. Routines that return an integer return one of the following:

E_OK – The function returned successfully.
E_SYSTEM_ERROR – System error.
ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO curses(3X), forms(3X), attributes(5)

NOTES

The header <form.h> automatically includes the headers <eti.h> and <curses.h>.
NAME  form_new, new_form, free_form – create and destroy forms

SYNOPSIS  cc [ flag ... ] file ... -lform -lcurses [ library ... ]
  #include <form.h>
  FORM *new_form(FIELD **fields);
  int free_form(FORM *form);

DESCRIPTION  new_form() creates a new form connected to the designated fields and returns a pointer to the form.
  free_form() disconnects the form from its associated field pointer array and deallocates the space for the form.

RETURN VALUES  new_form() always returns NULL on error.  free_form() returns one of the following:
  E_OK  – The function returned successfully.
  E_BAD_ARGUMENT  – An argument is incorrect.
  E_POSTED  – The form is posted.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO  curses(3X), forms(3X), attributes(5)

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

3X-656  SunOS 5.6  modified 31 Dec 1996
NAME    form_new_page, set_new_page, new_page – forms pagination

SYNOPSIS    cc [ flag ... ] file ... -lform -lcurses [ library ... ]
             #include <form.h>
             int set_new_page(FIELD *field, int bool);
             int new_page(FIELD *field);

DESCRIPTION    set_new_page() marks field as the beginning of a new page on the form.
new_page() returns a boolean value indicating whether or not field begins a new page of
the form.

RETURN VALUES    new_page returns TRUE or FALSE.
set_new_page() returns 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.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO    curses(3X), forms(3X), attributes(5)

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

modified 31 Dec 1996    SunOS 5.6    3X-657
NAME  form_opts, set_form_opts, form_opts_on, form_opts_off – forms option routines

SYNOPSIS  cc [ flag ... ] file ... -lform -lcurses [ library ... ]
#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);

DESCRIPTION  set_form_opts() turns on the named options for form and turns off all remaining options. Options are boolean values which can be OR-ed together.
form_opts_on() turns on the named options; no other options are changed.
form_opts_off() turns off the named options; no other options are changed.
form_opts() returns the options set for form.

Form Options:
O_NL_OVERLOAD Overload the REQ_NEW_LINE form driver request.
O_BS_OVERLOAD Overload the REQ_DEL_PREV form driver request.

RETURN VALUES  set_form_opts(), form_opts_on(), and form_opts_off() return one of the following:

E_OK – The function returned successfully.
E_SYSTEM_ERROR – System error.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO  curses(3X), forms(3X), attributes(5)

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

3X-658 SunOS 5.6 modified 31 Dec 1996
NAME  
form_page, set_form_page, set_current_field, current_field, field_index – set forms current page and field

SYNOPSIS  
```
cc [ flag ... ] file ... -lform -lcurses [ library ... ]
#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);
```

DESCRIPTION  
set_form_page() sets the page number of form to page. form_page() returns the current page number of form.
set_current_field() sets the current field of form to field. current_field() returns a pointer to the current field of form.
field_index() returns the index in the field pointer array of field.

RETURN VALUES  
form_page() returns -1 on error.
current_field() returns NULL on error.
field_index() returns -1 on error.
set_form_page() and set_current_field() return one of the following:

- **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.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO  
curses(3X), forms(3X), attributes(5)

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

modified 31 Dec 1996
SunOS 5.6
3X-659
NAME  form_post, post_form, unpost_form – write or erase forms from associated subwindows

SYNOPSIS  

cc [ flag . . . ] file . . . -lform -lcurses [ library . . . ]  
#include <form.h>  
int post_form(FORM *form);  
int unpost_form(FORM *form);

DESCRIPTION  post_form() writes form 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.  
unpost_form() erases form from its associated subwindow.

RETURN VALUES  These routines return one of the following:  

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.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO  curses(3X), forms(3X), panel_update(3X), panels(3X), attributes(5)

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

3X-660  SunOS 5.6  modified 31 Dec 1996
NAME  
form_userptr, set_form_userptr – associate application data with forms

SYNOPSIS  
cc [ flag ... ] file ... -lform -lcurses [ library ... ]
#include <form.h>
int set_form_userptr(FORM *form, char *ptr);
char *form_userptr(FORM *form);

DESCRIPTION  
Every form has an associated user pointer that can be used to store pertinent data.
set_form_userptr() sets the user pointer of form. form_userptr() returns the user pointer of form.

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.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO  
curses(3X), forms(3X), attributes(5)

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

modified 31 Dec 1996  SunOS 5.6  3X-661
NAME  
form_win, set_form_win, set_form_sub, form_sub, scale_form – forms window and subwindow association routines

SYNOPSIS  
cc [ flag ... ] file ... -lform -lcurses [ library ... ]
#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);

DESCRIPTION  
set_form_win() sets the window of form to win. form_win() returns a pointer to the window associated with form.
set_form_sub() sets the subwindow of form to sub. form_sub() returns a pointer to the subwindow associated with form.
scale_form() returns the smallest window size necessary for the subwindow of form. rows and cols are pointers to the locations used to return the number of rows and columns for the form.

RETURN VALUES  
Routines that return pointers always return NULL on error. Routines that return an integer return one of the following:

E_OK           – The function returned successfully.
E_SYSTEM_ERROR – System error.
E_BAD_ARGUMENT – An argument is incorrect.
E_NOT_CONNECTED – The field is not connected to a form.
E_POSTED       – The form is posted.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO  
curses(3X), forms(3X), attributes(5)

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

3X-662  
SunOS 5.6  
modified 31 Dec 1996
NAME  
fpgetround, fpsetround, fpgetmask, fpsetmask, fpgetsticky, fpsetsticky – IEEE floating-point environment control

SYNOPSIS  
#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);

DESCRIPTION  
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.

The following floating-point exception masks are OR-ed together to form mask.

- 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) */

The following floating-point rounding modes are passed to fpsetround() and returned by fpgetround().

- 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) */

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>.

RETURN VALUES  
fpgetround() returns the current rounding mode.
fpsetround() sets the rounding mode and returns the previous rounding mode.
fpgetmask() returns the current exception masks.
fpsetmask() sets the exception masks and returns the previous setting.
fpgetsticky() returns the current exception sticky flags.
fpsetsticky() sets (clears) the exception sticky flags and returns the previous setting.
ATTRIBUTES | See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO | isnan(3C), attributes(5)

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
fputwc – put wide-character code on a stream

## SYNOPSIS
```
#include <stdio.h>
#include <wchar.h>

wint_t fputwc(wint_t wc, FILE *stream);
```

## DESCRIPTION
The `fputwc()` function writes the character corresponding to the wide-character code `wc` to the output stream pointed to by `stream`, at the position indicated by the associated file-position indicator for the stream (if defined), and advances the indicator appropriately. If the file cannot support positioning requests, or if the stream was opened with append mode, the character is appended to the output stream. If an error occurs while writing the character, the shift state of the output file is left in an undefined state.

The `st_ctime` and `st_mtime` fields of the file will be marked for update between the successful execution of `fputwc()` and the next successful completion of a call to `fflush(3S)` or `fclose(3S)` on the same stream or a call to `exit(2)` or `abort(3C)`.

## RETURN VALUES
Upon successful completion, `fputwc()` returns `wc`. Otherwise, it returns WEOF, the error indicator for the stream is set, and `errno` is set to indicate the error.

## ERRORS
The `fputwc()` function will fail if either the stream is unbuffered or data in the `stream`'s buffer needs to be written, and:

- **EAGAIN**: The `O_NONBLOCK` flag is set for the file descriptor underlying `stream` and the process would be delayed in the write operation.
- **EBADF**: The file descriptor underlying `stream` is not a valid file descriptor open for writing.
- **EFBIG**: An attempt was made to write to a file that exceeds the maximum file size or the process' file size limit.
- **EFBIG**: The file is a regular file and an attempt was made to write at or beyond the offset maximum associated with the corresponding stream.
- **EINTR**: The write operation was terminated due to the receipt of a signal, and no data was transferred.
- **EIO**: A physical I/O error has occurred, or the process is a member of a background process group attempting to write to its controlling terminal, `TOSTOP` is set, the process is neither ignoring nor blocking `SIGTTOU` and the process group of the process is orphaned.
- **ENOSPC**: There was no free space remaining on the device containing the file.
- **EPIPE**: An attempt is made to write to a pipe or FIFO that is not open for reading by any process. A `SIGPIPE` signal will also be sent to the process.

The `fputwc()` function may fail if:

- **ENOMEM**: Insufficient storage space is available.
- **ENXIO**: A request was made of a non-existent device, or the request was outside
fputs (3S) Standard I/O Functions

the capabilities of the device.

EILSEQ The wide-character code wc does not correspond to a valid character.

ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO

exit(2), ulimit(2), abort(3C), fclose(3S), ferror(3S), fflush(3S), fopen(3S), setbuf(3S), attributes(5)
fputws – put wide character string on a stream

#include <stdio.h>
#include <wchar.h>

int fputws(const wchar_t *s, FILE *stream);

The fputws() function writes a character string corresponding to the (null-terminated) wide character string pointed to by ws to the stream pointed to by stream. No character corresponding to the terminating null wide-character code is written.

The st_ctime and st_mtime fields of the file will be marked for update between the successful execution of fputws() and the next successful completion of a call to fflush(3S) or fclose(3S) on the same stream or a call to exit(2) or abort(3C).

Upon successful completion, fputws() returns a non-negative number. Otherwise it returns −1, sets an error indicator for the stream and errno is set to indicate the error.

Refer to fputwc(3S).

The fputws() function does not append a NEWLINE character.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

exit(2), abort(3C), fclose(3S), flush(3S), fopen(3S), fputwc(3S), attributes(5)
NAME           fread, fwrite – buffered binary input/output

SYNOPSIS       #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);

DESCRIPTION     The fread() function reads into an array pointed to by ptr up to nitems items of data from stream, where an item of data is a sequence of bytes (not necessarily terminated by a null byte) of length size. It stops reading bytes if an end-of-file or error condition is encountered while reading stream, or if nitems items have been read. It increments the data pointer in stream to point to the byte following the last byte read if there is one. It does not change the contents of stream. It returns the number of items read.

The fwrite() function writes to the named output stream at most nitems items of data from the array pointed to by ptr, where an item of data is a sequence of bytes (not necessarily terminated by a null byte) of length size. It stops writing when it has written nitems items of data or if an error condition is encountered on stream. It does not change the contents of the array pointed to by ptr. It increments the data pointer in stream by the number of bytes written and returns the number of items written.

A call to fwrite() in buffered mode may return success even though the underlying call to write(2) fails. This can cause unpredictable results. Use either the write() function or the fwrite() function in unbuffered mode. See setvbuf(3).

The ferror() or feof() routines must be used to distinguish between an error condition and end-of-file condition. See ferror(3).

RETURN VALUES  The fread() function returns the number of items read. The fwrite() function returns the number of items written.

If size or nitems is 0, then fread() and fwrite() return 0 and do not effect the state of stream.

If an error occurs, fread() and fwrite() return 0 and set the error indicator for stream.

ERRORS          The fread() function will fail if data needs to be read and:

EOVERFLOW       The file is a regular file, size is greater than 0, the starting position is before the end-of-file, and an attempt was made to read at or beyond the offset maximum associated with the corresponding stream.

The fwrite() function will fail if either the stream is unbuffered or the stream’s buffer needed to be flushed and:

EFBIG           The file is a regular file and an attempt was made to write at or beyond the offset maximum.
ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO

read(2), write(2), fclose(3S), ferror(3S), fopen(3S), getc(3S), gets(3S), printf(3S), putc(3S), puts(3S), scanf(3S), setvbuf(3S), stdio(3S), attributes(5)
freopen (3S)  Standard I/O Functions

NAME  freopen – open a stream

SYNOPSIS  #include <stdio.h>
           FILE *freopen(const char *filename, const char *mode, FILE *stream);

DESCRIPTION  The freopen() function first attempts to flush the stream and close any file descriptor
              associated with stream. Failure to flush or close the file successfully is ignored. The error
              and end-of-file indicators for the stream are cleared.
              The freopen() function opens the file whose pathname is the string pointed to by filename
              and associates the stream pointed to by stream with it. The mode argument is used just as
              in fopen(3).
              The original stream is closed regardless of whether the subsequent open succeeds.
              The largest value that can be represented correctly in an object of type off_t will be esta-
              blished as the offset maximum in the open file description.

RETURN VALUES  Upon successful completion, freopen() returns the value of stream. Otherwise a null
                   pointer is returned and errno is set to indicate the error.

ERRORS  The freopen() function will fail if:
          EACCES  Search permission is denied on a component of the path prefix, or the
                  file exists and the permissions specified by mode are denied, or the file
                  does not exist and write permission is denied for the parent directory of
                  the file to be created.
          EINTR   A signal was caught during freopen().
          EISDIR  The named file is a directory and mode requires write access.
          ELOOP   Too many symbolic links were encountered in resolving path.
          EMFILE  OPEN_MAX file descriptors are currently open in the calling process.
          ENAMETOOLONG  The length of the filename exceeds PATH_MAX or a pathname component
                         is longer than NAME_MAX.
          ENFILE  The maximum allowable number of files is currently open in the system.
          ENOENT  A component of filename does not name an existing file or filename is an
                   empty string.
          ENOSPC  The directory or file system that would contain the new file cannot be
                   expanded, the file does not exist, and it was to be created.
          ENOTDIR A component of the path prefix is not a directory.
          ENXIO   The named file is a character special or block special file, and the device
                   associated with this special file does not exist.
          EOVERFLOW  The current value of the file position cannot be represented correctly in
                      an object of type fpos_t.
The `freopen()` function may fail if:

- **EINVAL**: The value of the `mode` argument is not valid.
- **ENAMETOOLONG**: Pathname resolution of a symbolic link produced an intermediate result whose length exceeds `PATH_MAX`.
- **ENOMEM**: Insufficient storage space is available.
- **ENXIO**: A request was made of a non-existent device, or the request was outside the capabilities of the device.
- **ETXTBSY**: The file is a pure procedure (shared text) file that is being executed and `mode` requires write access.

**USAGE**

The `freopen()` function is typically used to attach the preopened streams 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.

The `freopen()` function has an explicit 64-bit equivalent. See `interface64(5)`.

**ATTRIBUTES**

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**SEE ALSO** `fclose(3S)`, `fdopen(3S)`, `fopen(3S)`, `stdio(3S)`, `attributes(5)`, `interface64(5)`
NAME  frexp – extract mantissa and exponent from double precision number

SYNOPSIS  
```c
#include <math.h>

double frexp(double num, int *exp);
```

DESCRIPTION  The frexp() function breaks a floating-point number into a normalised fraction and an integral power of 2. It stores the integer exponent in the int object pointed to by exp.

RETURN VALUES  The frexp() function returns the value x, such that x is a double with magnitude in the interval [1/2, 1) or 0, and num equals x times 2 raised to the power *exp.

- If num is 0, both parts of the result are 0.
- If num is NaN, NaN is returned and the value of *exp is unspecified.
- If num is ±Inf, num is returned and the value of *exp is unspecified.

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

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  isnan(3M), ldexp(3C), modf(3C), attributes(5)
NAME

fseek, fseeko – reposition a file-position indicator in a stream

SYNOPSIS

```c
#include <stdio.h>

int fseek(FILE *stream, long offset, int whence);
int fseeko(FILE *stream, off_t offset, int whence);
```

DESCRIPTION

The `fseek()` function sets the file-position indicator for the stream pointed to by `stream`. The `fseeko()` function is identical to `fseek()` except for the type of `offset`. The new position, measured in bytes from the beginning of the file, is obtained by adding `offset` to the position specified by `whence`, whose values are defined in `<stdio.h>` as follows:

- `SEEK_SET` set position equal to `offset` bytes.
- `SEEK_CUR` set position to current location plus `offset`.
- `SEEK_END` set position to EOF plus `offset`.

If the stream is to be used with wide character input/output functions, `offset` must either be 0 or a value returned by an earlier call to `ftell(3S)` on the same stream and `whence` must be `SEEK_SET`.

A successful call to `fseek()` clears the end-of-file indicator for the stream and undoes any effects of `ungetc(3S)` and `ungetwc(3S)` on the same stream. After an `fseek()` call, the next operation on an update stream may be either input or output.

If the most recent operation, other than `ftell(3S)`, on a given stream is `fflush(3S)`, the file offset in the underlying open file description will be adjusted to reflect the location specified by `fseek()`.

The `fseek()` function allows the file-position indicator to be set beyond the end of existing data in the file. If data is later written at this point, subsequent reads of data in the gap will return bytes with the value 0 until data is actually written into the gap.

The value of the file offset returned by `fseek()` on devices which are incapable of seeking is undefined.

If the stream is writable and buffered data had not been written to the underlying file, `fseek()` will cause the unwritten data to be written to the file and mark the `st_ctime` and `st_mtime` fields of the file for update.

RETURN VALUES

The `fseek()` and `fseeko()` functions return 0 on success; otherwise they returned −1 and set `errno` to indicate the error.

ERRORS

The `fseek()` and `fseeko()` functions will fail if, either the `stream` is unbuffered or the `stream`’s buffer needed to be flushed, and the call to `fseek()` or `fseeko()` causes an underlying `lseek(2)` or `write(2)` to be invoked:

- `EAGAIN` The `O_NONBLOCK` flag is set for the file descriptor and the process would be delayed in the write operation.
EBADF
The file descriptor underlying the stream file is not open for writing or the stream’s buffer needed to be flushed and the file is not open.

EFBIG
An attempt was made to write a file that exceeds the maximum file size or the process’ file size limit.

EFBIG
The file is a regular file and an attempt was made to write at or beyond the offset maximum associated with the corresponding stream.

EINTR
The write operation was terminated due to the receipt of a signal, and no data was transferred.

EINVAL
The whence argument is invalid. The resulting file-position indicator would be set to a negative value.

EIO
A physical I/O error has occurred, or the process is a member of a background process group attempting to perform a write(2) to its controlling terminal, TOSTOP is set, the process is neither ignoring nor blocking SIGTTOU and the process group of the process is orphaned.

ENOSPC
There was no free space remaining on the device containing the file.

EPIPE
(a) The file descriptor underlying stream is associated with a pipe or FIFO.
(b) An attempt was made to write to a pipe or FIFO that is not open for reading by any process; a SIGPIPE signal will also be sent to the process.

ENXIO
A request was made of a non-existent device, or the request was outside the capabilities of the device.

The fseek() function will fail if:

EOVERFLOW
The resulting file offset would be a value which cannot be represented correctly in an object of type long.

The fseeko() function will fail if:

EOVERFLOW
The resulting file offset would be a value which cannot be represented correctly in an object of type off_t.

USAGE
The fseek() function has an explicit 64-bit equivalent. See interface64(5).

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
getrlimit(2), ulimit(2), fopen(3B), ftell(3S), ftello(), rewind(3S), ungetc(3S), ungetwc(3S), attributes(5), interface64(5)

NOTES
Although on the UNIX system an offset returned by ftell() or ftello() (see ftell(3S)) is measured in bytes, and it is permissible to seek to positions relative to that offset, portability to non-UNIX systems requires that an offset be used by fseek() directly.
Arithmetic may not meaningfully be performed on such an offset, which is not necessarily measured in bytes.
NAME    fsetpos – reposition a file pointer in a stream

SYNOPSIS    
#include <stdio.h>
int fsetpos(FILE *stream, const fpos_t *pos);

DESCRIPTION    The fsetpos() function sets the file position indicator for the stream pointed to by stream according to the value of the object pointed to by pos, which must be a value obtained from an earlier call to fgetpos(3S) on the same stream.

A successful call to fsetpos() function clears the end-of-file indicator for the stream and undoes any effects of ungetc(3S) on the same stream. After an fsetpos() call, the next operation on an update stream may be either input or output.

RETURN VALUES    The fsetpos() function returns 0 if it succeeds; otherwise it returns a non-zero value and sets errno to indicate the error.

ERRORS    The fsetpos() function may fail if:
EBADF    The file descriptor underlying stream is not valid.
ESPIPE    The file descriptor underlying stream is associated with a pipe, a FIFO, or a socket.

USAGE    The fsetpos() function has an explicit 64-bit equivalent. See interface64(5).

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO    lseek(2), fgetpos(3S), fopen(3S), fseek(3S), ftell(3S), rewind(3S), ungetc(3S), attributes(5), interface64(5)
NAME
fsync – synchronize a file’s in-memory state with that on the physical medium

SYNOPSIS
#include <unistd.h>
int fsync(int fildes);

DESCRIPTION
The fsync() function moves all modified data and attributes of the file descriptor fildes to a storage device. When fsync() returns, all in-memory modified copies of buffers associated with fildes have been written to the physical medium. The fsync() function is different from sync(), which schedules disk I/O for all files but returns before the I/O completes. The fsync() function forces all outstanding data operations to synchronized file integrity completion (see fcntl(5) definition of O_SYNC.)

The fsync() function 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.

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
The fsync() function fails if one or more of the following are true:
EBADF The fildes argument is not a valid file descriptor.
EINVAL 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.
ENOSPC There was no free space remaining on the device containing the file.
ETIMEDOUT Remote connection timed out. This occurs when the file is on an NFS file system mounted with the soft option. See mount_nfs(1M).

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Async-Signal-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
mount_nfs(1M), sync(2), fdatasync(3R), attributes(5), fcntl(5)

NOTES
The way the data reach the physical medium depends on both implementation and hardware. The fsync() function returns when the device driver tells it that the write has taken place.
NAME
ftell, ftello – return a file offset in a stream

SYNOPSIS
#include <stdio.h>
long ftell(FILE *stream);
off_t ftello(FILE *stream);

DESCRIPTION
The ftell() function obtains the current value of the file-position indicator for the stream pointed to by stream. The ftello() function is identical to ftell() except for the return type.

RETURN VALUES
Upon successful completion, ftell() returns the current value of the file-position indicator for the stream measured in bytes from the beginning of the file. Otherwise, it returns -1 and sets errno to indicate the error.

ERRORS
The ftell() and ftello() functions will fail if:
EBADF The file descriptor underlying stream is not an open file descriptor.
ESPIPE The file descriptor underlying stream is associated with a pipe, a FIFO, or a socket.
The ftell() function will fail if:
EOVERFLOW The current file offset cannot be represented correctly in an object of type long.
The ftello() function will fail if:
EOVERFLOW The current file offset cannot be represented correctly in an object of type off_t.

USAGE
The ftello() function has an explicit 64-bit equivalent. See interface64(5).

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
lseek(2), fopen(3S), fseek(3S), attributes(5), interface64(5)
### NAME
ftime – get date and time

### SYNOPSIS
```
#include <sys/timeb.h>

int ftime(struct timeb *tp);
```

### DESCRIPTION
The `ftime()` function sets the `time` and `millitm` members of the `timeb` structure pointed to by `tp`. The structure is defined in `<sys/timeb.h>` and contains the following members:

- `time_t time;`
- `unsigned short millitm;`
- `short timezone;`
- `short dstflag;`

The `time` and `millitm` members contain the seconds and milliseconds portions, respectively, of the current time in seconds since 00:00:00 UTC (Coordinated Universal Time), January 1, 1970.

The `timezone` member contains the local time zone. The `dstflag` member contains a flag that, if non-zero, indicates that Daylight Saving time applies locally during the appropriate part of the year.

The contents of the `timezone` and `dstflag` members of `tp` after a call to `ftime()` are unspecified.

### RETURN VALUES
Upon successful completion, the `ftime()` function returns 0. Otherwise, −1 is returned.

### ERRORS
No errors are defined.

### USAGE
For portability to implementations conforming to earlier versions of this document, `time(2)` is preferred over this function.

The millisecond value usually has a granularity greater than one due to the resolution of the system clock. Depending on any granularity (particularly a granularity of one) renders code non-portable.

### SEE ALSO
date(1), time(2), ctime(3C), gettimeofday(3C), timezone(4)
NAME
ftok – generate an IPC key

SYNOPSIS
#include <sys/ipc.h>
key_t ftok(const char *path, int id);

DESCRIPTION
The ftok() function returns a key based on path and id that is usable in subsequent calls to msgget(2), semget(2) and shmget(2). The path argument must be the pathname of an existing file that the process is able to stat(2).

The ftok() function will return the same key value for all paths that name the same file, when called with the same id value, and will return different key values when called with different id values or with paths that name different files existing on the same file system at the same time.

If the file named by path is removed while still referred to by a key, a call to ftok() with the same path and id returns an error. If the same file is recreated, then a call to ftok() with the same path and id is likely to return a different key.

Only the low order 8-bits of id are significant. The behavior of ftok() is unspecified if these bits are 0.

RETURN VALUES
Upon successful completion, ftok() returns a key. Otherwise, ftok() returns (key_t)-1 and sets errno to indicate the error.

ERRORS
The ftok() function will fail if:
EACCES Search permission is denied for a component of the path prefix.
ELOOP Too many symbolic links were encountered in resolving path.
ENAMETOOLONG The length of the path argument exceeds PATH_MAX or a pathname component is longer than NAME_MAX.
ENOENT A component of path does not name an existing file or path is an empty string.
ENOTDIR A component of the path prefix is not a directory.

The ftok() function may fail if:
ENAMETOOLONG Pathname resolution of a symbolic link produced an intermediate result whose length exceeds PATH_MAX.

USAGE
For maximum portability, id should be a single-byte character.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe in multi-thread applications</td>
</tr>
</tbody>
</table>

3C-680 SunOS 5.6 modified 29 Dec 1996
SEE ALSO  msgget(2), semget(2), shmget(2), stat(2), attributes(5)

NOTES  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.
NAME
ftw, nftw – walk a file tree

SYNOPSIS
#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);

DESCRIPTION
The ftw() function recursively descends the directory hierarchy rooted in path. For each
object in the hierarchy, ftw() calls the user-defined function fn, 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:

  FTW_F   The object is a file.
  FTW_D   The object is a directory.
  FTW_DNR The object is a directory that cannot be read. Descendants of the direc-
tory will not be processed.
  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 fn is undefined.

ftw() visits a directory before visiting any of its descendants.
The tree traversal continues until the tree is exhausted, an invocation of fn 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 fn returns a nonzero value, ftw() stops its tree traversal
and returns whatever value was returned by fn.

The function nftw() is similar to ftw() except that it takes an additional argument, flags.
The flags field is used to specify:

  FTW_PHYS  Physical walk, does not follow symbolic links. Otherwise, nftw() will
            follow links but will not walk down any path that crosses itself.
  FTW_MOUNT The walk will not cross a mount point.
  FTW_DEPTH All subdirectories will be visited before the directory itself.
  FTW_CHDIR The walk will change to each directory before reading it.

The function nftw() calls fn 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:

    int base;
    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 descriptors they have opened; they do not close any file descriptors that may have been opened by *fn*.

**RETURN VALUES**
If successful, *ftw()* and *nftw()* return 0. If either function detects an error other than **EACCES**, it returns −1, and sets the error type in *errno*.

**USAGE**
The *ftw()* and *nftw()* functions have explicit 64-bit equivalents. See *interface64*(5).

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>See NOTES below.</td>
</tr>
</tbody>
</table>

**SEE ALSO**
*stat*(2), *longjmp*(3C), *malloc*(3C), *attributes*(5), *interface64*(5)

**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, getacflag, getacmin, getacna, setac, endac – get audit control file information

SYNOPSIS  
cc [ flag ... ] file ... -l bsm -l socket -l nsl -l ntl [ library ... ]
#include <bsm/libbsm.h>
int getacdir( char *dir, int len);
int getacmin( int *min_val);
int getacflag( char *auditstring, int len);
int getacna( char *auditstring, int len);
void setac( void);
void endac( void);

DESCRIPTION  
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 len specifies the length of the buffer dir. On return, dir points to the directory entry.

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.

getacflag() reads the system audit value from the audit_control file and returns the value in auditstring. The parameter len specifies the length of the buffer auditstring.

getacna() reads the system audit value for non-attributable audit events from the audit_control file and returns the value in auditstring. The parameter len specifies the length of the buffer auditstring. Non-attributable events are events that cannot be attributed to an individual user. inetd(1M) and several other daemons record non-attributable events.

Calling setac rewinds the audit_control file to allow repeated searches.
Calling endac closes the audit_control file when processing is complete.

FILES  
/etc/security/audit_control contains default parameters read by the audit daemon, auditd(1M)

RETURN VALUES  
getacdir(), getacflag(), getacna() and getacmin() return:
 0 on success.
-2 on failure and set errno to indicate the error.

getacmin() and getacflag() return:
1 on EOF.

3-684 SunOS 5.6 modified 29 Dec 1996
getacdir() returns:
−1 on EOF.
2 if the directory search had to start from the beginning because one of the other functions was called between calls to getacdir().

These functions return:
−3 if the directory entry format in the audit_control file is incorrect.

getacdir(), getacflg() and getacna() return:
−3 if the input buffer is too short to accommodate the record.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe.</td>
</tr>
</tbody>
</table>

SEE ALSO
audit_warn(1M), bsmconv(1M), inetd(1M), audit_control(4), attributes(5)

NOTES
The functionality described in this man page is available only if the Basic Security Module (BSM) has been enabled. See bsmconv(1M) for more information.
NAME
getauclassent, getauclassent_r, setauclass, endauclass, getauclassnam, getauclassnam_r, getauclassent_r

SYNOPSIS
cc [ flag  ... ] file  ...  -lbsm  -lsocket  -lnsl  -lintl [ library  ... ]
#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 *class_int, const char *name);
struct au_class_ent *getauclassent( void);
struct au_class_ent *getauclassent_r( au_class_ent_t *class_int);
void setauclass( void);
void endauclass( void);

DESCRIPTION
getauclassent() and getauclassnam() each return an audit_class entry.
getauclassnam() searches for an audit_class entry with a given class name name.
getauclassent() enumerates audit_class entries: successive calls to getauclassent() will
return either successive audit_class entries or NULL.
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().
endauclass() may be called to indicate that audit_class processing is complete; the sys-
tem may then close any open audit_class file, deallocate storage, and so forth.

getauclassent_r() and getauclassnam_r() both return a pointer to an audit_class entry as
do their similarly named counterparts. They each take an additional argument, a pointer
to pre-allocated space for an au_class_ent_t, which is returned if the call is successful. To
assure there is enough space for the information returned, the applications programmer
should be sure to allocate AU_CLASS_NAME_MAX and AU_CLASS_DESC_MAX bytes for the
ac_name and ac_desc elements of the au_class_ent_t data structure.
The internal representation of an audit_user entry is an au_class_ent structure defined in
<bsm/libbsm.h> with the following members:
char  *ac_name;
au_class_t  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

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe with exceptions.</td>
</tr>
</tbody>
</table>

All of the functions described in this man-page are MT-Safe except getauclassent() and getauclassnam(). The two functions, getauclassent_r() and getauclassnam_r() have the same functionality as the unsafe functions, but have a slightly different function call interface in order to make them MT-Safe.

SEE ALSO
bsmconv(1M), audit_class(4), audit_event(4), attributes(5)

NOTES
All information is contained in a static area, so it must be copied if it is to be saved.
The functionality described in this man page is available only if the Basic Security Module (BSM) has been enabled. See bsmconv(1M) for more information.
NAME
getauditflags, getauditflagsbin, getauditflagschar – convert audit flag specifications

SYNOPSIS
cc [ flag ... ] file ... -l bsm -lsocket -l nmpl -lintl [ library ... ]
#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);

DESCRIPTION
getauditflagsbin() converts the character representation of audit values pointed to by
auditstring into au_mask_t fields pointed to by masks. 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 audit_control(4).
getauditflagschar() converts the au_mask_t fields pointed to by masks into a string
pointed to by auditstring. If verbose is zero, the short (2-character) flag names are used. If
verbose is non-zero, the long flag names are used. auditstring should be large enough to
contain the ASCII representation of the events.

auditstring contains a series of event names, each one identifying a single audit class,
separated by commas. The au_mask_t fields pointed to by masks correspond to binary
values defined in <bsm/audit.h>, which is read by <bsm/libbsm.h>.

RETURN VALUES
getauditflagsbin() and getauditflagschar() – 1 is returned on error and 0 on success.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
bsmconv(1M), audit.log(4), audit_control(4), attributes(5)

BUGS
This is not a very extensible interface.

NOTES
The functionality described in this man page is available only if the Basic Security
Module (BSM) has been enabled. See bsmconv(1M) for more information.
NAME
getauevent, getauevnam, getauevnum, getauevnonam, setauevent, endauevent,
getauevent_r, getauevnam_r, getauevnum_r – get audit_event entry

SYNOPSIS
cc [ flag ... ] file ... –lbsm –lsocket –lnsl –lintl [ library ... ]
#include <sys/param.h>
#include <bsm/libbsm.h>

struct au_event_ent *getauevent(void);
struct au_event_ent *getauevnam(char *name);
struct au_event_ent *getauevnum(au_event_t event_number);
au_event_t *getauevnonam(char *event_name);
void setauevent(void);
void endauevent(void);
struct au_event_ent *getauevent_r(au_event_ent_t *e);
struct au_event_ent *getauevnam_r(au_event_ent_t *e, char *name);
struct au_event_ent *getauevnum_r(au_event_ent_t *e, au_event_t event_number);

DESCRIPTION
These interfaces document the programming interface for obtaining entries from the
audit_event(4) file. getauevent(), getauevnam(), getauevnum(), getauevent(), getauev-
nonam(), and getauevnum() each return a pointer to an audit_event structure.
getauevent() and getauevent_r() enumerate audit_event entries; successive calls to these
functions will return either successive audit_event entries or NULL.
getauevnam() and getauevnam_r() search for an audit_event entry with a given
event_name.
getauevnum() and getauevnum_r() search for an audit_event entry with a given
event_number.
getauevnonam() searches for an audit_event entry with a given event_name and returns
the corresponding event number.
setauevent() “rewinds” to the beginning of the enumeration of audit_event entries.
Calls to getauevnam(), getauevnum(), getauevnonam(), getauevnum_r(), or
getauevnum_r() may leave the enumeration in an indeterminate state; setauevent() should be
called before the first getauevent() or getauevent_r().
endauevent() may be called to indicate that audit_event processing is complete; the sys-
tem 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 suc-
cessful 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 a struct au_event_ent structure
defined in <bsm/libbsm.h> with the following members:

modified 29 Dec 1996 SunOS 5.6 3-689
au_event_t ae_number;
char *ae_name;
char *ae_desc;
au_class_t ae_class;

RETURN VALUES
getauevent(), getauevnam(), getauevnum(), getauevent_r(), getauevnam_r(), and getauevnum_r() return a pointer to a struct au_event_ent if the requested entry is successfully located; otherwise it returns NULL.
getauevnonam() returns an event number of type au_event_t if it successfully enumerates an entry; otherwise it returns NULL, indicating it could not find the requested event name.

FILES
/etc/security/audit_event Maps audit event numbers to audit event names.
/etc/passwd Stores user-ID to username mappings.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe with exceptions.</td>
</tr>
</tbody>
</table>

The functions getauevent(), getauevnam(), and getauevnum() are not MT-Safe; however, there are equivalent functions: getauevent_r(), getauevnam_r(), and getauevnum_r() — all of which provide the same functionality and a MT-Safe function call interface.

SEE ALSO
bsmconv(1M), getauclassent(3), getpwnam(3C), audit_class(4), audit_event(4), passwd(4), attributes(5)

NOTES
All information for the functions getauevent(), getauevnam(), and getauevnum() is contained in a static area, so it must be copied if it is to be saved.
The functionality described in this man page is available only if the Basic Security Module (BSM) has been enabled. See bsmconv(1M) for more information.
NAME
getauusernam, getauuserent, setauuser, endauuser – get audit_user entry

SYNOPSIS
c
cc [ flag ... ] file ... -lsbm -lsocket -lnsl -lintl [ library ... ]
#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);

DESCRIPTION
getauuserent(), getauusernam(), getauuserent_r(), and getauusernam_r() each return an audit_user entry.
getauusernam() and getauusernam_r() search for an audit_user entry with a given login name name.
getauuserent() and getauuserent_r() enumerate audit_user entries: successive calls to these functions will return either successive audit_user entries or NULL.
setauuser() ‘‘rewinds’’ to the beginning of the enumeration of audit_user entries. Calls to getauusernam() and getauusernam_r() may leave the enumeration in an indeterminate state, so setauuser() should be called before the first getauuserent() or getauuserent_r().
endauuser() may be called to indicate that audit_user processing is complete; the system may then close any open audit_user file, deallocate storage, and so forth.
The two functions getauuserent_r() and getauusernam_r() both take an argument u, which is a pointer to an au_user_ent. This is the pointer that is returned on successful function calls.
The internal representation of an audit_user entry is an au_user_ent structure defined in <bsm/libbsm.h> with the following members:

char *au_name;
uu_mask_t au_always;
uu_mask_t au_never;

RETURN VALUES
getauusernam() returns a pointer to a struct au_user_ent if it successfully locates the requested entry; otherwise it returns NULL.
getauuserent() 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.

modified 29 Dec 1996
SunOS 5.6
3-691
FILES
/etc/security/audit_user Stores per-user audit event mask
/etc/passwd Stores user-id to username mappings

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe with exceptions.</td>
</tr>
</tbody>
</table>

The functions getauusernam() and getauuserent() are not MT-safe. However, the functions getauusernam_r() and getauuserent_r() provide the same functionality with an MT-Safe interfaces.

SEE ALSO bsmconv(1M), getpwnam(3C), audit_user(4), passwd(4), attributes(5)

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

The functionality described in this man page is available only if the Basic Security Module (BSM) has been enabled. See bsmconv(1M) for more information.
**NAME**
getbegyx, getmaxyx, getparyx, getyx – get cursor or window coordinates

**SYNOPSIS**
```c
#include <curses.h>
void getbegyx(WINDOW *win, int y, int x);
void getmaxyx(WINDOW *win, int y, int x);
void getparyx(WINDOW *win, int y, int x);
void getyx(WINDOW *win, int y, int x);
```

**ARGUMENTS**
- `win` Is a pointer to a window.
- `y` stores the y coordinate for the cursor or origin. The `getmaxyx()` macro uses it to store the number of rows in the window.
- `x` stores the x coordinate for the cursor or origin. The `getmaxyx()` macro uses it to store the number of columns in the window.

**DESCRIPTION**
The `getyx()` macro stores the current cursor position of the specified window in `x` and `y`. The `getparyx()` macro stores the `x` and `y` coordinates (relative to the parent window) of the specified window’s origin (upper-left corner). If `win` does not point to a subwindow, `x` and `y` are set to −1.

The `getbegyx()` macro stores the `x` and `y` coordinates of the specified window’s origin (upper-left corner).

The `getmaxyx()` macro stores the numbers of rows in the specified window in `y` and the number of columns in `x`.

**RETURN VALUES**
These macros do not return a value.

**ERRORS**
None.
getc (3S)  Standard I/O Functions

NAME  
getc, getc_unlocked, getchar, getchar_unlocked, fgetc, getw – get character or word from a stream

SYNOPSIS  
#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);

DESCRIPTION  
The getc() function returns the next character (that is, byte) from the named input stream (see intro(3)) as an unsigned char converted to an int. It also moves the file pointer, if defined, ahead one character in stream. The getchar() function is defined as getc(stdin). getc() and getchar() are macros.

The getc_unlocked() and getchar_unlocked() functions are variants of getc() and getchar(), respectively, that do not lock the stream. It is the caller’s responsibility to acquire the stream lock before calling these functions and releasing the lock afterwards; see flockfile(3S) and stdio(3S).

The fgetc() function behaves like getc(), but is a function rather than a macro. The fgetc() function runs more slowly than getc(), but it takes less space per invocation and its name can be passed as an argument to a function.

The getw() function returns the next word (that is, integer) from the named input stream. The getw() function 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. The getw() function assumes no special alignment in the file.

RETURN VALUES  
These functions return the constant EOF at end-of-file or upon an error and set the EOF or error indicator of stream, respectively. Because EOF is a valid integer, ferror() should be used to detect getw() errors.

ERRORS  
The fgetc(), getc(), getchar(), and getw() functions will fail if data needs to be read and:
EROVERFLOW  
The file is a regular file and an attempt was made to read at or beyond the offset maximum associated with the corresponding stream.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>See NOTES below.</td>
</tr>
</tbody>
</table>

3S-694   SunOS 5.6  modified 30 Dec 1996
SEE ALSO intro(3), fclose(3S), ferror(3S), flockfile(3S), fopen(3S), fread(3S), gets(3S), putc(3S), scanf(3S), stdio(3S), ungetc(3S), attributes(5)

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).

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

**NAME**
getcchar – get a wide character string (with rendition) from a cchar_t

**SYNOPSIS**
```
#include <curses.h>
int getcchar(const cchar_t *wcval, wchar_t *wch, attr_t *attrs,
             short *color_pair, void *opt);
```

**ARGUMENTS**
- `wcval` Is a pointer to a cchar_t object.
- `wch` Is a pointer to an object where a wide character string can be stored.
- `attrs` Is a pointer to an object where attributes can be stored.
- `color_pair` Is a pointer to an object where a color pair can be stored.
- `opts` Is reserved for future use. Currently, this must be a null pointer.

**DESCRIPTION**
If `wch` is not a null pointer, the `getcchar()` function splits the cchar_t object pointed to by `wcval` into a wide character string, attributes, and a color pair. It stores the attributes in the location pointed to by `attrs`, the color pair in the location pointed to by `color_pair`, and the wide character string in the location pointed to by `wch`.

If `wch` is a null pointer, the `getcchar()` function simply returns the number of wide characters in the cchar_t object pointed to by `wcval`. The objects pointed to by `attrs` and `color_pair` are not changed.

**RETURN VALUES**
- When `wch` is a null pointer, the `getcchar()` function returns the number of wide characters in the string pointed to by `wcval` including the null terminator.
- When `wch` is not a null pointer, the `getcchar()` function returns OK on success and ERR otherwise.

**ERRORS**
None

**SEE ALSO**
attroff(3XC), can_change_color(3XC), setcchar(3XC)
NAME
getch, wgetch, mvgetch, mvwgetch – get a single-byte character from terminal

SYNOPSIS
#include <curses.h>

int getch(void);
int wgetch (WINDOW *win);
int mvgetch(int y, int x);
int mvwgetch(WINDOW *win, int y, int x);

ARGUMENTS
win  Is a pointer to the window associated with the terminal from which the character is to be read.
y    Is the y (row) coordinate for the position of the character to be read.
x    Is the x (column) coordinate for the position of the character to be read.

DESCRIPTION
The getch() and wgetch() functions get a single-byte character from the terminal associated with the window stdscr or window win, respectively. The mvgetch() and mvwgetch() functions move the cursor to the position specified in stdscr or win, respectively, then get a character.

If the window is not a pad and has been changed since the last call to refresh(3XC), getch() calls refresh() to update the window before the next character is read.

The setting of certain functions affects the behavior of the getch() set of functions. For example, if cbreak(3XC) is set, characters typed by the user are immediately processed. If halfdelay(3XC) is set, getch() waits until a character is typed or returns ERR if no character is typed within the specified timeout period. This timeout can also be specified for individual windows with the delay parameter of timeout(3XC). A negative value waits for input; a value of 0 returns ERR if no input is ready; a positive value blocks until input arrives or the time specified expires (in which case, ERR is returned). If nodelay(3XC) is set, ERR is returned if no input is waiting; if not set, getch() waits until input arrives. Each character will be echoed to the window unless noecho(3XC) has been set.

If keypad handling is enabled (keypad(3XC) is TRUE), the token for the function key is returned. If a character is received that could be the beginning of a function key (for example, ESC), an inter-byte timer is set. If the remainder of the sequence is not received before the time expires, the character is passed through; otherwise, the value of the function key is returned. If notimeout() is set, the inter-byte timer is not used.

The ESC key is typically a prefix key used with function keys and should not be used as a single character.

The following is a list of tokens for function keys that are returned by the getch() set of functions if keypad handling is enabled (some terminals may not support all tokens).
## Constant Values for Function Keys

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEY_BREAK</td>
<td>Break key</td>
</tr>
<tr>
<td>KEY_DOWN</td>
<td>The down arrow key</td>
</tr>
<tr>
<td>KEY_UP</td>
<td>The up arrow key</td>
</tr>
<tr>
<td>KEY_LEFT</td>
<td>The left arrow key</td>
</tr>
<tr>
<td>KEY_RIGHT</td>
<td>The right arrow key</td>
</tr>
<tr>
<td>KEY_HOME</td>
<td>Home key</td>
</tr>
<tr>
<td>KEY_BACKSPACE</td>
<td>Backspace</td>
</tr>
<tr>
<td>KEY_F0</td>
<td>Function keys. Space for 64</td>
</tr>
<tr>
<td>KEY_F(n)</td>
<td>(KEY_F0+(n)) key is reserved</td>
</tr>
<tr>
<td>KEY_DL</td>
<td>Delete line</td>
</tr>
<tr>
<td>KEY_IL</td>
<td>Insert line</td>
</tr>
<tr>
<td>KEY_DC</td>
<td>Delete character</td>
</tr>
<tr>
<td>KEY_IC</td>
<td>Insert char or enter insert mode</td>
</tr>
<tr>
<td>KEY_EIC</td>
<td>Exit insert char mode</td>
</tr>
<tr>
<td>KEY_CLEAR</td>
<td>Clear screen</td>
</tr>
<tr>
<td>KEY_EOS</td>
<td>Clear to end of screen</td>
</tr>
<tr>
<td>KEY_EOL</td>
<td>Clear to end of line</td>
</tr>
<tr>
<td>KEY_SF</td>
<td>Scroll 1 line forward</td>
</tr>
<tr>
<td>KEY_SR</td>
<td>Scroll 1 line backwards</td>
</tr>
<tr>
<td>KEY_NPAGE</td>
<td>Next page</td>
</tr>
<tr>
<td>KEY_PPAGE</td>
<td>Previous page</td>
</tr>
<tr>
<td>KEY_STAB</td>
<td>Set tab</td>
</tr>
<tr>
<td>KEY_CTAB</td>
<td>Clear tab</td>
</tr>
<tr>
<td>KEY_CATAB</td>
<td>Clear all tabs</td>
</tr>
<tr>
<td>KEY_ENTER</td>
<td>Enter or send</td>
</tr>
<tr>
<td>KEY_SRESET</td>
<td>Soft (partial) reset</td>
</tr>
<tr>
<td>KEY_RESET</td>
<td>Reset or hard reset</td>
</tr>
<tr>
<td>KEY_PRINT</td>
<td>Print or copy</td>
</tr>
<tr>
<td>KEY_LL</td>
<td>Home down or bottom (lower left)</td>
</tr>
<tr>
<td>KEY_A1</td>
<td>Upper left of keypad</td>
</tr>
<tr>
<td>KEY_A3</td>
<td>Upper right of keypad</td>
</tr>
<tr>
<td>KEY_B2</td>
<td>Center of keypad</td>
</tr>
<tr>
<td>KEY_C1</td>
<td>Lower left of keypad</td>
</tr>
<tr>
<td>KEY_C3</td>
<td>Lower right of keypad</td>
</tr>
<tr>
<td>KEY_BTAB</td>
<td>Back tab</td>
</tr>
<tr>
<td>KEY_BEG</td>
<td>Beginning key</td>
</tr>
<tr>
<td>KEY_CANCEL</td>
<td>Cancel key</td>
</tr>
<tr>
<td>KEY_CLOSE</td>
<td>Close key</td>
</tr>
<tr>
<td>KEY_COMMAND</td>
<td>Cmd (command) key</td>
</tr>
<tr>
<td>KEY_COPY</td>
<td>Copy key</td>
</tr>
<tr>
<td>KEY_CREATE</td>
<td>Create key</td>
</tr>
<tr>
<td>Constant</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>KEY_END</td>
<td>End key</td>
</tr>
<tr>
<td>KEY_EXIT</td>
<td>Exit key</td>
</tr>
<tr>
<td>KEY_FIND</td>
<td>Find key</td>
</tr>
<tr>
<td>KEY_HELP</td>
<td>Help key</td>
</tr>
<tr>
<td>KEY_MARK</td>
<td>Mark key</td>
</tr>
<tr>
<td>KEY_MESSAGE</td>
<td>Message key</td>
</tr>
<tr>
<td>KEY_MOVE</td>
<td>Move key</td>
</tr>
<tr>
<td>KEY_NEXT</td>
<td>Next object key</td>
</tr>
<tr>
<td>KEY_OPEN</td>
<td>Open key</td>
</tr>
<tr>
<td>KEY_OPTIONS</td>
<td>Options key</td>
</tr>
<tr>
<td>KEY_PREVIOUS</td>
<td>Previous object key</td>
</tr>
<tr>
<td>KEY_REDO</td>
<td>Redo key</td>
</tr>
<tr>
<td>KEY_REFERENCE</td>
<td>Ref(ERENCE) key</td>
</tr>
<tr>
<td>KEY_REFRESH</td>
<td>Refresh key</td>
</tr>
<tr>
<td>KEY_REPLACE</td>
<td>Replace key</td>
</tr>
<tr>
<td>KEY_RESTART</td>
<td>Restart key</td>
</tr>
<tr>
<td>KEY_RESUME</td>
<td>Resume key</td>
</tr>
<tr>
<td>KEY_SAVE</td>
<td>Save key</td>
</tr>
<tr>
<td>KEY_SBEG</td>
<td>Shifted beginning key</td>
</tr>
<tr>
<td>KEY_SCANCEL</td>
<td>Shifted cancel key</td>
</tr>
<tr>
<td>KEY_SCOMMAND</td>
<td>Shifted command key</td>
</tr>
<tr>
<td>KEY_SCOPY</td>
<td>Shifted copy key</td>
</tr>
<tr>
<td>KEY_SCREATE</td>
<td>Shifted create key</td>
</tr>
<tr>
<td>KEY_SDC</td>
<td>Shifted delete char key</td>
</tr>
<tr>
<td>KEY(SDL)</td>
<td>Shifted delete line key</td>
</tr>
<tr>
<td>KEY_SELECT</td>
<td>Select key</td>
</tr>
<tr>
<td>KEY_SEND</td>
<td>Shifted end key</td>
</tr>
<tr>
<td>KEY_SEOL</td>
<td>Shifted clear line key</td>
</tr>
<tr>
<td>KEY_SEXIT</td>
<td>Shifted exit key</td>
</tr>
<tr>
<td>KEY_SFIND</td>
<td>Shifted find key</td>
</tr>
<tr>
<td>KEY_SHELP</td>
<td>Shifted help key</td>
</tr>
<tr>
<td>KEY_SHOME</td>
<td>Shifted home key</td>
</tr>
<tr>
<td>KEY_SIC</td>
<td>Shifted input key</td>
</tr>
<tr>
<td>KEY_SLEFT</td>
<td>Shifted left key</td>
</tr>
<tr>
<td>KEY_SMESSAGES</td>
<td>Shifted messages key</td>
</tr>
<tr>
<td>KEY_SMOVE</td>
<td>Shifted move key</td>
</tr>
<tr>
<td>KEY_SNEXT</td>
<td>Shifted next key</td>
</tr>
<tr>
<td>KEY_SOPTIONS</td>
<td>Shifted options key</td>
</tr>
<tr>
<td>KEY_SPREVIOUS</td>
<td>Shifted previous key</td>
</tr>
<tr>
<td>KEY_SPRINT</td>
<td>Shifted print key</td>
</tr>
<tr>
<td>KEY_SREDO</td>
<td>Shifted redo key</td>
</tr>
</tbody>
</table>

modified 1 Jun 1996

SunOS 5.6

3XC-699
Constant | Description
---|---
KEY_SREPLACE | Shifted replace key
KEY_SRIGHT | Shifted right key
KEY_SRSUME | Shifted resume key
KEY_SSAVE | Shifted save key
KEY_SSUSPEND | Shifted suspend key
KEY_SUNDO | Shifted undo key
KEY_SUSPEND | Suspend key
KEY_UNDO | Undo key

RETURN VALUES
On success, these function return OK. Otherwise, they return ERR.

ERRORS
None

SEE ALSO
cbreak(3XC), echo(3XC), halfdelay(3XC), keypad(3XC), nodelay(3XC), notimeout(3XC), raw(3XC), timeout(3XC)
NAME getcwd – get pathname of current working directory

SYNOPSIS

```c
#include <unistd.h>

extern char *getcwd(char *buf, size_t size);
```

DESCRIPTION

The `getcwd()` function returns a pointer to the current directory pathname. The value of `size` must be at least one greater than the length of the pathname to be returned.

If `buf` is not `NULL`, the pathname will be stored in the space pointed to by `buf`.

If `buf` is a null pointer, `getcwd()` will obtain `size` 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

The `getcwd()` function returns `NULL` with `errno` set if `size` is not large enough, or if an error occurs in a lower-level function.

ERRORS

The `getcwd()` function will fail if one or more of the following are true:

- **EACCES**
  A parent directory cannot be read to get its name.
- **EINVAL**
  The `size` argument is equal to 0.
- **ERANGE**
  The `size` argument 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.

```c
#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);
}
```

ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

modified 27 Jan 1997

SunOS 5.6

3C-701
SEE ALSO  
chdir(2), malloc(3C), attributes(5)

NOTES  
Applications should exercise care when using chdir(2) in conjunction with getcwd(). The current working directory is global to all threads within a process. If more than one thread calls chdir() to change the working directory, a subsequent call to getcwd() could produce results that are unexpected.
NAME
getdate – convert user format date and time

SYNOPSIS
#include <time.h>

struct tm *getdate(const char *string);

extern int getdate_err;

DESCRIPTION
getdate() converts user-definable date and/or time specifications pointed to by string into a tm structure. The tm structure declaration is in the <time.h> header file.

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 strftime(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.

The following conversion specifications are supported:

- % %        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
### Modified Conversion Specifications

Some conversion specifications can be modified by the E and O modifier characters to indicate that an alternative format or specification should be used rather than the one normally used by the unmodified specification. If the alternative format or specification does not exist in the current locale, the behavior be as if the unmodified conversion specification were used.

- `%Ec` locale’s alternative appropriate date and time representation
- `%EC` name of the base year (period) in the locale’s alternative representation
- `%Ex` locale’s alternative date representation
- `%EX` locale’s alternative time representation
- `%Ey` offset from `%EC` (year only) in the locale’s alternative representation
- `%EY` full alternative year representation
- `%Od` day of the month using the locale’s alternative numeric symbols; leading zeros are permitted but not required
- `%Oe` same as `%Od`
- `%OH` hour (24-hour clock) using the locale’s alternative numeric symbols
- `%OI` hour (12-hour clock) using the locale’s alternative numeric symbols
- `%Om` month using the locale’s alternative numeric symbols
- `%OM` minutes using the locale’s alternative numeric symbols
- `%OS` seconds using the locale’s alternative numeric symbols
- `%OU` week number of the year (Sunday as the first day of the week) using the locale’s alternative numeric symbols
- `%Ow` number of the weekday (Sunday=0) using the locale’s alternative numeric symbols
- `%OW` week number of the year (Monday as the first day of the week) using the locale’s alternative numeric symbols
- `%Oy` year (offset from `%C`) in the locale’s alternative representation and using the locale’s alternative numeric symbols

### Internal Format Conversion

The following rules are applied for converting the input specification into the internal format:

- If only the weekday is given, today is assumed if the given day is equal to the current day and next week if it is less.
- If only the month is given, the current month is assumed if the given month is equal to the current month and next year if it is less and no year is given. (The first day of month is assumed if no day is given.)
If the century is given, but the year within the century is not given, the current year within the century is assumed.
If no hour, minute, and second are given, the current hour, minute, and second are assumed.
If no date is given, today is assumed if the given hour is greater than the current hour and tomorrow is assumed if it is less.

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 DATEMSK environment variable is null or undefined.
2. The template file cannot be opened for reading.
3. Failed to get file status information.
4. The template file is not a regular file.
5. An error is encountered while reading the template file.
6. malloc() 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.

<table>
<thead>
<tr>
<th>Invocation</th>
<th>Line in Template</th>
</tr>
</thead>
<tbody>
<tr>
<td>getdate(&quot;11/27/86&quot;)</td>
<td>%m/%d/%y</td>
</tr>
<tr>
<td>getdate(&quot;27.11.86&quot;)</td>
<td>%d.%m.%y</td>
</tr>
<tr>
<td>getdate(&quot;86-11-27&quot;)</td>
<td>%y-%m-%d</td>
</tr>
<tr>
<td>getdate(&quot;Friday 12:00:00&quot;)</td>
<td>%A %H:%M:%S</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Input</th>
<th>Line in Template</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mon</td>
<td>%a</td>
<td>Mon Sep 22 12:19:48 EDT 1986</td>
</tr>
<tr>
<td>Sun</td>
<td>%a</td>
<td>Sun Sep 28 12:19:49 EDT 1986</td>
</tr>
<tr>
<td>Fri</td>
<td>%a</td>
<td>Fri Sep 26 12:19:49 EDT 1986</td>
</tr>
<tr>
<td>September</td>
<td>%B</td>
<td>Mon Sep 1 12:19:49 EDT 1986</td>
</tr>
<tr>
<td>January</td>
<td>%B</td>
<td>Thu Jan 1 12:19:49 EST 1987</td>
</tr>
<tr>
<td>December</td>
<td>%B</td>
<td>Mon Dec 1 12:19:49 EST 1986</td>
</tr>
<tr>
<td>Sep Mon</td>
<td>%b %a</td>
<td>Mon Sep 1 12:19:50 EDT 1986</td>
</tr>
<tr>
<td>Jan Fri</td>
<td>%b %a</td>
<td>Fri Jan 2 12:19:50 EST 1987</td>
</tr>
<tr>
<td>Dec Mon</td>
<td>%b %a</td>
<td>Mon Dec 1 12:19:50 EST 1986</td>
</tr>
<tr>
<td>Jan Wed 1989</td>
<td>%b %a %Y</td>
<td>Wed Jan 4 12:19:51 EST 1989</td>
</tr>
<tr>
<td>Fri 9</td>
<td>%a %H</td>
<td>Fri Sep 26 09:00:00 EDT 1986</td>
</tr>
<tr>
<td>Feb 10:30</td>
<td>%b %H:%S</td>
<td>Sun Feb 1 10:00:30 EST 1987</td>
</tr>
<tr>
<td>10:30</td>
<td>%H:%M</td>
<td>Tue Sep 23 10:30:00 EDT 1986</td>
</tr>
<tr>
<td>13:30</td>
<td>%H:%M</td>
<td>Mon Sep 22 13:30:00 EDT 1986</td>
</tr>
</tbody>
</table>

ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
<tr>
<td>CSI</td>
<td>Enabled</td>
</tr>
</tbody>
</table>

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

NOTES

Subsequent calls to getdate() alter the contents of getdate_err.

Dates before 1902 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  
getdtablesizereal - get the file descriptor table size

SYNOPSIS  
#include <unistd.h>

int getdtablesizereal(void);

DESCRIPTION  
The getdtablesizereal() function is equivalent to getrlimit(2) with the RLIMIT_NOFILE option.

RETURN VALUES  
The getdtablesizereal() function returns the current soft limit as if obtained from a call to getrlimit() with the RLIMIT_NOFILE option.

ERRORS  
No errors are defined.

USAGE  
There is no direct relationship between the value returned by getdtablesizereal() and (OPEN_MAX) defined in <limits.h>.

SEE ALSO  
close(2), getrlimit(2), open(2), setrlimit(2), select(3C)

NOTES  
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 getdtablesizereal() function returns the current maximum size of this table by calling the getrlimit() function.
NAME
getenv – return value for environment name

SYNOPSIS
#include <stdlib.h>
char *getenv(const char *name);

DESCRIPTION
getenv() searches the environment list (see environ(5)) for a string of the form
name=value and, if the string is present, returns a pointer to the value in the current
environment.

RETURN VALUES
If successful, getenv() returns a pointer to the value in the current environment; otherwise, it returns a null pointer.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
exec(2), putenv(3C), attributes(5), 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        getexecname – return pathname of executable

SYNOPSIS    #include <stdlib.h>
            const char * getexecname(void);

DESCRIPTION The getexecname() function returns the pathname of the executable that started the process as passed as the first argument to execve(char * file, ...).
            
            Normally this is an absolute pathname, as the majority of commands are executed by the shells who append the command name to the users PATH components. If this is not an absolute path, getcwd(3C) can be prepended to it to create an absolute path.

RETURN VALUES If successful, getexecname() returns a pointer to the executables pathname; otherwise, it returns 0.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO      exec(2), getcwd(3C), attributes(5)

NOTES         The getexecname() function obtains the executable pathname from the AT_SUN_EXECNAME aux vector. These vectors are made available to dynamically linked processes only.
            
            A successful call to one of the exec family of functions will always have in the aux vector. The associate pathname is guaranteed to be less than, or equal, to PATH_MAX, not counting the trailing null byte, which is always present.
NAME
getfauditflags – generates the process audit state

SYNOPSIS
cc [ flag ...] file ... -lbsm -lssocket -linsl -lintl [ library ... ]
#include <sys/param.h>
#include <bsm/libbsm.h>

int getfauditflags(au_mask_t *usremasks, au_mask_t *usrdmasks, au_mask_t *lastmasks);

DESCRIPTION
getfauditflags() generates a process audit state by combining the audit masks passed as parameters with the system audit masks specified in the audit_control(4) file.
getfauditflags() obtains the system audit value by calling getacflg() (see getacinfo(3)).
usremasks points to au_mask_t fields which contains two values. The first value defines which events are always to be audited when they succeed. The second value defines which events are always to be audited when they fail.
usrdmasks also points to au_mask_t fields which contains two values. The first value defines which events are never to be audited when they succeed. The second value defines which events are never to be audited when they fail.
The structures pointed to by usremasks and usrdmasks may be obtained from the audit_user(4) file by calling getauusernam() which returns a pointer to a structure containing all audit_user(4) fields for a user.
The output of this function is stored in lastmasks which is a pointer of type au_mask_t as well. The first value defines which events are to be audited when they succeed and the second defines which events are to be audited when they fail.
Both usremasks and usrdmasks override the values in the system audit values.

RETURN VALUES
-1 is returned on error and 0 on success.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe.</td>
</tr>
</tbody>
</table>

SEE ALSO
bsmconv(1M), getacinfo(3), getauditflags(3), getauusernam(3), audit.log(4), audit_control(4), audit_user(4), attributes(5)

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

modified 29 Dec 1996     SunOS 5.6     3-711
NAME
getgrnam, getgrnam_r, getgrent, getgrent_r, getgrgid, getgrgid_r, setgrent, endgrent, 
__fgetgrent, fgetgrent_r -- get group entry

SYNOPSIS
#include <grp.h>

struct group *getgrnam(const char *name);

struct group *getgrnam_r(const char *name, struct group *grp,
    char *buffer, int buflen);

struct group *getgrent(void);

struct group *getgrent_r(struct group *grp, char *buffer, int buflen);

struct group *getgrgid(gid_t gid);

struct group *getgrgid_r(gid_t gid, struct group *grp, char *buffer, int buflen);

void setgrent(void);

void endgrent(void);

struct group *fgetgrent(FILE *f);

struct group *fgetgrent_r(FILE *f, struct group *grp, char *buffer, int buflen);

POSIX
cc [ flag ... ] file ... -D_POSIX_PTHREAD_SEMANTICS [ library... ]

int getgrnam_r(const char *name, struct group *grp, char *buffer, size_t bufsize,
    struct group **result);

int getgrgid_r(gid_t gid, struct group *grp, char *buffer, size_t bufsize,
    struct group **result); DESCRIPTION section of this page.

DESCRIPTION
These functions are used to obtain entries describing user groups. Entries can come from 
any of the sources for group specified in the /etc/nsswitch.conf file (see 
nsswitch.conf(4)).

getgrnam() searches for an entry with the group name specified by the character string 
p parameter name.

getgrgid() searches for an entry with the (numeric) group id specified by gid.

The functions setgrent(), getgrent(), and endgrent() are used to enumerate group entries 
from the database. 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. Success- 

cive calls to getgrent() return either successive entries or NULL, indicating the end of 
the enumeration.

dgrent() 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().

3C-712 SunOS 5.6 modified 20 Mar 1997
\texttt{fgetgrent()}, unlike the other functions above, does not use \texttt{nsswitch.conf}; it reads and parses the next line from the stream \texttt{f}, which is assumed to have the format of the \texttt{group} file (see \texttt{group(4)}).

### Reentrant Interfaces

The functions \texttt{getgrnam()}, \texttt{getgrgid()}, \texttt{getgrent()}, and \texttt{fgetgrent()} use static storage that is re-used in each call, making them unsafe for multithreaded applications.

The parallel functions \texttt{getgrnam_r()}, \texttt{getgrgid_r()}, \texttt{getgrent_r()}, and \texttt{fgetgrent_r()} provide reentrant interfaces for these operations.

Each reentrant interface performs the same operation as its non-reentrant counterpart, named by removing the \texttt{"_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 \texttt{grp} must be a pointer to a \texttt{struct group} structure allocated by the caller. On successful completion, the function returns the group entry in this structure. The parameter \texttt{buffer} is a pointer to a buffer supplied by the caller, used as storage space for the group data. All of the pointers within the returned \texttt{struct group} \texttt{grp} point to data stored within this buffer; see \texttt{RETURN VALUES}. The buffer must be large enough to hold all the data associated with the group entry. The parameter \texttt{buflen} (or \texttt{bufsize} for the POSIX versions; see \texttt{standards(5)}) should give the size in bytes of \texttt{buffer}. The POSIX versions place a pointer to the modified \texttt{grp} structure in the \texttt{result} parameter, instead of returning a pointer to this structure.

For enumeration in multithreaded applications, the position within the enumeration is a process-wide property shared by all threads. \texttt{setgrent()} may be used in a multithreaded application but resets the enumeration position for all threads. If multiple threads interleave calls to \texttt{getgrent_r()}, the threads will enumerate disjoint subsets of the group database. Like their non-reentrant counterparts, \texttt{getgrnam_r()} and \texttt{getgrgid_r()} leave the enumeration position in an indeterminate state.

### RETURN VALUES

Group entries are represented by the \texttt{struct group} structure defined in \texttt{<grp.h>}:\n
\begin{verbatim}
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 */
};
\end{verbatim}

The functions \texttt{getgrnam()}, \texttt{getgrnam_r()}, \texttt{getgrgid()}, and \texttt{getgrgid_r()} each return a pointer to a \texttt{struct group} if they successfully locate the requested entry; otherwise they return \texttt{NULL}. The POSIX functions \texttt{getgrnam_r()} and \texttt{getgrgid_r()} return 0 upon success or the error number in case of failure.

The functions \texttt{getgrent()}, \texttt{getgrent_r()}, \texttt{fgetgrent()}, and \texttt{fgetgrent_r()} each return a pointer to a \texttt{struct group} if they successfully enumerate an entry; otherwise they return \texttt{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 `grp` pointer that was supplied by the caller.

**ERRORS**

The reentrant functions `getgrnam_r()`, `getgrgid_r()`, `getgrent_r()`, and `fgetgrent_r()` return `NULL` and set `errno` to `ERANGE` (or in the case of POSIX functions `getgrnam_r()` and `getgrgid_r()` return the `ERANGE` error) if the length of the buffer supplied by caller is not large enough to store the result. See `Intro(2)` for the proper usage and interpretation of `errno` in multithreaded applications.

**FILES**

`/etc/group`
`/etc/nsswitch.conf`

**ATTRIBUTES**

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>See “Reentrant Interfaces” in DESCRIPTION.</td>
</tr>
</tbody>
</table>

**SEE ALSO**

`getpwnam(3C)`, `group(4)`, `nsswitch.conf(4)`, `passwd(4)`, `attributes(5)`, `standards(5)`

**NOTES**

When compiling multithread programs, see `Intro(3)`, *Notes On Multithread Applications*.

Programs that use the interfaces described in this manual page cannot be linked statically since the implementations of these functions employ dynamic loading and linking of shared objects at run time.

Use of the enumeration interfaces `getgrent()` and `getgrent_r()` is discouraged; enumeration is supported for the group file, NIS, and NIS+, but in general is not efficient and may not be supported for all database sources. The semantics of enumeration are discussed further in `nsswitch.conf(4)`.

Previous releases allowed the use of “+” and “-” entries in `/etc/group` to selectively include and exclude entries from NIS. The primary usage of these entries is superseded by the name service switch, so the “+/-” form *may not be supported in future releases*.

If required, the “+/-” functionality can still be obtained for NIS by specifying `compat` as the source for `group`

If the “+/-” functionality is required in conjunction with NIS+, specify both `compat` as the source for `group` and `nisplus` as the source for the pseudo-database `group_compat`. See `group(4)`, and `nsswitch.conf(4)` for details.

Solaris 2.4 and earlier releases provided definitions of the `getgrnam_r()` and `getgrgid_r()` functions as specified in POSIX.1c Draft 6. The final POSIX.1c standard changed the interface for these functions. Support for the Draft 6 interface is provided for compatibility only and may not be supported in future releases. New applications and libraries...
should use the POSIX standard interface.

For POSIX.1c complaint applications, the \_POSIX_PTHREAD_SEMANTICS and \_REENTRANT flags are automatically turned on by defining the \_POSIX_C\_SOURCE flag with a value \textgeq 199506L.
NAME  
gethostbyname, gethostbyname_r, gethostbyaddr, gethostbyaddr_r, gethostent,  
gethostent_r, sethostent, endhostent – get network host entry

SYNOPSIS  
cc [ flag ... ] file ... -lnsl [ library ... ]  
#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);

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

gethostbyname() searches for information for a host with the hostname specified by the character-string parameter name.

gethostbyaddr() searches for information for a host with a given host address. The parameter type specifies the family of the address. This should be one of the address families defined in <sys/socket.h>. The parameter addr 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 addr parameter. The parameter len specifies the length of the buffer indicated by addr.

The functions sethostent(), gethostent(), and endhostent() are used to enumerate host entries from the database.

sethostent() sets (or resets) the enumeration to the beginning of the set of host entries. This function should be called before the first call to gethostent(). Calls to gethostbyname() and gethostbyaddr() leave the enumeration position in an indeterminate state. If the stayopen flag is non-zero, the system may keep allocated resources such as open file descriptors until a subsequent call to endhostent().

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

dendhostent() 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_errno` 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>`:

```c
struct hostent {
    char  *h_name;  /* canonical name of host */
    char **h_aliases; /* alias list */
    int   h_addrtype; /* host address type */
    int   h_length;  /* length of address */
    char **h_addr_list; /* list of addresses */
};
```

See the EXAMPLES section below for information about how to retrieve a “.” separated Internet IP address string from the `h_addr_list` field of `struct hostent`.

The functions `gethostbyname()`, `gethostbyname_r()`, `gethostbyaddr()`, and `gethostbyaddr_r()` each return a pointer to a `struct hostent` if they successfully locate the requested entry; otherwise they return `NULL`.

modified 16 May 1997

SunOS 5.6

3N-717
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	%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

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>See “Reentrant Interfaces” in DESCRIPTION.</td>
</tr>
</tbody>
</table>

SEE ALSO
inet(3N), netdir(3N), hosts(4), netconfig(4), nsswitch.conf(4), attributes(5), fns(5), fns_policies(5), netdb(5)

WARNINGS
The reentrant interfaces gethostbyname_r(), gethostbyaddr_r(), and gethostent_r() are included in this release on an uncommitted basis only, and are subject to change or removal in future minor releases.

modified 16 May 1997
SunOS 5.6
3N-719
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()`.

When `gethostbyname()` or `gethostbyname_r()` are given a slash-separated FNS host name to look up (see `fns(5)` and `fns_policies(5)`), then the host is looked up using FNS directly and `nsswitch.conf(4)` is not consulted.

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.
<table>
<thead>
<tr>
<th>NAME</th>
<th>gethostid – get unique identifier of current host</th>
</tr>
</thead>
</table>
| SYNOPSIS   | `#include <unistd.h>`
|            | `long gethostid(void);`                         |
| 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(1), sysinfo(2)`                         |
NAME
gethostname, sethostname – get or set name of current host

SYNOPSIS
int gethostname(char *name, int namelen);
int sethostname(char *name, int namelen);

DESCRIPTION
The gethostname() function returns the standard host name for the current processor, as previously set by sethostname(). The namelen argument specifies the size of the array pointed to by name. The returned name is null-terminated unless insufficient space is provided.

The sethostname() function sets the name of the host machine to be name, which has length namelen. This call is restricted to the super-user and is normally used only when the system is bootstrapped.

RETURN VALUES
Upon successful completion, gethostname() and sethostname() return 0. Otherwise, they return −1 and set errno to indicate the error.

ERRORS
The gethostname() and sethostname() functions will fail if:

EFAULT The name or namelen argument gave an invalid address.

The sethostname() function will fail if:

EPERM The caller was not the super-user.

SEE ALSO
sysinfo(2), uname(2), gethostid(3C)

NOTES
Host names are limited to MAXHOSTNAMELEN characters, currently 256, defined in the <netdb.h> header.
NAME  gethostname – get name of current host

SYNOPSIS  cc [ flag ... ] file ... -lxnet [ library ... ]
           #include <unistd.h>
           int gethostname(char *name, size_t namelen);

DESCRIPTION  The gethostname() function returns the standard host name for the current machine. The namelen argument specifies the size of the array pointed to by the name argument. The returned name is null-terminated, except that if namelen is an insufficient length to hold the host name, then the returned name is truncated and it is unspecified whether the returned name is null-terminated.
            Host names are limited to 255 bytes.

RETURN VALUES  On successful completion, 0 is returned. Otherwise, −1 is returned.

ERRORS  No errors are defined.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  uname(2), gethostid(3C), attributes(5), unistd(5)
NAME
gethrtime, gethrvtime – get high resolution time

SYNOPSIS
#include <sys/time.h>
hrtime_t gethrtime(void);
hrtime_t gethrvtime(void);

DESCRIPTION The gethrtime() function returns the current high-resolution real time. Time is expressed as nanoseconds since some arbitrary time in the past; it is not correlated in any way to the time of day, and thus is not subject to resetting or drifting by way of adjtime(2) or settimeofday(3C). The hi-res timer is ideally suited to performance measurement tasks, where cheap, accurate interval timing is required.

The gethrvtime() function returns the current high-resolution LWP virtual time, expressed as total nanoseconds of execution time. This function requires that micro state accounting be enabled with the ptime utility (see proc(1)).

The gethrtime() and gethrvtime() functions both return an hrtime_t, which is a 64-bit (long long) signed integer.

EXAMPLE The following code fragment measures the average cost of getpid(2):

```c
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);
```

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO proc(1), adjtime(2), gettimeofday(3C), settimeofday(3C), attributes(5)

NOTES Although the units of hi-res time are always the same (nanoseconds), the actual resolution is hardware dependent. Hi-res time is guaranteed to be monotonic (it won’t go backward, it won’t periodically wrap) and linear (it won’t occasionally speed up or slow down for adjustment, like the time of day can), but not necessarily unique: two sufficiently proximate calls may return the same value.
NAME  getlogin, getlogin_r — get login name

SYNOPSIS  
```c
#include <unistd.h>
char *getlogin(void);
char *getlogin_r(char *name, int namelen);
```

POSIX  
```c
int getlogin_r(char *name, size_t namesize);
```

DESCRIPTION  The `getlogin()` function returns a pointer to the login name as found in /var/adm/utmp. It may be used in conjunction with `getpwnam(3C)` to locate the correct password file entry when the same user ID is shared by several login names.

If `getlogin()` is called within a process that is not attached to a terminal, it returns a null pointer. The correct procedure for determining the login name is to call `cuserid(3S)`, or to call `getlogin()` and if it fails to call `getpwuid(3C)`.

The `getlogin_r()` function has the same functionality as `getlogin()` except that the caller must supply a buffer `name` with length `namelen` to store the result. The `name` buffer must be at least `_POSIX_LOGIN_NAME_MAX` bytes in size (defined in `<limits.h>`). The POSIX version (see `standards(5)`) of `getlogin_r()` takes a `namesize` parameter of type `size_t`.

RETURN VALUES  Upon successful completion, `getlogin()` returns a pointer to the login name or a null pointer if the user's login name cannot be found. Otherwise it returns a null pointer and sets `errno` to indicate the error.

The POSIX `getlogin_r()` returns 0 if successful, or the error number upon failure.

ERRORS  The `getlogin()` function may fail if:

- `EMFILE`  (OPEN_MAX) file descriptors are currently open in the calling process.
- `ENFILE`  The maximum allowable number of files is currently open in the system.
- `ENXIO`  The calling process has no controlling terminal.

The `getlogin_r()` function will fail if:

- `ERANGE`  The size of the buffer is smaller than the result to be returned.
- `EINVAL`  And entry for the current user was not found in the /var/adm/utmp file.

USAGE  The return value may point to static data whose content is overwritten by each call. Three names associated with the current process can be determined:

- `getpwuid(geteuid())` returns the name associated with the effective user ID of the process;
- `getlogin()` returns the name associated with the current login activity; and
- `getpwuid(getuid())` returns the name associated with the real user ID of the process.

modified 7 Apr 1997

SunOS 5.6

3C-725
FILES
/var/adm/utmp    accounting file

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>See NOTES below.</td>
</tr>
</tbody>
</table>

SEE ALSO
getuid(2), getuid(2), cuserid(3S), getgrnam(3C), getpwnam(3C), getpwuid(3C), utmp(4), attributes(5), standards(5)

NOTES
When compiling multithread programs, see Intro(3), Notes On Multithread Applications.
The return values point to static data whose content is overwritten by each call.
The getlogin() function is unsafe in multi-thread applications. The getlogin_r() function should be used instead.

Solaris 2.4 and earlier releases provided a getlogin_r() as specified in POSIX.1c Draft 6. The final POSIX.1c standard changed the interface as described above. Support for the Draft 6 interface is provided for compatibility only and may not be supported in future releases. New applications and libraries should use the POSIX standard interface.
NAME  getmntent, getmntany, hasmntopt, putmntent – get mnttab file information

SYNOPSIS  
#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);

DESCRIPTION  getmntent() and getmntany() each fill in the structure pointed to by mp 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:

char  *mnt_special;
char  *mnt_mountp;
char  *mnt_fstype;
char  *mnt_mntopts;
char  *mnt_time;

The fields have meanings described in mnttab(4).

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 fp until a match is found between a line in the file and mpref. mpref matches the line if all non-null entries in mpref match the corresponding fields in the file. Note that these routines do not open, close, or rewind the file.

hasmntopt() scans the mnt_mntopts field of the mnttab structure mnt for a substring that matches opt. It returns the address of the substring if a match is found, otherwise it returns 0.

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.

RETURN VALUES  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:

MNT_TOOLONG A line in the file exceeded the internal buffer size of MNT_LINE_MAX.
MNT_TOOFEW A line in the file contains too few fields.
MNT_TOOMANY A line in the file contains too many fields.
On success, `putmntent()` returns the number of bytes printed to the specified file and on failure returns EOF.

**FILES**
/etc/mnttab

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

**SEE ALSO**
mnttab(4), attributes(5)

**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

cc [flag ...] file ... -lsocket -lnsl [ library ... ]
#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);

DESCRIPTION section of this page.

DESCRIPTION

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

getnetbyname() searches for a network entry with the network name specified by the
character string parameter name.

getnetbyaddr() searches for a network entry with the network address specified by net.
The parameter type specifies the family of the address. This should be one of the address
families defined in <sys/socket.h>. See the NOTES section below for more information.
The functions setnetent(), getnetent(), and endnetent() are used to enumerate network
entries from the database.

setnetent() sets (or resets) the enumeration to the beginning of the set of network entries.
This function should be called before the first call to getnetent(). Calls to getnet-
byname() and getnetbyaddr() leave the enumeration position in an indeterminate state.
If the stayopen flag is non-zero, the system may keep allocated resources such as open file
descriptors until a subsequent call to endnetent().

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

denetnetent() 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().
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` 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.

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

```c
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

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**SEE ALSO**
`inet(3N)`, `networks(4)`, `nsswitch.conf(4)`, `attributes(5)`, `netdb(5)`

**WARNINGS**
The reentrant interfaces `getnetbyname_r()`, `getnetbyaddr_r()` and `getnetent_r()` are included in this release on an uncommitted basis only, and are subject to change or removal in future minor releases.

**NOTES**
The current implementation of these functions only return or accept network numbers for the Internet address family (type `AF_INET`). The functions described in `inet(3N)` may be helpful in constructing and manipulating addresses and network numbers in this form.

Programs that use the interfaces described in this manual page cannot be linked statically since the implementations of these functions employ dynamic loading and linking of shared objects at run time.

When compiling 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)`.

modified 16 May 1997
SunOS 5.6
3N-731
NAME  
getnetconfig, setnetconfig, endnetconfig, getnetconfigent, freenetconfigent, nc_perror, nc_sperror – get network configuration database entry

SYNOPSIS  
#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_sperror(void);

DESCRIPTION  
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, /etc/netconfig. In addition to the routines for accessing the netconfig database, Network Selection includes the environment variable NETPATH (see environ(5)) and the NETPATH access routines described in getnetpath(3N).

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. handlep is the handle obtained through setnetconfig().

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 not 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. handlep 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().

getnetconfigent() returns a pointer to the struct netconfig structure corresponding to netid. It returns NULL if netid is invalid (that is, does not name an entry in the netconfig database).

freenetconfigent() frees the netconfig structure pointed to by netconfigp (previously returned by getnetconfigent()).

nc_perror() prints a message to the standard error indicating why any of the above routines failed. The message is prepended with the string msg and a colon. A NEWLINE is appended at the end of the message.
**Network Functions**

**getnetconfig(3N)**

nc_sperror() 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_sperror() 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_sperror() 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, getnetconfigent() returns a pointer to the struct netconfig structure corresponding to netid; otherwise it returns NULL.

nc_sperror() returns a pointer to a buffer which contains the error message string. This buffer is overwritten on each call. In multithreaded applications, this buffer is implemented as thread-specific data.

**ATTRIBUTES**

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**SEE ALSO**

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

*ONC+ Developer’s Guide*

*Transport Interfaces Programming Guide*

modified 30 Dec 1996 SunOS 5.6 3N-733
getnetgrent (3N) Network Functions

NAME
getnetgrent, getnetgrent_r, setnetgrent, endnetgrent, innetgr – get network group entry

SYNOPSIS
int getnetgrent(char **machinep, char **userp, char **domainp);
int getnetgrent_r(char **machinep, char **userp, char **domainp, char *buffer, int buflen);
void setnetgrent(const char *netgroup);
void endnetgrent(void);
int innetgr(const char *netgroup, const char *machine, const char *user, const char *domain);

DESCRIPTION
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)).

These functions consult the source specified for netgroup in the /etc/nsswitch.conf file (see nsswitch.conf(4)).

The function innetgr() returns 1 if there is a netgroup netgroup that contains the specified machine, user, domain triple as a member; otherwise it returns 0. Any of the supplied pointers machine, user, and domain may be NULL, signifying a ”wild card” that matches all values in that position of the triple.

The innetgr() function is safe for use in single-threaded and multi-threaded applications.

The functions setnetgrent(), getnetgrent(), and endnetgrent() are used to enumerate the members of a given network group.

The function setnetgrent() establishes the network group specified in the parameter netgroup as the current group whose members are to be enumerated.

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.

When calling either getnetgrent() or getnetgrent_r(), addresses of the three character pointers are used as arguments; i.e.:

char *mp, *up, *dp;
getnetgrent(&mp, &up, &dp);

Upon successful return from getnetgrent(), the pointer mp points to a string containing the name of the machine part of the member triple, up points to a string containing the user name and dp points to a string containing the domain name. If the pointer returned for mp, up, or dp is NULL, it signifies that the element of the netgroup contains wild card specifier in that position of the triple.

The pointers returned by getnetgrent() point into a buffer allocated by setnetgrent() that is 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.

3N-734 SunOS 5.6 modified 30 Dec 1996
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 `buffer` should be a pointer to a buffer allocated by the caller and the length of this buffer should be specified by the parameter `buflen`. The buffer must be large enough to hold the data associated with the triple. The `getnetgrent_r()` function is safe for use both in single-threaded and multi-threaded applications.

The function `endnetgrent()` frees the space allocated by the previous `setnetgrent()` call. The equivalent of an `endnetgrent()` implicitly performed whenever a `setnetgrent()` call is made to a new network group.

Note that while `setnetgrent()` and `endnetgrent()` are safe for use in multi-threaded applications, the effect of each is process-wide. Calling `setnetgrent()` resets the enumeration position for all threads. If multiple threads interleave calls to `getnetgrent_r()` each will enumerate a disjoint subset of the netgroup. Thus the effective use of these functions in multi-threaded applications may require coordination by the caller.

**ERRORS**
The function `getnetgrent_r()` will return 0 and set `errno` to `ERANGE` if the length of the buffer supplied by caller is not large enough to store the result. See `intro(2)` for the proper usage and interpretation of `errno` in multi-threaded applications.

**FILES**
`/etc/nsswitch.conf`

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>See DESCRIPTION section.</td>
</tr>
</tbody>
</table>

**SEE ALSO**
`netgroup(4) nsswitch.conf(4), attributes(5)`

**WARNINGS**
The function `getnetgrent_r()` is included in this release on an uncommitted basis only, and is subject to change or removal in future minor releases.

**NOTES**
Only the Network Information Services, NIS and NIS+, are supported as sources for the `netgroup` database.

Programs that use the interfaces described in this manual page cannot be linked statically since the implementations of these functions employ dynamic loading and linking of shared objects at run time.

When compiling multi-threaded applications, see `Intro(3), Notes On Multithread Applications`, for information about the use of the `_REENTRANT` flag.
NAME  
getnetpath, setnetpath, endnetpath – get /etc/netconfig entry corresponding to NETPATH component

SYNOPSIS  
#include <netconfig.h>

struct netconfig *getnetpath(void *handlep);

void *setnetpath(void);

int endnetpath(void *handlep);

DESCRIPTION  
The routines described on this page are part of the Network Selection component. They provide the application access to the system network configuration database, /etc/netconfig, as it is “filtered” by the NETPATH environment variable (see environ(5)). See getnetconfig(3N) for other routines that also access the network configuration database directly. The NETPATH variable is a list of colon-separated network identifiers.

getnetpath() returns a pointer to the netconfig database entry corresponding to the first valid NETPATH component. The netconfig entry is formatted as a struct netconfig. On each subsequent call, getnetpath() returns a pointer to the netconfig entry that corresponds to the next valid NETPATH component. getnetpath() can thus be used to search the netconfig database for all networks included in the NETPATH variable. When NETPATH has been exhausted, getnetpath() returns NULL.

A call to setnetpath() “binds” to or “rewinds” NETPATH. setnetpath() must be called before the first call to getnetpath() and may be called at any other time. It returns a handle that is used by getnetpath().

getnetpath() silently ignores invalid NETPATH components. A NETPATH component is invalid if there is no corresponding entry in the netconfig database.

If the NETPATH variable is unset, getnetpath() behaves as if NETPATH were set to the sequence of “default” or “visible” networks in the netconfig database, in the order in which they are listed.

endnetpath() may be called to “unbind” from NETPATH when processing is complete, releasing resources for reuse. Programmers should be aware, however, that endnetpath() frees all memory allocated by getnetpath() for the struct netconfig data structure. endnetpath() returns 0 on success and -1 on failure (for example, if setnetpath() was not called previously).

RETURN VALUES  
setnetpath() returns a handle that is used by getnetpath(). In case of an error, setnetpath() returns NULL. nc_perror() or nc_sperror() can be used to print out the reason for failure. See getnetconfig(3N).
When first called, `getnetpath()` returns a pointer to the `netconfig` database entry corresponding to the first valid `NETPATH` component. When `NETPATH` has been exhausted, `getnetpath()` returns NULL.

`endnetpath()` returns 0 on success and -1 on failure (for example, if `setnetpath()` was not called previously).

**ATTRIBUTES**

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**SEE ALSO**

`getnetconfig(3N)`, `netconfig(4)`, `attributes(5)`, `environ(5)`

*ONC+ Developer’s Guide*

*Transport Interfaces Programming Guide*
NAME
getnstr, getstr, mvgetnstr, mvgetstr, mvwgetnstr, mvwgetstr, wgetnstr, wgetstr – get a multibyte character string from terminal

SYNOPSIS
#include <curses.h>

int getnstr(char *str, int n);
int getstr(char *str);
int mvgetnstr(int y, int x, char *str, int n);
int mvgetstr(int y, int x, char *str);
int mvwgetnstr(WINDOW *win, int y, int x, char *str, int n);
int mvwgetstr(WINDOW *win, int y, int x, char *str);
int wgetnstr(WINDOW *win, char *str, int n);
int wgetstr(WINDOW *win, char *str);

ARGUMENTS
str Is a pointer to the area where the character string is to be placed.
n Is the maximum number of characters to read from input.
y Is the y (row) coordinate of starting position of character string to be read.
x Is the x (column) coordinate of starting position of character string to be read.
win Points to the window associated with the terminal from which the character is to be read.

DESCRIPTION
The getstr() and wgetstr() functions get a character string from the terminal associated with the window stdscr or window win, respectively. The mvgetstr() and mvwgetstr() functions move the cursor to the position specified in stdscr or win, respectively, then get a character string.

These functions call wgetch(3XC) and place each received character in str until a newline is received, which is also placed in str. The erase and kill characters set by the user are processed.

The getnstr(), mvgetnstr(), mvwgetnstr() and wgetnstr() functions read at most n characters. These functions are used to prevent overflowing the input buffer.

The getnstr(), wgetnstr(), mvgetnstr(), and mvwgetnstr() functions only return complete multibyte characters. If the area pointed to by str is not large enough to hold at least one character, these functions fail.

RETURN VALUES
On success, these functions return OK. Otherwise, they return ERR.

ERRORS
None.

SEE ALSO
getch(3XC)
**NAME**
getn_wstr, get_wstr, mvgetn_wstr, mvget_wstr, mvwgetn_wstr, mvwget_wstr, wgetn_wstr, wget_wstr – get a wide character string from terminal

**SYNOPSIS**
```
#include <curses.h>

int getn_wstr(wint_t *wstr, int n);
int get_wstr(wint_t *wstr);
int mvgetn_wstr(int y, int x, wint_t *wstr, int n);
int mvget_wstr(int y, int x, wint_t *wstr);
int mvwgetn_wstr(WINDOW *win, int y, int x, wint_t *wstr, int n);
int mvwget_wstr(WINDOW *win, int y, int x, wint_t *wstr);
int wgetn_wstr(WINDOW *win, wint_t *wstr, int n);
int wget_wstr(WINDOW *win, wint_t *wstr);
```

**ARGUMENTS**
- `wstr` Is a pointer to the area where the character string is to be placed.
- `n` Is the maximum number of characters to read from input.
- `y` Is the y (row) coordinate of starting position of character string to be read.
- `x` Is the x (column) coordinate of starting position of character string to be read.
- `win` points to the window associated with the terminal from which the character is to be read.

**DESCRIPTION**
The `get_wstr()` and `wget_wstr()` functions get a wide character string from the terminal associated with the window `stdscr` or window `win`, respectively. The `mvget_wstr()` and `mvwget_wstr()` functions move the cursor to the position specified in `stdscr` or `win`, respectively, then get a wide character string.

These functions call `wget_wch(3XC)` and place each received character in `wstr` until a newline character, end-of-line character, or end-of-file character is received, which is also placed in `wstr`. The erase and kill characters set by the user are processed.

The `getn_wstr()`, `mvgetn_wstr()`, `mvwgetn_wstr()` and `wgetn_wstr()` functions read at most `n` characters. These functions are used to prevent overflowing the input buffer.

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

**ERRORS**
None.

**SEE ALSO**
`get_wch(3XC)`, `getnstr(3XC)`
NAME  getopt – get option letter from argument vector

SYNOPSIS  

#include <stdlib.h>

int getopt(int argc, char * argv, const char *optstring);
extern char *optarg;
extern int optind, opterr, optopt;

DESCRIPTION  getopt( ) returns the next option letter in argv that matches a letter in optstring. It sup-
supports all the rules of the command syntax standard (see intro(1)). Since all new com-
mands 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.

optstring 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. optarg is set to point to the
start of the option argument on return from getopt( ).

g getopt( ) places in optind the argv index of the next argument to be processed. optind 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).

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 optstring 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  The following code fragment shows how one might process the arguments for a com-
mand that can take the mutually exclusive options a and b, and the option o, which
requires an argument:

#include <stdlib.h>
#include <stdio.h>
main (int argc, char **argv)
{
   int c;
   extern char *optarg;
   extern int optind;
   int aflag = 0;
   int bflag = 0;
   int errorflag = 0;
   char *ofile = NULL;
while ((c = getopt(argc, argv, "abo:")) != EOF) {
  switch (c) {
    case 'a':
      if (bflag)
        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 \ldots\n\n"};
    exit (2);
  }
  for (; optind < argc; optind++)
    (void)printf("\%s\n", argv[optind]);
  return 0;
}

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO intro(1), getopt(1), getopt(3C), getsubopt(3C), setlocale(3C), gettext(3C), attributes(5)

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 of: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  #include <unistd.h>
           int getpagesize(void);

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.
The getpagesize() function is equivalent to sysconf(_SC_PAGE_SIZE) and sysconf(_SC_PAGESIZE).

RETURN VALUES  The getpagesize() function returns the current page size.

ERRORS  No errors are defined.

USAGE  The value returned by getpagesize() need not be the minimum value that malloc(3C) can allocate. Moreover, the application cannot assume that an object of this size can be allocated with malloc().

SEE ALSO  pagesize(1), brk(2), getrlimit(2), mmap(2), mprotect(2), munmap(2), malloc(3C), msync(3C), sysconf(3C)
NAME  getpass, getpassphrase – read a string of characters without echo

SYNOPSIS  
#include <unistd.h>
char *getpass(const char *prompt);
char *getpassphrase(const char *prompt);

DESCRIPTION  The getpass() function opens the process’ controlling terminal, writes to that device the null-terminated string prompt, disables echoing, reads a string of characters up to the next newline character or EOF, restores the terminal state and closes the terminal.

The function getpassphrase() is identical to getpass(), except that it will read and return a string of up to 256 characters in length.

RETURN VALUES  Upon successful completion, getpass() returns a pointer to a null-terminated string of at most {PASS_MAX} bytes that were read from the terminal device. If an error is encountered, the terminal state is restored and a null pointer is returned.

ERRORS  The getpass() and getpassphrase() functions may fail if:
EINTR  The function was interrupted by a signal.
EIO  The process is a member of a background process attempting to read from its controlling terminal, the process is ignoring or blocking the SIGTTIN signal or the process group is orphaned.
EMFILE {OPEN_MAX} file descriptors are currently open in the calling process.
ENFILE  The maximum allowable number of files is currently open in the system.
ENXIO  The process does not have a controlling terminal.

USAGE  The return value points to static data whose content may be overwritten by each call.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO  attributes(5)
NAME
getpeername – get name of connected peer

SYNOPSIS
cc [ flag ...] file ... -lsocl -lssl [ library ...]
int getpeername(int s, struct sockaddr *name, int *namelen);

DESCRIPTION
getpeername() returns the name of the peer connected to socket s. The int pointed to by
the namelen parameter should be initialized to indicate the amount of space pointed to by
name. On return it contains the actual size of the name returned (in bytes). 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 s 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 opera-
tion to complete.
ENOTCONN The socket is not connected.
ENOTSOCK The argument s is not a socket.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
accept(3N), bind(3N), getsockname(3N), socket(3N), attributes(5), socket(5)

modified 16 May 1997 SunOS 5.6 3N-745
NAME  getpeername — get the name of the peer socket

SYNOPSIS  

```c
cc [ flag ... ] file ... -lxnet [ library ... ]
#include <sys/socket.h>
int getpeername(int socket, struct sockaddr *address, size_t *address_len);
```

DESCRIPTION  The `getpeername()` function retrieves the peer address of the specified socket, stores this address in the `sockaddr` structure pointed to by the `address` argument, and stores the length of this address in the object pointed to by the `address_len` argument.

If the actual length of the address is greater than the length of the supplied `sockaddr` structure, the stored address will be truncated.

If the protocol permits connections by unbound clients, and the peer is not bound, then the value stored in the object pointed to by `address` is unspecified.

RETURN VALUES  Upon successful completion, 0 is returned. Otherwise, -1 is returned and `errno` is set to indicate the error.

ERRORS  The `getpeername()` function will fail if:

- **EBADF**  The `socket` argument is not a valid file descriptor.
- **ENOTSOCK**  The `socket` argument does not refer to a socket.
- **ENOTCONN**  The socket is not connected or otherwise has not had the peer prespecified.
- **EINVAL**  The socket has been shut down.
- **EOPNOTSUPP**  The operation is not supported for the socket protocol.

The `getpeername()` function may fail if:

- **ENOBUFS**  Insufficient resources were available in the system to complete the call.
- **ENOSR**  There were insufficient STREAMS resources available for the operation to complete.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  `getsockoptname(3XN)`, `socket(3XN)`, `attributes(5)`, `socket(5)`

3XN-746  SunOS 5.6  modified 16 May 1997
NAME
getpriority, setpriority – get or set process scheduling priority

SYNOPSIS
#include <sys/resource.h>

int getpriority(int which, id_t who);
int setpriority(int which, id_t who, int priority);

DESCRIPTION
The getpriority() function obtains the current scheduling priority of a process, process

group, or user. The setpriority() function sets the scheduling priority of a process, pro-

cess group, or user.

Target processes are specified by the values of the which and who arguments. The which

argument may be one of the following values: PRIO_PROCESS, PRIO_PGRP, or

PRIO_USER, indicating that the who argument is to be interpreted as a process ID, a pro-

cess group ID, or a user ID, respectively. A 0 value for the who argument specifies the
current process, process group, or user.

If more than one process is specified, getpriority() returns the highest priority (lowest

numerical value) pertaining to any of the specified processes, and setpriority() sets the

priorities of all of the specified processes to the specified value.

The default priority is 0; negative priorities cause more favorable scheduling. While the

range of valid priority values is [−20, 20], implementations may enforce more restrictive

limits. If the value specified to setpriority() is less than the system’s lowest supported

priority value, the system’s lowest supported value is used; if it is greater than the

system’s highest supported value, the system’s highest supported value is used.

Only a process with appropriate privileges can raise its priority (that is, assign a lower

numerical priority value).

RETURN VALUES
Upon successful completion, getpriority() returns an integer in the range from −20 to 20.

Otherwise, −1 is returned and errno is set to indicate the error.

Upon successful completion, setpriority() returns 0. Otherwise, −1 is returned and errno

is set to indicate the error.

ERRORS
The getpriority() and setpriority() functions will fail if:

ESRCH No process could be located using the which and who argument values

specified.

EINVAL The value of the which argument was not recognized, or the value of the who

argument is not a valid process ID, process group ID, or user ID.

In addition, setpriority() may fail if:

EPERM A process was located, but neither the real nor effective user ID of the execut-

ing process is the privileged user or match the effective user ID of the process

whose priority is being changed.

EACCES A request was made to change the priority to a lower numeric value (that is,
to a higher priority) and the current process does not have appropriate
privileges.

**USAGE**

The effect of changing the scheduling priority may vary depending on the process-scheduling algorithm in effect.

Because `getpriority()` can return the value \(-1\) on successful completion, it is necessary to set `errno` to 0 prior to a call to `getpriority()`. If `getpriority()` returns the value \(-1\), then `errno` can be checked to see if an error occurred or if the value is a legitimate priority.

**SEE ALSO**

`nice(1)`, `renice(1)`, `fork(2)`
Network Functions

NAME
getprotobyname, getprotobyname_r, getprotobynumber, getprotobynumber_r, getprotoent, getprotoent_r, setprotoent, endprotoent – get protocol entry

SYNOPSIS
cc [ flag ... ] file ... -lsocket -lnsl [ library ... ]
#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);

DESCRIPTION
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.

The reentrant routines require additional parameters which are used to return results data. result is a pointer to a struct protoent structure and will be where the returned results will be stored. buffer is used as storage space for elements of the returned results. buflen is the size of buffer and should be large enough to contain all returned data. buflen must be at least 1024 bytes.

getprotobyname_r(), getprotobynumber_r(), and getprotoent_r() each return a protocol entry.

The entry may come from one of the following sources: the protocols file (see protocols(4)), the NIS maps “protocols.bynam” and “protocols.bynumber”, and the NIS+ table “protocols”. The sources and their lookup order are specified in the /etc/nsswitch.conf file (see nsswitch.conf(4) for details). Some name services such as NIS will return only one name for a host, whereas others such as NIS+ or DNS will return all aliases.

getprotobyname_r() and getprotobynumber_r() sequentially search from the beginning of the file until a matching protocol name or protocol number is found, or until an EOF is encountered.

getprotobyname() and getprotobynumber() have the same functionality as getprotobyname_r() and getprotobynumber_r() except that a static buffer is used to store returned results. These routines are unsafe in a multi-threaded application.

modified 16 May 1997
SunOS 5.6
3N-749
getprotobyname_r() enumerates protocol entries: successive calls to getprotobyname_r() will return either successive protocol entries or NULL. Enumeration may not be supported by some sources. Note that if multiple threads call getprotobyname_r(), each will retrieve a sub-set of the protocol database.

getprotot() has the same functionality as getprotobyname_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 getprotobyname_r() or getprotobyname(). Note that setprotoent() has process-wide scope, and “rewinds” the protocol entries for all threads calling getprotobyname_r() as well as main-thread calls to getprotobyname().

endprotoent() may be called to indicate that protocol processing is complete; the system may then close any open protocols file, deallocate storage, and so forth. It is legitimate, but possibly less efficient, to call more protocol routines after endprotoent().

The internal representation of a protocol entry is a protoent structure defined in <netdb.h> with the following members:

```c
    char  *p_name;
    char  **p_aliases;
    int p_proto;
```

RETURN VALUES

getprotobyname_r(), getprotobyname(), getprotobynumber_r(), and getprotobynumber() return a pointer to a struct protoent if they successfully locate the requested entry; otherwise they return NULL.

getprotot_r() and getprotot() 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 getprotot_r() will fail if the following is true:

ERANGE

length of the buffer supplied by caller is not large enough to store the result.

FILES

/etc/protocols
/etc/nsswitch.conf

ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>See NOTES below.</td>
</tr>
</tbody>
</table>
SEE ALSO

intro(3), nsswitch.conf(4), protocols(4), attributes(5), netdb(5)

NOTES

Although getprotobyname_r(), getprotobynumber_r(), and getprotoent_r() are not mentioned by POSIX.4a Draft 6, they were added to complete the functionality provided by similar thread-safe functions. These interfaces are subject to change to be compatible with the “spirit” of POSIX.4a when it is approved as a standard.

When compiling multithreaded applications, see intro(3), Notes On Multithread Applications, for information about the use of the _REENTRANT flag.

The routines getprotobyname_r(), getprotobynumber_r(), and getprotoent_r() are reentrant and multi-thread safe. The reentrant interfaces can be used in single-threaded as well as multi-threaded applications and are therefore the preferred interfaces.

The routines getprotobyname(), getprotobyaddr(), and getprotoent() use static storage, so returned data must be copied if it is to be saved. Because of their use of static storage for returned data, these routines are not safe for multi-threaded applications.

setprotoent() and endprotoent() have process-wide scope, and are therefore not safe in multi-threaded applications.

Use of getprotoent_r() and getprotoent() is discouraged; enumeration is well-defined for the protocols file and is supported (albeit inefficiently) for NIS and NIS+, but in general may not be well-defined. The semantics of enumeration are discussed in nsswitch.conf(4).

BUGS

Only the Internet protocols are currently understood.

Programs that call getprotobyname_r() or getprotobynumber_r() routines cannot be linked statically since the implementation of these routines requires dynamic linker functionality to access shared objects at run time.
NAME
getpublickey, getsecretkey, publickey – retrieve public or secret key

SYNOPSIS
#include <rpc/rpc.h>
#include <rpc/key_prot.h>

int getpublickey(const char netname[MAXNETNAMELEN],
                 char publickey[HEXKEYBYTES+1]);
int getsecretkey(const char netname[MAXNETNAMELEN],
                 char secretkey[HEXKEYBYTES+1], const char *passwd);

DESCRIPTION
getpublickey() and getsecretkey() get public and secret keys for netname. The key may come from one of the following sources: the /etc/publickey file (see publickey(4)) or the NIS map "publickeybyname" 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)).

getsecretkey() has an extra argument, passwd, used to decrypt the encrypted secret key stored in the database.

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 secretkey[0] will be set to NULL.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
secure_rpc(3N), nsswitch.conf(4), publickey(4), attributes(5)

WARNINGS
If getpublickey() gets the public key from any source other than NIS+, all authenticated NIS+ operations may fail. To ensure that this does not happen, edit the nsswitch.conf(4) file to make sure that the public key is obtained from NIS+.
NAME  getpw – get passwd entry from UID

SYNOPSIS  #include <stdlib.h>
            int getpw(uid_t uid, char *buf);

DESCRIPTION  getpw() searches the user data base for a user id number that equals uid, copies the line
              of the password file in which uid was found into the array pointed to by buf, and returns
              0. getpw() returns non-zero if uid 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

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


SEE ALSO  getpwnam(3C), passwd(4), attributes(5)

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  #include <pwd.h>
            struct passwd *getpwnam(const char *name);
            struct passwd *getpwnam_r(const char *name, struct passwd *pwd,
                                         char *buffer, int buflen);
            struct passwd *getpwent(void);
            struct passwd *getpwent_r(struct passwd *pwd, char *buffer, int buflen);
            struct passwd *getpwuid(uid_t uid);
            struct passwd *getpwuid_r(uid_t uid, struct passwd *pwd, char *buffer, int buflen);
            void setpwent(void);
            void endpwent(void);
            struct passwd *fgetpwent(FILE *f);
            struct passwd *fgetpwent_r(FILE *f, struct passwd *pwd, char *buffer, int buflen);

POSIX    cc [ flag ... ] file ... −D_POSIX_PTHREAD_SEMANTICS [ library... ]
            int getpwnam_r(const char *name, struct passwd *pwd, char *buffer, size_t bufsize
                struct passwd **result);
            int getpwuid_r(uid_t uid, struct passwd *pwd, char *buffer, size_t bufsize
                struct passwd **result);

DESCRIPTION  These functions are used to obtain password entries. Entries can come from any of the
sources for passwd specified in the /etc/nsswitch.conf file (see nsswitch.conf(4)).

The getpwnam() function searches for a password entry with the login name specified by
the character string parameter name.

The getpwuid() function searches for a password entry with the (numeric) user ID
specified by the parameter uid.

The setpwent(), getpwent(), and endpwent() functions are used to enumerate password
entries from the database. setpwent() sets (or resets) the enumeration to the beginning
of the set of password entries. This function should be called before the first call to
getpwent(). Calls to getpwnam() and getpwuid() leave the enumeration position in an
indeterminate state. Successive calls to getpwent() return either successive entries or
NULL, indicating the end of the enumeration.

The endpwent() function may be called to indicate that the caller expects to do no further
password retrieval operations; the system may then close the password file, deallocate
resources it was using, and so forth. It is still allowed, but possibly less efficient, for the
process to call more password functions after calling endpwent().

3C-754 SunOS 5.6 modified 20 Mar 1997
The `fgetpwent()` function, unlike the other functions above, does not use `nsswitch.conf`; it reads and parses the next line from the stream `f`, which is assumed to have the format of the `passwd` file. See `passwd(4)`.

### Reentrant Interfaces

The functions `getpwnam()`, `getpwuid()`, `getpwent()`, and `fgetpwent()` use static storage that is re-used in each call, making these routines unsafe for use in multithreaded applications.

The parallel functions `getpwnam_r()`, `getpwuid_r()`, `getpwent_r()`, and `fgetpwent_r()` provide reentrant interfaces for these operations.

Each reentrant interface performs the same operation as its non-reentrant counterpart, named by removing the “_r” suffix. The reentrant interfaces, however, use buffers supplied by the caller to store returned results, and are safe for use in both single-threaded and multithreaded applications.

Each reentrant interface takes the same parameters as its non-reentrant counterpart, as well as the following additional parameters. The parameter `pwd` must be a pointer to a `struct passwd` structure allocated by the caller. On successful completion, the function returns the password entry in this structure. The parameter `buffer` is a pointer to a buffer supplied by the caller, used as storage space for the password data. All of the pointers within the returned `struct passwd` `pwd` point to data stored within this buffer; see `RETURN VALUES`. The buffer must be large enough to hold all the data associated with the password entry. The parameter `buflen` (or `bufsize` for the POSIX versions; see `standards(5)`) should give the size in bytes of `buffer`. The POSIX versions place a pointer to the modified `pwd` structure in the `result` parameter, instead of returning a pointer to this structure.

For enumeration in multithreaded applications, the position within the enumeration is a process-wide property shared by all threads. The `setpwent()` function may be used in a multithreaded application but resets the enumeration position for all threads. If multiple threads interleave calls to `getpwent_r()`, the threads will enumerate disjoint subsets of the password database.

Like their non-reentrant counterparts, `getpwnam_r()` and `getpwuid_r()` leave the enumeration position in an indeterminate state.

### RETURN VALUES

Password entries are represented by the `struct passwd` structure defined in `<pwd.h>`:

```c
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 */
};
```

modified 20 Mar 1997 SunOS 5.6 3C-755
The `getpwnam()`, `getpwnam_r()`, `getpwuid()`, and `getpwuid_r()` functions each return a pointer to a `struct passwd` if they successfully locate the requested entry; otherwise they return `NULL`. The POSIX functions `getpwnam_r()` and `getpwuid_r()` return 0 upon success, or the error number in case of failure.

The `getpwent()`, `getpwent_r()`, `fgetpwent()`, and `fgetpwent_r()` functions each return a pointer to a `struct passwd` if they successfully enumerate an entry; otherwise they return `NULL`, indicating the end of the enumeration.

The `getpwnam()`, `getpwuid()`, `getpwent()`, and `fgetpwent()` functions use static storage, so returned data must be copied before a subsequent call to any of these functions if the data is to be saved.

When the pointer returned by the reentrant functions `getpwnam_r()`, `getpwuid_r()`, `getpwent_r()`, and `fgetpwent_r()` is non-null, it is always equal to the `pwd` pointer that was supplied by the caller.

**ERRORS**

The reentrant functions `getpwnam_r()`, `getpwuid_r()`, `getpwent_r()`, and `fgetpwent_r()` will return `NULL` and set `errno` to `ERANGE` (or in the case of POSIX functions `getpwnam_r()` and `getpwuid_r()` return the `ERANGE` error) if the length of the buffer supplied by caller is not large enough to store the result. See `Intro(2)` for the proper usage and interpretation of `errno` in multithreaded applications.

**FILES**

- `/etc/passwd`
- `/etc/shadow`
- `/etc/nsswitch.conf`

**ATTRIBUTES**

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>See “Reentrant Interfaces” in DESCRIPTION.</td>
</tr>
</tbody>
</table>

**SEE ALSO**

- `nispasswd(1)`, `passwd(1)`, `yppasswd(1)`, `Intro(2)`, `Intro(3)`, `cuserid(3S)`, `getgrnam(3C)`, `getlogin(3C)`, `getspnam(3C)`, `nsswitch.conf(4)`, `passwd(4)`, `shadow(4)`, `attributes(5)`, `standards(5)`

**NOTES**

When compiling multithread programs, see `Intro(3)`, `Notes On Multithread Applications`.

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.

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)`.

3C-756  SunOS 5.6    modified 20 Mar 1997
Previous releases allowed the use of `+' and `-' entries in /etc/passwd to selectively include and exclude NIS entries. The primary usage of these `+/-' entries is superseded by the name service switch, so the `+/-' form may not be supported in future releases.

If required, the `+/-' functionality can still be obtained for NIS by specifying compatible as the source for passwd.

If the `+/-' functionality is required in conjunction with NIS+, specify both compatible as the source for passwd and nisplus as the source for the pseudo-database passwd_compat. See passwd(4), shadow(4), and nsswitch.conf(4) for details.

If the `+/-' is used, both /etc/shadow and /etc/passwd should have the same `+' and `-' entries to ensure consistency between the password and shadow databases.

If a password entry from any of the sources contains an empty uid or gid field, that entry will be ignored by the files, NIS, and NIS+ name service switch backends. This will cause the user to appear unknown to the system.

If a password entry contains an empty gecos, home directory, or shell field, getpwnam() and getpwnam_r() return a pointer to a null string in the respective field of the passwd structure.

If the shell field is empty, login(1) automatically assigns the default shell. See login(1).

Solaris 2.4 and earlier releases provided definitions of the getpwnam_r() and getpwuid_r() functions as specified in POSIX.1c Draft 6. The final POSIX.1c standard changed the interface for these functions. Support for the Draft 6 interface is provided for compatibility only and may not be supported in future releases. New applications and libraries should use the POSIX standard interface.

For POSIX.1c complaint applications, the _POSIX_PTHREAD_SEMANTICS and _REENTRANT flags are automatically turned on by defining the _POSIX_C_SOURCE flag with a value >= 199506L.
getrpcbyname (3N)

NAME
getrpcbyname, getrpcbyname_r, getrpcbyname_r, getrpcbyname_r, getrpcent, getrpcent_r, setrpcent, endrpcent – get RPC entry

SYNOPSIS
c [ flag ... ] file ... -lsl [ library ... ]
#include <rpc/rpcent.h>
struct rpcent *getrpcbyname(const char *name);
struct rpcent *getrpcbyname_r(const char *name, struct rpcent *result, char *buffer, int buflen);
struct rpcent *getrpcbyname(const int number);
struct rpcent *getrpcbyname_r(const int number, struct rpcent *result, char *buffer, int buflen);
struct rpcent *getrpcent(void);
struct rpcent *getrpcent_r(struct rpcent *result, char *buffer, int buflen);
void setrpcent(const int stayopen);
void endrpcent(void);

DESCRIPTION
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 /etc/nsswitch.conf file (see nsswitch.conf(4)).

getrpcbyname() searches for an entry with the RPC service name specified by the parameter name.

getrpcbyname() searches for an entry with the RPC program number number.
The functions setrpcent(), getrpcent(), and endrpcent() are used to enumerate RPC entries from the database.

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 getrpcbyname() leave the enumeration position in an indeterminate state. If the stayopen flag is non-zero, the system may keep allocated resources such as open file descriptors until a subsequent call to endrpcent().

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

derpcent() 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().

Reentrant Interfaces
The functions getrpcbyname(), getrpcbyname(), and getrpcbyname() 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
getrpcent_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. setrpcent() may be used in a multithreaded application but resets the enumeration position for all threads. If multiple threads interleave calls to getrpcent_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 getrpcent() and getrpcent_r() each return a pointer to a struct rpcent if they successfully enumerate an entry; otherwise they return NULL, indicating the end of the enumeration.

The functions getrpcbyname(), getrpcbynumber(), and getrpcent() use static storage, so returned data must be copied before a subsequent call to any of these functions if the data is to be saved.

modified 30 Dec 1996

SunOS 5.6

3N-759
When the pointer returned by the reentrant functions `getrpcbyname_r()`, `getrpcbyname_r()`, and `getrpcent_r()` is non-NULL, it is always equal to the result pointer that was supplied by the caller.

**ERRORS**
The reentrant functions `getrpcbyname_r()`, `getrpcbyname_r()` and `getrpcent_r()` will return NULL and set `errno` to `ERANGE` if the length of the buffer supplied by caller is not large enough to store the result. See `intro(2)` for the proper usage and interpretation of `errno` in multithreaded applications.

**FILES**
`/etc/rpc`
`/etc/nsswitch.conf`

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>See “Reentrant Interfaces” in DESCRIPTION.</td>
</tr>
</tbody>
</table>

**SEE ALSO**
`rpcinfo(1M)`, `rpc(3N)`, `nsswitch.conf(4)`, `rpc(4)`, `attributes(5)`

**WARNINGS**
The reentrant interfaces `getrpcbyname_r()`, `getrpcbyname_r()`, and `getrpcent_r()` are included in this release on an uncommitted basis only, and are subject to change or removal in future minor releases.

**NOTES**
Programs that use the interfaces described in this manual page cannot be linked statically since the implementations of these functions employ dynamic loading and linking of shared objects at run time.

When compiling multithreaded applications, see `Intro(3)`, Notes On Multithread Applications, for information about the use of the `_REENTRANT` flag.

Use of the enumeration interfaces `getrpcent()` and `getrpcent_r()` is discouraged; enumeration may not be supported for all database sources. The semantics of enumeration are discussed further in `nsswitch.conf(4)`.
C Library Functions

NAME
getrusage – get information about resource utilization

SYNOPSIS
#include <sys/resource.h>

int getrusage(int who, struct rusage *r_usage);

DESCRIPTION
The getrusage() function provides measures of the resources used by the current process or its terminated and waited-for child processes. If the value of the who argument is RUSAGE_SELF, information is returned about resources used by the current process. If the value of the who argument is RUSAGE_CHILDREN, information is returned about resources used by the terminated and waited-for children of the current process. If the child is never waited for (for instance, if the parent has SA_NOCLDWAIT set or sets SIGCHLD to SIG_IGN), the resource information for the child process is discarded and not included in the resource information provided by getrusage().

The r_usage argument is a pointer to an object of type struct rusage in which the returned information is stored. The members of rusage are as follows:

- **ru_utime**: The total amount of time spent executing in user mode. Time is given in seconds and microseconds.
- **ru_stime**: The total amount of time spent executing in system mode. Time is given in seconds and microseconds.
- **ru_maxrss**: The maximum resident set size. Size is given in pages (the size of a page, in bytes, is given by the getpagesize(3C) function). See the NOTES section of this page.
- **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 section of this page.
- **ru_majflt**: The number of major faults requiring physical I/O.
- **ru_minflt**: The number of minor faults not requiring physical I/O.
- **ru_nswap**: The number of swaps.
- **ru_inblock**: The number of block input operations.
- **ru_oublock**: The number of block output operations.
- **ru_msgsnd**: The number of messages sent.
- **ru_msgrcv**: The number of messages received.
- **ru_nsignals**: The number of signals received.
- **ru_nvcsw**: The number of voluntary context switches.
- **ru_nivcsw**: The number of involuntary context switches.

The fields are interpreted as follows:

- **ru_utime**: The total amount of time spent executing in user mode. Time is given in seconds and microseconds.
- **ru_stime**: The total amount of time spent executing in system mode. Time is given in seconds and microseconds.
- **ru_maxrss**: The maximum resident set size. Size is given in pages (the size of a page, in bytes, is given by the getpagesize(3C) function). See the NOTES section of this page.
- **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 section of this page.

modified 15 Jan 1997

SunOS 5.6
3C-761
ru_minflt  The number of page faults serviced which did not require any physical
I/O activity. See the 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 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  Upon successful completion, getrusage() returns 0. Otherwise, −1 is returned and errno
   is set to indicate the error.

ERRORS  getrusage() will fail if:
EFAULT  The address specified by the r_usage argument is not in a valid portion
   of the process’ address space.
EINVAL  The who parameter is not a valid value.

SEE ALSO  sar(1M), read(2), times(2), wait(2), write(2), getpagesize(3C), gettimeofday(3C)

NOTES  Only the timeval 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.

3C-762  SunOS 5.6  modified 15 Jan 1997
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);

DESCRIPTION  
The gets() function reads characters from the standard input stream (see intro(3)), stdin, into the array pointed to by s, 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.

The fgets() function reads characters from the stream into the array pointed to by s, until \( n-1 \) characters are read, or a newline character is read and transferred to s, or an end-of-file condition is encountered. The string is then terminated with a null character.

When using gets(), if the length of an input line exceeds the size of s, indeterminate behavior may result. For this reason, it is strongly recommended that gets() be avoided in favor of fgets().

RETURN VALUES  
If end-of-file is encountered and no characters have been read, no characters are transferred to s 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 s is returned.

ERRORS  
The gets() and fgets() functions will fail if data needs to be read and:

EOVERFLOW  
The file is a regular file and an attempt was made to read at or beyond the offset maximum associated with the corresponding stream.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  
iseek(2), read(2), ferror(3S), fopen(3S), fread(3S), getc(3S), scanf(3S), stdio(3S), ungetc(3S), attributes(5)
Network Functions

NAME
getservbyname, getservbyname_r, getservbyport, getservbyport_r, getservent,
getservent_r, setservent, endservent – get service entry

SYNOPSIS
cc [flag ...] file ... -lsoclet -lnsl [ library ... ]
#include <netdb.h>
struct servant *getservbyname(const char *name, const char *proto);
struct servant *getservbyname_r(const char *name, const char *proto,
    struct servant *result, char *buffer, int buflen);
struct servant *getservbyport(int port, const char *proto);
struct servant *getservbyport_r(int port, const char *proto, struct servant *result,
    char *buffer, int buflen);
struct servant *getservent(void);
struct servant *getservent_r(struct servant *result, char *buffer, int buflen);
int setservent(int stayopen);
int endservent(void);

DESCRIPTION
These functions are used to obtain entries for Internet services. An entry may come from
any of the sources for services specified in the /etc/nsswitch.conf file. See
nsswitch.conf(4).

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.

getservbyname() searches for an entry with the Internet service name specified by the
parameter name.

getservbyport() searches for an entry with the Internet port number port.
The string proto is used by both getservbyname() and getservbyport() to restrict the
search to entries with the specified protocol. If proto is NULL, entries with any protocol
may be returned.
The functions setservent(), getservent(), and endservent() are used to enumerate entries
from the services database.

setservent() sets (or resets) the enumeration to the beginning of the set of service entries.
This function should be called before the first call to getservent(). Calls to the functions
getservbyname() and getservbyport() leave the enumeration position in an indeterminate state. If the stayopen flag is non-zero, the system may keep allocated resources
such as open file descriptors until a subsequent call to endservent().

getservent() reads the next line of the file, opening the file if necessary. getservent() opens and rewinds the file. If the stayopen flag is non-zero, the net data base will not be
closed after each call to getservent() (either directly, or indirectly through one of the
other "getserv" calls).

modified 16 May 1997
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>`:

```c
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 */
};
```

3N-766          SunOS 5.6          modified 16 May 1997
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

**ATTRIBUTES**

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>See “Reentrant Interfaces” in DESCRIPTION.</td>
</tr>
</tbody>
</table>

**SEE ALSO**

`intro(2)`, `intro(3)`, `netdir(3N)`, `netconfig(4)`, `nsswitch.conf(4)`, `services(4)`, `attributes(5)`, `netdb(5)`

**WARNINGS**

The reentrant interfaces `getservbyname_r()`, `getservbyport_r()`, and `getservent_r()` are included in this release on an uncommitted basis only, and are subject to change or removal in future minor releases.
NOTES

The functions that return `struct servent` return the least significant 16-bits of the `s_port` field in network byte order. `getservbyport()` and `getservbyport_r()` also expect the input parameter `port` in the network byte order. See `htons(3N)` for more details on converting between host and network byte orders.

Programs that use the interfaces described in this manual page cannot be linked statically since the implementations of these functions employ dynamic loading and linking of shared objects at run time.

In order to ensure that they all return consistent results, `getservbyname()`, `getservbyname_r()`, and `netdir_getbyname()` are implemented in terms of the same internal library function. This function obtains the system-wide source lookup policy based on the `inet` family entries in `netconfig(4)` and the `services: entry in nsswitch.conf(4)`. Similarly, `getservbyport()`, `getservbyport_r()`, and `netdir_getbyaddr()` are implemented in terms of the same internal library function. If the `inet` family entries in `netconfig(4)` have a “-” in the last column for nametoaddr libraries, then the entry for `services` in `nsswitch.conf` will be used; otherwise the nametoaddr libraries in that column will be used, and `nsswitch.conf` will not be consulted.

There is no analogue of `getservent()` and `getservent_r()` in the netdir functions, so these enumeration functions go straight to the `services` entry in `nsswitch.conf`. Thus enumeration may return results from a different source than that used by `getservbyname()`, `getservbyname_r()`, `getservbyport()`, and `getservbyport_r()`.

When compiling multithreaded applications, see `intro(3), Notes On Multithread Applications`, for information about the use of the `_REENTRANT` flag.

Use of the enumeration interfaces `getservent()` and `getservent_r()` is discouraged; enumeration may not be supported for all database sources. The semantics of enumeration are discussed further in `nsswitch.conf(4)`.
NAME
getsockname – get socket name

SYNOPSIS
cc [ flag ... ] file ... -lssocket -lssl [ library ... ]
#include <sys/types.h>
#include <sys/socket.h>

int getsockname(int s, struct sockaddr *name, int *namelen);

DESCRIPTION
getsockname() returns the current name for socket s. The namelen parameter should be
initialized to indicate the amount of space pointed to by name. 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 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.
ENOSR       There were insufficient STREAMS resources available for the operation to
            complete.
ENOTSOCK    The argument s is not a socket.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
bind(3N), getpeername(3N), socket(3N), attributes(5)
**NAME**
getsockname — get the socket name

**SYNOPSIS**
```c
#include <sys/socket.h>

int getsockname(int socket, struct sockaddr *address, size_t *address_len);
```

**DESCRIPTION**
The `getsockname()` function retrieves the locally-bound name of the specified socket, stores this address in the `sockaddr` structure pointed to by the `address` argument, and stores the length of this address in the object pointed to by the `address_len` argument.

If the actual length of the address is greater than the length of the supplied `sockaddr` structure, the stored address will be truncated.

If the socket has not been bound to a local name, the value stored in the object pointed to by `address` is unspecified.

**RETURN VALUES**
Upon successful completion, 0 is returned, the `address` argument points to the address of the socket, and the `address_len` argument points to the length of the address. Otherwise, −1 is returned and `errno` is set to indicate the error.

**ERRORS**
The `getsockname()` function will fail:

- **EBADF** The `socket` argument is not a valid file descriptor.
- **ENOTSOCK** The `socket` argument does not refer to a socket.
- **EOPNOTSUPP** The operation is not supported for this socket’s protocol.

The `getsockname()` function may fail if:

- **EINVAL** The socket has been shut down.
- **ENOBUS** Insufficient resources were available in the system to complete the call.
- **ENOSR** There were insufficient STREAMS resources available for the operation to complete.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**SEE ALSO**
`accept(3XN)`, `bind(3XN)`, `getpeername(3XN)`, `socket(3XN)`, `attributes(5)`, `socket(5)`
NAME       getsockopt, setsockopt – get and set options on sockets

SYNOPSIS   cc [ flag ... ] file ... -lsockets -lnsl [ library ... ]

#include <sys/types.h>
#include <sys/socket.h>

int getsockopt(int s, int level, int optname, char *optval, int *optlen);
int setsockopt(int s, int level, int optname, const char *optval, int optlen);

DESCRIPTION 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.

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, level is
specified as SOL_SOCKET. To manipulate options at any other level, level 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, level is set to the TCP protocol number (see
getprotobyname(3N)).

The parameters optval and optlen 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 getsockopt(), optlen is a value-result parameter, initially containing
the size of the buffer pointed to by optval, and modified on return to indicate the actual
size of the value returned. Use a 0 optval if no option value is to be supplied or returned.

optname 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.

Most socket-level options take an int for optval. For setsockopt(), the optval 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:

   l_onoff    on = 1/off = 0
   l_linger   linger time, in seconds

The following options are recognized at the socket level. Except as noted, each may be
examined with getsockopt() and set with setsockopt().

SO_DEBUG     enable/disable recording of debugging information
SO_REUSEADDR enable/disable local address reuse
SO_KEEPALIVE enable/disable keep connections alive
SO_DONTROUTE enable/disable routing bypass for outgoing messages
SO_LINGER    linger on close if data is present

modified 30 Dec 1996 SunOS 5.6 3N-771
getsockopt (3N)  Network Functions

**SO_BROADCAST** enable/disable permission to transmit broadcast messages
**SO_OOBINLINE** enable/disable reception of out-of-band data in band
**SO_SNDBUF** set buffer size for output
**SO_RCVBUF** set buffer size for input
**SO_DGRAM_ERRIND** application wants delayed error
**SO_TYPE** get the type of the socket (get only)
**SO_ERROR** get and clear error on the socket (get only)

**SO_DEBUG** enables debugging in the underlying protocol modules. **SO_REUSEADDR** indicates that the rules used in validating addresses supplied in a `bind(3N)` call should allow reuse of local addresses. **SO_KEEPALIVE** enables the periodic transmission of messages on a connected socket. If the connected party fails to respond to these messages, the connection is considered broken and processes using the socket are notified using a `SIGPIPE` signal. **SO_DONTROUTE** indicates that outgoing messages should bypass the standard routing facilities. Instead, messages are directed to the appropriate network interface according to the network portion of the destination address.

**SO_LINGER** controls the action taken when unsent messages are queued on a socket and a `close(2)` is performed. If the socket promises reliable delivery of data and **SO_LINGER** is set, the system will block the process on the `close()` attempt until it is able to transmit the data or until it decides it is unable to deliver the information (a timeout period, termed the linger interval, is specified in the `setsockopt()` call when **SO_LINGER** is requested). If **SO_LINGER** is disabled and a `close()` is issued, the system will process the `close()` in a manner that allows the process to continue as quickly as possible.

The option **SO_BROADCAST** requests permission to send broadcast datagrams on the socket. With protocols that support out-of-band data, the **SO_OOBINLINE** option requests that out-of-band data be placed in the normal data input queue as received; it will then be accessible with `recv()` or `read()` calls without the `MSG_OOB` flag.

**SO_SNDBUF** and **SO_RCVBUF** are options that adjust the normal buffer sizes allocated for output and input buffers, respectively. The buffer size may be increased for high-volume connections or may be decreased to limit the possible backlog of incoming data. SunOS sets the maximum buffer size for both UDP and TCP to 256 Kbytes.

By default, delayed errors (such as ICMP port unreachable packets) are returned only for connected datagram sockets. **SO_DGRAM_ERRIND** makes it possible to receive errors for datagram sockets that are not connected. When this option is set, certain delayed errors received after completion of a `sendto()` or `sendmsg()` operation will cause a subsequent `sendto()` or `sendmsg()` operation using the same destination address (`to` parameter) to fail with the appropriate error. See `send(3N)`.

Finally, **SO_TYPE** and **SO_ERROR** are options used only with `getsockopt()`. **SO_TYPE** returns the type of the socket (for example, `SOCK_STREAM`). It is useful for servers that inherit sockets on startup. **SO_ERROR** returns any pending error on the socket and clears the error status. It may be used to check for asynchronous errors on connected datagram sockets or for other asynchronous errors.
RETURN VALUES
If successful, getsockopt() returns 0; otherwise, it returns −1 and sets errno to indicate
the error.

ERRORS
The call succeeds unless:

EBADF       The argument 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.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO close(2), ioctl(2), bind(3N), getprotobynamel(3N), send(3N), socket(3N), attributes(5)
### NAME
getsockopt – get the socket options

### SYNOPSIS
```c
#include <sys/socket.h>

int getsockopt(int socket, int level, int option_name, void *option_value,
               size_t *option_len);
```

### DESCRIPTION
The `getsockopt()` function retrieves the value for the option specified by the `option_name` argument for the socket specified by the `socket` argument. If the size of the option value is greater than `option_len`, the value stored in the object pointed to by the `option_value` argument will be silently truncated. Otherwise, the object pointed to by the `option_len` argument will be modified to indicate the actual length of the value.

The `level` argument specifies the protocol level at which the option resides. To retrieve options at the socket level, specify the `level` argument as `SOL_SOCKET`. To retrieve options at other levels, supply the appropriate protocol number for the protocol controlling the option. For example, to indicate that an option will be interpreted by the TCP (Transport Control Protocol), set `level` to the protocol number of TCP, as defined in the `<netinet/in.h>` header, or as determined by using `getprotobyname(3XN)` function.

The `option_name` argument specifies a single option to be retrieved. It can be one of the following values defined in `<sys/socket.h>`:

- **SO_DEBUG**
  Reports whether debugging information is being recorded. This option stores an `int` value.

- **SO_ACCEPTCONN**
  Reports whether socket listening is enabled. This option stores an `int` value.

- **SO_BROADCAST**
  Reports whether transmission of broadcast messages is supported, if this is supported by the protocol. This option stores an `int` value.

- **SO_REUSEADDR**
  Reports whether the rules used in validating addresses supplied to `bind(3XN)` should allow reuse of local addresses, if this is supported by the protocol. This option stores an `int` value.

- **SO_KEEPALIVE**
  Reports whether connections are kept active with periodic transmission of messages, if this is supported by the protocol.

If the connected socket fails to respond to these messages, the connection is broken and processes writing to that socket are notified with a `SIGPIPE` signal. This option stores an `int` value.

- **SO_LINGER**
  Reports whether the socket lingers on `close(2)` if data is present. If `SO_LINGER` is set, the system blocks the process during `close(2)` until it can transmit the data or until the end of the interval indicated by the `l_linger` member, whichever comes first. If `SO_LINGER` is not specified,
and close(2) is issued, the system handles the call in a way that allows
the process to continue as quickly as possible. This option stores a
linger structure.

SO_OOBINLINE Reports whether the socket leaves received out-of-band data (data
marked urgent) in line. This option stores an int value.

SO_SNDBUF Reports send buffer size information. This option stores an int value.

SO_RCVBUF Reports receive buffer size information. This option stores an int value.

SO_ERROR Reports information about error status and clears it. This option stores
an int value.

SO_TYPE Reports the socket type. This option stores an int value.

For boolean options, 0 indicates that the option is disabled and 1 indicates that the option
is enabled.

Options at other protocol levels vary in format and name.

RETURN VALUES

Upon successful completion, getsockopt() returns 0. Otherwise, −1 is returned and errno
is set to indicate the error.

ERRORS

The getsockopt() function will fail if:

EBADF The socket argument is not a valid file descriptor.

ENOPROTOOPT The option is not supported by the protocol.

ENOTSOCK The socket argument does not refer to a socket.

EINVAL The specified option is invalid at the specified socket level.

EOPNOTSUPP The operation is not supported by the socket protocol.

The getsockopt() function may fail if:

EINVAL The socket has been shut down.

ENOBFS Insufficient resources are available in the system to complete the call.

ENOSR There were insufficient STREAMS resources available for the operation to
complete.

ATTRIBUTES

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

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
bind(3XN), close(2), endprotoent(3XN), setsockopt(3XN), socket(3XN), attributes(5),
socket(5)
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);

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 pass-
word 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, deal-
locate 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 sha-
dow 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(),
  fgetspent_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 \texttt{result} must be a pointer to a \texttt{struct spwd} structure allocated by the caller. On successful completion, the function returns the shadow password entry in this structure. The parameter \texttt{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 \texttt{struct spwd result} point to data stored within this buffer (see \texttt{RETURN VALUES}). The buffer must be large enough to hold all of the data associated with the shadow password entry. The parameter \texttt{buflen} should give the size in bytes of the buffer indicated by \texttt{buffer}.

For enumeration in multithreaded applications, the position within the enumeration is a process-wide property shared by all threads. \texttt{setspent()} may be used in a multithreaded application but resets the enumeration position for all threads. If multiple threads interleave calls to \texttt{getspent_r()}, the threads will enumerate disjoint subsets of the shadow password database.

Like its non-reentrant counterpart, \texttt{getspnam_r()} leaves the enumeration position in an indeterminate state.

\begin{verbatim}
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 */
};
\end{verbatim}

See \texttt{shadow(4)} for more information on the interpretation of this information.

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

The functions \texttt{getspent()}, \texttt{getspent_r()}, \texttt{fgetspent()}, and \texttt{fgetspent()} each return a pointer to a \texttt{struct spwd} if they successfully enumerate an entry; otherwise they return \texttt{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 `fgetspent_r()` is non-NULL, it is always equal to the `result` pointer that was supplied by the caller.

**ERRORS**

The reentrant functions `getspnam_r()`, `getspent_r()`, and `fgetspent_r()` will return NULL and set `errno` to `ERANGE` if the length of the buffer supplied by caller is not large enough to store the result. See `intro(2)` for the proper usage and interpretation of `errno` in multithreaded applications.

**FILES**

/etc/shadow
/etc/nsswitch.conf
/etc/passwd

**ATTRIBUTES**

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>See “Reentrant Interfaces” in DESCRIPTION.</td>
</tr>
</tbody>
</table>

**SEE ALSO**

`nispasswd(1)`, `passwd(1)`, `yppasswd(1)`, `intro(3)`, `getlogin(3C)`, `getpwnam(3C)`, `nsswitch.conf(4)`, `passwd(4)`, `shadow(4)`, `attributes(5)`

**WARNINGS**

The reentrant interfaces `getspnam_r()`, `getspent_r()`, and `fgetspent_r()` are included in this release on an uncommitted basis only, and are subject to change or removal in future minor releases.

**NOTES**

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.
getsubopt(3C) C Library Functions

NAME
getsubopt – parse suboptions from a string

SYNOPSIS
#include <stdlib.h>

int getsubopt(char **optionp, const char *const *tokens, char **valuep);

DESCRIPTION
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:

mount -o rw,hard,bg,wsize=1024 speed:/usr /usr

In this example there are four suboptions: rw, hard, bg, and wsize, the last of which has
an associated value of 1024.

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 optionp contains only one suboption, getsubopt() updates optionp 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 optionp to point to the start of the
next suboption. If the suboption has an associated value, getsubopt() updates valuep to
point to the value's first character. Otherwise it sets valuep to NULL.

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.

When getsubopt() returns, if valuep 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.

Additionally, when getsubopt() fails to match the suboption with the tokens in the tokens
array, the calling program should decide if this is an error, or if the unrecognized option
should be passed to another program.

RETURN VALUES
getsubopt() returns -1 when the token it is scanning is not in the token vector. The vari-
able addressed by valuep contains a pointer to the first character of the token that was not
recognized rather than a pointer to a value for that token.

The variable addressed by optionp points to the next option to be parsed, or a null charac-
ter if there are no more options.

EXAMPLE
The following code fragment shows how to process options to the mount(1M) command
using getsubopt().

#include <stdlib.h>

char *myopts[] = {
    
3C-780 SunOS 5.6 modified 29 Dec 1996
#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) {
            ...
```c
error_no_arg();
errflag++;
} else
    write_size = atoi(value);
break;
case READING:
    /* 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 */
    .
    .
    .
}
```

### ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### SEE ALSO

`mount(1M)`, `getopt(3C)`, `attributes(5)`

### 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  #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);

DESCRIPTION gettext(), dgettext(), and dcgettext() attempt to retrieve a target string based on the
specified msgid 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 msgid argument is a null-terminated string.

NLSPATH is searched first for the location of the LC_MESSAGES catalogue. The setting of
the LC_MESSAGES category of the current locale determines the locale used by gettext() and
dgettext() for string retrieval. category determines the locale used by dcgettext(). If
NLSPATH is not defined and the current locale is "C", gettext(), dgettext(), and dcget-
text() simply return the message string that was passed. In a locale other than "C", if
NLSPATH is not defined or if a message catalogue is not found in any of the components
specified by NLSPATH, the routines search for the message catalogue
dirnamelocaledirectorycategorydomainname.mo, after querying bindtextdomain() for dirname.
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.
For dgettext() and dcgettext(), the domain used is specified by the domainname argument.
The domainname argument is equivalent in syntax and meaning to the domainname
argument to textdomain(), except that the selection of the domain is valid only for the
duration of the dgettext() or dcgettext() call.

textdomain() sets or queries the name of the current domain of the active LC_MESSAGES
locale category. The domainname argument is a null-terminated string that can contain
only the characters allowed in legal filenames.

The domainname argument is the unique name of a domain on the system. If there are
multiple versions of the same domain on one system, namespace collisions can be
avoided by using bindtextdomain(). If textdomain() is not called, a default domain is
selected. The setting of domain made by the last valid call to textdomain() remains valid
across subsequent calls to setlocale(3C), and gettext().
The domainname argument is applied to the currently active LC_MESSAGES locale.
The current setting of the domain can be queried without affecting the current state of the domain by calling `textdomain()` with `domainname` set to the null pointer. Calling `textdomain()` with a `domainname` argument of a null string sets the domain to the default domain (`messages`).

`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 from `bindtextdomain()` is a null-terminated string containing `dirname` or the directory binding associated with `domainname` if `dirname` is NULL. If no binding is found, the default return value is `/usr/lib/locale`. If `domainname` is a null pointer or an empty string, `bindtextdomain()` takes no action and returns a null pointer. The string returned must not be modified by the caller.

### 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()`
location for files containing messages for domain `domainname', language locale, and path predicate `dirname' after a successful call to `bindtextdomain()' for `dgettext()' calls where `LC_XXX' is one of `LC_CTYPE', `LC_NUMERIC', `LC_TIME', `LC_COLLATE', `LC_MONETARY', or `LC_MESSAGES'.

ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe with exceptions</td>
</tr>
</tbody>
</table>

SEE ALSO `msgfmt(1)', `xgettext(1)', `setlocale(3C)', `attributes(5)', `environ(5)'

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
/usr/ucb/cc [ flag ... ] file ...
#include <sys/time.h>
int gettimeofday( tp, tzp )
struct timeval +tp;
struct timezone +tzp;
int settimeofday( tp, tzp )
struct timeval +tp;
struct timezone +tzp;

DESCRIPTION
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.

tp points to a timeval structure, which includes the following members:
    long tv_sec; /* seconds since Jan. 1, 1970 */
    long tv_usec; /* and microseconds */

If tp is a NULL pointer, the current time information is not returned or set.

tzp is an obsolete pointer formerly used to get and set timezone information. tzp is now ignored. Timezone information is now handled using the TZ environment variable; see TIMEZONE(4).

Only the privileged user may set the time of day.

RETURN VALUES
A −1 return value indicates an error occurred; in this case an error code is stored in the global variable errno.

ERRORS
The following error codes may be set in errno:
EINVAL tp specifies an invalid time.
EPERM A user other than the privileged user attempted to set the time.

SEE ALSO
adjtime(2), ctime(3C), gettimeofday(3C), TIMEZONE(4)

NOTES
Use of these interfaces should be restricted to only applications written on BSD platforms. Use of these interfaces with any of the system libraries or in multi-thread applications is unsupported.
tzp is ignored in SunOS 5.X releases.
tv_usec is always 0.
NAME
gettimeofday, settimeofday – get or set the date and time

SYNOPSIS
#include <sys/time.h>
int gettimeofday(struct timeval *tv, void *);  
int settimeofday(struct timeval *tv, void *);

DESCRIPTION
The gettimeofday() function gets and the settimeofday() function sets the system’s
notion of the current time. The current time is expressed in elapsed seconds and
microseconds since 00:00 Universal Coordinated Time, January 1, 1970. The resolution of
the system clock is hardware dependent; the time may be updated continuously or in
clock ticks.

The tv argument points to a timeval structure, which includes the following members:

long tv_sec; /* seconds since Jan. 1, 1970 */
long tv_usec; /* and microseconds */

If tv is a null pointer, the current time information is not returned or set.

The TZ environment variable holds time zone information. See TIMEZONE(4).

The second argument to gettimeofday() and settimeofday() should be a pointer to
NULL.

Only the super-user may set the time of day.

RETURN VALUES
A −1 return value indicates that an error occurred and errno has been set.

ERRORS
The following error codes may be set in errno:

EINVAL       tv specifies an invalid time.
EPERM        A user other than the privileged user attempted to set the time or
time zone.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
adjtime(2), ctime(3C), TIMEZONE(4), attributes(5)

NOTES
If the tv_usec member of tv is > 500000, settimeofday() rounds the seconds upward. If
the time needs to be set with better than one second accuracy, call settimeofday() for the
seconds and then adjtime() for finer accuracy.

modified 21 Apr 1997

SunOS 5.6

3C-787
NAME  
gettxt – retrieve a text string

SYNOPSIS  
#include <nl_types.h>

char *gettxt(const char *msgid, const char *dflt_str);

DESCRIPTION  
gettxt() retrieves a text string from a message file. The arguments to the function are a message identification msgid and a default string dflt_str 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.

The directory locale 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/*.

The user can also change the language in which the messages are displayed by invoking the setlocale() function with the appropriate arguments.

If gettxt() 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 dflt_str points to. A pointer to the second argument is returned if the second argument is not the null string. If dflt_str points to the null string, a pointer to the U.S. English text string "Message not found!!\n" is returned.

The following depicts the acceptable syntax of msgid for a call to gettxt().

    <msgid> = <msgfilename>:<msgnumber>

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 n where n is the number of strings in the file.

On failure to pass the correct msgid or a valid message number to gettxt() a pointer to the text string "Message not found!!\n" is returned.

EXAMPLES  
gettxt("UX:10", "hello world\n")
gettxt("UX:10", ")"

UX is the name of the file that contains the messages. 10 is the message number.
C Library Functions

FILES

/usr/lib/locale/C/LC_MESSAGES/* contains default message files created by mkmsgs
/usr/lib/locale/locale/LC_MESSAGES/* contains message files for different languages created by mkmsgs

ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe with exceptions</td>
</tr>
</tbody>
</table>

SEE ALSO

extr(1), mkmsgs(1), srchtxt(1), gettext(3C), fmtmsg(3C), setlocale(3C), attributes(5), environ(5)

NOTES

It is recommended that gettext(3C) be used in place of this routine.
getusershell (3C)  C Library Functions

NAME
getusershell, setusershell, endusershell – get legal user shells

SYNOPSIS
char *getusershell()
void setusershell()
void endusershell()

DESCRIPTION
getusershell() returns a pointer to a legal user shell as defined by the system manager in the file /etc/shells. If /etc/shells does not exist, a list of the ten locations of the standard system shells: /usr/bin/sh, /usr/bin/csh, /usr/bin/ksh, /usr/bin/jsh, /bin/sh, /bin/csh, /bin/ksh, /bin/jsh, /sbin/sh, /sbin/jsh, are used instead of the file.

getusershell() (opens the file /etc/shells if it exists) returns the next entry in the list of shells.

setusershell() rewinds the file, or the list.

endusershell() closes the file, and frees any memory used by getusershell() and setusershell(). As a side effect, endusershell() rewinds the file /etc/shells.

FILES
/etc/shells
/usr/bin/sh
/usr/bin/csh
/usr/bin/ksh
/usr/bin/jsh
/bin/sh
/bin/csh
/bin/ksh
/bin/jsh
/sbin/sh
/sbin/jsh

RETURN VALUES
getusershell() returns a NULL pointer on EOF.

BUGS
All information is contained in memory that may be freed with a call to endusershell(), so it must be copied if it is to be saved.
NAME
getutent, getutid, getutline, pututline, setutent, endutent, utmpname — access utmp file
entry

SYNOPSIS
#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);

DESCRIPTION
getutent(), getutid(), getutline(), and pututline() each return a pointer to a utmp structure
with the following members:

char ut_user[8]; /* user login name */
char ut_id[4];  /* /sbin/inittab id */
    /* (usually line #) */
char ut_line[12]; /* device name (console, lnxx) */
short ut_pid;   /* process id */
short ut_type;  /* type of entry */
struct exit_status ut_exit; /* exit status of a process */
    /* marked as DEAD_PROCESS */
time_t ut_time;  /* time entry was made */

The structure exit status includes the following members:

short e_termination; /* termination status */
short e_exit;       /* exit status */

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.

getutid() searches forward from the current point in the utmp file until it finds an entry
with a ut_type matching id->ut_type if the type specified is RUN_LVL, BOOT_TIME,
OLD_TIME, or NEW_TIME. If the type specified in id 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
id->ut_id. If the end of file is reached without a match, it fails.

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 line->ut_line string. If the
end of file is reached without a match, it fails.

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
find a matching slot for the new entry, it will add a new entry to the end of the file. It

modified 29 Dec 1996
SunOS 5.6
3C-791
returns a pointer to the \texttt{utmp} structure. When called by a non-root user, \texttt{pututline()} invokes a \texttt{setuid()} root program to verify and write the entry, since \texttt{/etc/utmp} is normally writable only by root. In this event, the \texttt{ut_name} field must correspond to the actual user name associated with the process; the \texttt{ut_type} field must be either \texttt{USER_PROCESS} or \texttt{DEAD_PROCESS}; and the \texttt{ut_line} field must be a device special file and be writable by the user.

\texttt{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.

\texttt{endutent()} closes the currently open file.

\texttt{utmpname()} allows the user to change the name of the file examined, from \texttt{/var/adm/utmp} to any other file. It is most often expected that this other file will be \texttt{/var/adm/wtmp}. If the file does not exist, this will not be apparent until the first attempt to reference the file is made. \texttt{utmpname()} does not open the file. It just closes the old file if it is currently open and saves the new file name.

\textbf{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, \texttt{utmpname()} returns 0. Otherwise, it returns 1.

\textbf{FILES} \texttt{/var/adm/utmp}  
\texttt{/var/adm/wtmp}

\textbf{ATTRIBUTES} See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

\textbf{SEE ALSO} \texttt{getutxent(3C)}, \texttt{ttyslot(3C)}, \texttt{utmp(4)}, attributes(5)

\textbf{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 \texttt{getutid()} or \texttt{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 \texttt{getutline()} to search for multiple occurrences, it would be necessary to zero out the static area after each success, or \texttt{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 \texttt{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 \texttt{getutent()}, \texttt{getutid()} or \texttt{getutline()} routines, if the user has just modified those contents and passed the pointer back to \texttt{pututline()}.

These routines use buffered standard I/O for input, but \texttt{pututline()} uses an unbuffered non-standard write to avoid race conditions between processes trying to modify the \texttt{utmp} and \texttt{wtmp}.
NAME
getutxent, getutxid, getutxline, pututxline, setutxent, endutxent, utmpxname, getutmp,
getutmpx, updwtmp, updwtmpx – access utmpx file entry

SYNOPSIS
#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);

DESCRIPTION
getutxent(), getutxid(), and getutxline() each return a pointer to a utmpx structure with
the following members:

- char ut_user[32]; /* user login name */
- char ut_id[4]; /* /etc/inittab id */
- char ut_line[32]; /* device name (console, lnxx) */
- pid_t ut_pid; /* process id */
- short ut_type; /* type of entry */
- struct exit_status ut_exit; /* exit status of a process */
- struct timeval ut_tv; /* time entry was made */
- long ut_session; /* session ID, used for windowing */
- long pad[5]; /* reserved for future use */
- short ut_syslen; /* significant length of ut_host */
- char ut_host[257]; /* host name, if remote */

The structure exit status includes the following members:

- short e_termination; /* termination status */
- short e_exit; /* exit status */

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.

getutxid() Searches forward from the current point in the utmpx file until it finds an entry with a
ut_type matching id→ut_type if the type specified is RUN_LVL, BOOT_TIME, OLD_TIME,
or NEW_TIME. If the type specified in id is INIT_PROCESS, LOGIN_PROCESS,
USER_PROCESS, or DEAD_PROCESS, then getutxid() will return a pointer to the first
getutxent(3C)  C Library Functions

entry whose type is one of these four and whose ut_id field matches id\(\rightarrow\)ut_id. If the end of file is reached without a match, it fails.

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\(\rightarrow\)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.

dendutxent()  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. (See getutent(3C) for utmp structure)

getutmpx()  Copies the information stored in the fields of the utmp structure to the corresponding fields of the utmpx structure. (See getutent(3C) for utmp 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 \textit{wfilex} and its parallel file, whose name is obtained by truncating the final “\textbackslash{}x” from \textit{wfilex}. If only one of them exists, the second one is created and initialized to reflect the state of the existing file. \textit{utmpx} is written to \textit{wfilex}, and the corresponding \textit{utmp} structure is written to the parallel file.

RETURN VALUES A null pointer is returned upon failure to read, whether for permissions or having reached the end of file, or upon failure to write.

FILES
\begin{itemize}
\item /var/adm/utmp contains current user access and administrative information (old format)
\item /var/adm/utmpx contains current user access and administration information (new format)
\item /var/adm/wtmp contains a history of user access and administrative information.
\item /var/adm/wtmpx contains a history of user access and administrative information.
\end{itemize}

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

\begin{center}
\begin{tabular}{|l|l|}
\hline
ATTRIBUTE TYPE & ATTRIBUTE VALUE \\
\hline
MT-Level & Unsafe \\
\hline
\end{tabular}
\end{center}

SEE ALSO getutent(3C), ttyslot(3C), utmp(4), utmpx(4), attributes(5)

NOTES The most current entry is saved in a static structure. Multiple accesses require that it be copied before further accesses are made. On each call to either \texttt{getutxid()} or \texttt{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 \texttt{getutxline()} to search for multiple occurrences it would be necessary to zero out the static after each success, or \texttt{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 \texttt{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 \texttt{getutxent()}, \texttt{getutxid()}, or \texttt{getutxline()} routines, if the user has just modified those contents and passed the pointer back to \texttt{pututxline()}.

These routines use buffered standard I/O for input, but \texttt{pututxline()} uses an unbuffered write to avoid race conditions between processes trying to modify the \texttt{utmpx} and \texttt{wtmpx} files.
NAME  getvfsent, getvfsfile, getvfsspec, getvfsany – get vfstab file entry

SYNOPSIS  
#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, struct vfstab *vref);

DESCRIPTION  The getvfsent(), getvfsfile(), getvfsspec(), and getvfsany() functions each fill in the structure pointed to by vp with the broken-out fields of a line in the /etc/vfstab file. Each line in the file contains a vfstab structure, declared in the <sys/vfstab.h> header:

    char *vfs_special;
    char *vfs_fsckdev;
    char *vfs_mountp;
    char *vfs_fstype;
    char *vfs_fsckpass;
    char *vfs_automnt;
    char *vfs_mntopts;

The fields have meanings described in vfstab(4).

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 fp until a mount point matching file is found and fills vp with the fields from the line in the file. getvfsspec() searches the file referenced by fp until a special device matching spec is found and fills vp with the fields from the line in the file. spec 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 fp until a match is found between a line in the file and vref. vref matches the line if all non-null entries in vref match the corresponding fields in the file.

Note that these routines do not open, close, or rewind the file.

RETURN VALUES  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:

VFS_TOOLONG  A line in the file exceeded the internal buffer size of VFS_LINE_MAX.
VFS_TOOMANY  A line in the file contains too many fields.
VFS_TOOFEW   A line in the file contains too few fields.
FILES
/etc/vfstab

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
vfstab(4), attributes(5)

NOTES
The members of the vfstab structure point to information contained in a static area, so it must be copied if it is to be saved.
NAME  getwc – get wide character from a stream

SYNOPSIS  #include <stdio.h>
# include <wchar.h>

wint_t getwc(FILE *stream);

DESCRIPTION  The getwc() function is equivalent to fgetwc(3S), except that if it is implemented as a macro it may evaluate stream more than once, so the argument should never be an expression with side effects.

RETURN VALUES  Refer to fgetwc(3S).

ERRORS  Refer to fgetwc(3S).

USAGE  This interface is provided in order to align with some current implementations, and with possible future ISO standards.

Because it may be implemented as a macro, getwc() may treat incorrectly a stream argument with side effects. In particular, getwc(*f++) may not work as expected. Therefore, use of this function is not recommended; fgetwc(3S) should be used instead.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  fgetwc(3S), attributes(5)
NAME  
get_wch, wget_wch, mvget_wch, mvwget_wch – get a wide character from terminal

SYNOPSIS
#include <curses.h>

int get_wch(wint_t *ch);
int wget_wch (WINDOW *win, wint_t *ch);
int mvget_wch(int y, int x, wint_t *ch);
int mvwget_wch(WINDOW *win, int y, int x, wint_t *ch);

ARGUMENTS

ch   Is a pointer to a wide integer where the returned wide character or KEY_value can be stored.

win  Is a pointer to the window associated with the terminal from which the character is to be read.

y    Is the y (row) coordinate for the position of the character to be read.

x    Is the x (column) coordinate for the position of the character to be read.

DESCRIPTION

The get_wch() and wget_wch() functions get a wide character from the terminal associated with the window stdscr or window win, respectively. The mvget_wch() and mvwget_wch() functions move the cursor to the position specified in stdscr or win, respectively, then get a character.

If the window is not a pad and has been changed since the last call to refresh(3XC), get_wch() calls refresh() to update the window before the next character is read.

The setting of certain functions affects the behavior of the get_wch() set of functions. For example, if cbreak(3XC) is set, characters typed by the user are immediately processed. If halfdelay(3XC) is set, get_wch() waits until a character is typed or returns ERR if no character is typed within the specified timeout period. This timeout can also be specified for individual windows with the delay parameter of timeout(3XC). A negative value waits for input; a value of 0 returns ERR if no input is ready; a positive value blocks until input arrives or the time specified expires (in which case ERR is returned). If nodelay(3XC) is set, ERR is returned if no input is waiting; if not set, get_wch() waits until input arrives. Each character will be echoed to the window unless noecho(3XC) has been set.

If keypad handling is enabled (keypad(3XC) is TRUE), the token for the function key (a KEY_value) is stored in the object pointed to by ch and KEY_CODE_YES is returned. If a character is received that could be the beginning of a function key (for example, ESC), an inter-byte timer is set. If the remainder of the sequence is not received before the time expires, the character is passed through; otherwise, the value of the function key is returned. If notimeout() is set, the inter-byte timer is not used.

The ESC key is typically a prefix key used with function keys and should not be used as a single character.

See the getch(3XC) manual page for a list of tokens for function keys that are returned by the get_wch() set of functions if keypad handling is enabled (Some terminals may not support all tokens).
RETURN VALUES

When these functions successfully report the pressing of a function key, they return \texttt{KEY\_CODE\_YES}. When they successfully report a wide character, they return \texttt{OK}. Otherwise, they return \texttt{ERR}.

ERRORS

None.

SEE ALSO

\texttt{cbreak(3XC)}, \texttt{echo(3XC)}, \texttt{halfdelay(3XC)}, \texttt{keypad(3XC)}, \texttt{nodelay(3XC)}, \texttt{notimeout(3XC)}, \texttt{raw(3XC)}, \texttt{timeout(3XC)}
NAME
getwchar – get wide character from stdin stream

SYNOPSIS
#include <wchar.h>

wint_t getwchar(void);

DESCRIPTION
The getwchar() function is equivalent to getwc(stdin).

RETURN VALUES
Refer to fgetwc(3).

ERRORS
Refer to fgetwc(3).

USAGE
If the wint_t value returned by getwchar() is stored into a variable of type wchar_t and then compared against the wint_t macro WEOF, the comparison may never succeed, because wchar_t is defined as unsigned.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
fgetwc(3), getwc(3), attributes(5)
## NAME
getwd – get current working directory pathname

## SYNOPSIS
```c
#include <unistd.h>

char *getwd(char *path_name);
```

## DESCRIPTION
The `getwd()` function determines an absolute pathname of the current working directory of the calling process, and copies that pathname into the array pointed to by the `path_name` argument.

If the length of the pathname of the current working directory is greater than (`PATH_MAX` + 1) including the null byte, `getwd()` fails and returns a null pointer.

## RETURN VALUES
Upon successful completion, a pointer to the string containing the absolute pathname of the current working directory is returned. Otherwise, `getwd()` returns a null pointer and the contents of the array pointed to by `path_name` are undefined.

## ERRORS
No errors are defined.

## USAGE
For portability to implementations conforming to versions of the X/Open Portability Guide prior to XPG4v2, `getcwd(3C)` is preferred over this function.

## SEE ALSO
`getcwd(3C)`, `standards(5)`
NAME
getwidth — get codeset information

SYNOPSIS
#include <euc.h>
#include <getwidth.h>
void getwidth(eucwidth_t *ptr);

DESCRIPTION
The getwidth() function reads the character class table for the current locale to get information on the supplementary codesets. getwidth() sets this information into the struct eucwidth_t. This struct is defined in <euc.h> and has the following members:

short int _eucw1,_eucw2,_eucw3;
short int _scrw1,_scrw2,_scrw3;
short int _pcw;
char _multibyte;

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.

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.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe with exceptions</td>
</tr>
</tbody>
</table>

SEE ALSO
euclen(3C), setlocale(3C), attributes(5)

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

This function will only work with EUC locales.

modified 20 Dec 1996

SunOS 5.6

3C-803
NAME  getwin, putwin – read a window from, and write a window to, a file

SYNOPSIS  
#include <curses.h>

WINDOW *getwin(FILE *filep);
int putwin(WINDOW *win, FILE *filep);

ARGUMENTS  
filep  Is a pointer to a stdio stream.
win  Is a pointer to a window.

DESCRIPTION  
The getwin() function reads window-related data (written earlier by putwin()) from the stdio stream pointed to by filep. It then creates and initializes a new window using that data.

The putwin() function writes all the data associated with the window pointed to by win to the stdio stream pointed to by filep. The getwin() function can later retrieve this data.

RETURN VALUES  
On success, the getwin() function returns a pointer to the new window created. Otherwise, it returns a null pointer.

On success, the putwin() function returns OK. Otherwise, it returns ERR.

ERRORS  
None.

SEE ALSO  scr_dump(3XC)
NAME  
getws, fgetws – convert a string of EUC characters from the stream to Process Code

SYNOPSIS  
#include <stdio.h>
#include <widec.h>
wchar_t *getws(wchar_t *s);
wchar_t *fgetws(wchar_t *s, int n, FILE *stream);

DESCRIPTION  
The getws() function 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 s, 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 wchar_t NULL character. The getws() function returns its argument.

The fgetws() function reads EUC characters from the stream, converts them to Process Code, and writes them to the array pointed to by s. It stops when either n−1 characters are read, a new-line character is read and transferred to s, or an end-of-file condition is encountered. The string is then terminated with a wchar_t NULL character. The fgetws() function returns its first argument.

RETURN VALUES  
If end-of-file is encountered and no characters have been read, no characters are transferred to s 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 s is returned.

ERRORS  
The fgetws() function will fail if data needs to be read and:
EOVERFLOW  The file is a regular file and an attempt was made to read at or beyond the offset maximum associated with the corresponding stream.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  
ferror(3S), fread(3S), getwc(3S), putws(3S), scanf(3S), attributes(5)
NAME  glob, globfree – generate path names matching a pattern

SYNOPSIS  
#include <glob.h>

int glob(const char *pattern, int flags, int(*errfunc)(const char *epath, int eerrno),
  glob_t *pglob);

void globfree(glob_t *pglob);

DESCRIPTION  The glob() function is a path name generator. The globfree() function frees any memory allocated by glob() associated with pglob.

pattern Argument  The argument pattern is a pointer to a path name pattern to be expanded. The glob() function matches all accessible path names against this pattern and develops a list of all path names that match. In order to have access to a path name, glob() requires search permission on every component of a path except the last, and read permission on each directory of any filename component of pattern that contains any of the following special characters:

*  ?  [  

pglob Argument  The structure type glob_t is defined in the header <glob.h> and includes at least the following members:

size_t gl_pathc  Count of paths matched by pattern.

char **gl_pathv  Pointer to a list of matched path names.

size_t gl_offs  Slots to reserve at the beginning of gl_pathv.

The glob() function stores the number of matched path names into pglob->gl_pathc and a pointer to a list of pointers to path names into pglob->gl_pathv. The path names are in sort order as defined by the current setting of the LC_COLLATE category. The first pointer after the last path name is a NULL pointer. If the pattern does not match any path names, the returned number of matched paths is set to zero, and the contents of pglob->gl_pathv are implementation-dependent.

It is the caller’s responsibility to create the structure pointed to by pglob. The glob() function allocates other space as needed, including the memory pointed to by gl_pathv. The globfree() function frees any space associated with pglob from a previous call to glob().

flags Argument  The flags argument is used to control the behavior of glob(). The value of flags is a bitwise inclusive OR of zero or more of the following constants, which are defined in the header <glob.h>:

GLOB_APPEND  Append path names generated to the ones from a previous call to glob().

GLOB_DOFFS  Make use of pglob->gl_offs. If this flag is set, pglob->gl_offs is used to specify how many NULL pointers to add to the beginning of pglob->gl_pathv. In other words, pglob->gl_pathv will point to pglob->gl_offs NULL pointers, followed by pglob->gl_pathc path name pointers, followed by a NULL pointer.

3C-806 SunOS 5.6 modified 29 Dec 1996
GLOB_ERR Causes glob() to return when it encounters a directory that it cannot open or read. Ordinarily, glob() continues to find matches.

GLOB_MARK Each path name that is a directory that matches pattern has a slash appended.

GLOB_NOCHECK If pattern does not match any path name, then glob() returns a list consisting of only pattern, and the number of matched path names is 1.

GLOB_NOESCAPE Disable backslash escaping.

GLOB_NOSORT Ordinarily, glob() sorts the matching path names according to the current setting of the LC_COLLATE category. When this flag is used the order of path names returned is unspecified.

The GLOB_APPEND flag can be used to append a new set of path names to those found in a previous call to glob(). The following rules apply when two or more calls to glob() are made with the same value of pglob and without intervening calls to globfree():

1. The first such call must not set GLOB_APPEND. All subsequent calls must set it.
2. All the calls must set GLOB_DOOFFS, or all must not set it.
3. After the second call, pglob->gl_pathv points to a list containing the following:
   a. Zero or more NULL pointers, as specified by GLOB_DOOFFS and pglob->gl_offs.
   b. Pointers to the path names that were in the pglob->gl_pathv list before the call, in the same order as before.
   c. Pointers to the new path names generated by the second call, in the specified order.
4. The count returned in pglob->gl_pathc will be the total number of path names from the two calls.
5. The application can change any of the fields after a call to glob(). If it does, it must reset them to the original value before a subsequent call, using the same pglob value, to globfree() or glob() with the GLOB_APPEND flag.

errfunc and epath Arguments If, during the search, a directory is encountered that cannot be opened or read and errfunc is not a NULL pointer, glob() calls (*errfunc) with two arguments:

1. The epath argument is a pointer to the path that failed.
2. The errno argument is the value of errno from the failure, as set by the opendir(3C), readdir(3C) or stat(2) functions. (Other values may be used to report other errors not explicitly documented for those functions.)

The following constants are defined as error return values for glob():

GLOB_ABORTED The scan was stopped because GLOB_ERR was set or (*errfunc) returned non-zero.

GLOB_NOMATCH The pattern does not match any existing path name, and GLOB_NOCHECK was not set in flags.

modified 29 Dec 1996 SunOS 5.6 3C-807
GLOB_NOSPACE   An attempt to allocate memory failed.

If (*errfunc) is called and returns non-zero, or if the GLOB_ERR flag is set in flags, glob() stops the scan and returns GLOB_ABORTED after setting gl_pathc and gl_pathv in pglob to reflect the paths already scanned. If GLOB_ERR is not set and either errfunc is a NULL pointer or (*errfunc) returns zero, the error is ignored.

RETURN VALUES   The following values are returned by glob():

0   successful completion. The argument pglob->gl_pathc returns the number of matched path names and the argument pglob->gl_pathv contains a pointer to a null-terminated list of matched and sorted path names. However, if pglob->gl_pathc is zero, the content of pglob->gl_pathv is undefined.

non-zero   an error has occurred. Non-zero constants are defined in <glob.h>. The arguments pglob->gl_pathc and pglob->gl_pathv are still set as defined above.

The globfree() function returns no value.

USAGE   This function is not provided for the purpose of enabling utilities to perform path name expansion on their arguments, as this operation is performed by the shell, and utilities are explicitly not expected to redo this. Instead, it is provided for applications that need to do path name expansion on strings obtained from other sources, such as a pattern typed by a user or read from a file.

If a utility needs to see if a path name matches a given pattern, it can use fnmatch(3C).

Note that gl_pathc and gl_pathv have meaning even if glob() fails. This allows glob() to report partial results in the event of an error. However, if gl_pathc is zero, gl_pathv is unspecified even if glob() did not return an error.

The GLOB_NOCHECK option could be used when an application wants to expand a path name if wildcards are specified, but wants to treat the pattern as just a string otherwise.

The new path names generated by a subsequent call with GLOB_APPEND are not sorted together with the previous path names. This mirrors the way that the shell handles path name expansion when multiple expansions are done on a command line.

Applications that need tilde and parameter expansion should use the wordexp() function.

EXAMPLES   One use of the GLOB_DOOFFS flag is by applications that build an argument list for use with the execv(2), execve() or execvp() functions. Suppose, for example, that an application wants to do the equivalent of:

ls -l *.c

but for some reason:

system("ls -l *.c")
is not acceptable. The application could obtain approximately the same result using the sequence:

```c
globbuf.gl_offs = 2;
glob ("*.c", GLOB_DOFFS, NULL, &globbuf);
globbuf.gl_pathv[0] = "ls";
globbuf.gl_pathv[1] = "-l";
execvp ("ls", &globbuf.gl_pathv[0]);
```

Using the same example:

```bash
ls -l *.c *.h
```

could be approximately simulated using GLOB_APPEND as follows:

```c
globbuf.gl_offs = 2;
glob ("*.c", GLOB_DOFFS, NULL, &globbuf);
glob ("*.h", GLOB_DOFFS | GLOB_APPEND, NULL, &globbuf);
...}
```

**ATTRIBUTES**

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**SEE ALSO**

execv(2), stat(2), fnmatch(3C), opendir(3C), readdir(3C), wordexp(3C), attributes(5)
global_variables

NAME

global_variables – variables used for X/Open Curses

DESCRIPTION

The global variables defined for X/Open Curses are as follows:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLORS</td>
<td>Number of colors supported by terminal</td>
</tr>
<tr>
<td>COLOR_PAIRS</td>
<td>Number of color pairs supported by terminal</td>
</tr>
<tr>
<td>COLS</td>
<td>Number of columns supported by terminal</td>
</tr>
<tr>
<td>LINES</td>
<td>Number of lines supported by terminal</td>
</tr>
<tr>
<td>boolcodes[]</td>
<td>termcap capability names</td>
</tr>
<tr>
<td>boolfnames[]</td>
<td>Full C names</td>
</tr>
<tr>
<td>boolnames[]</td>
<td>terminfo capability names</td>
</tr>
<tr>
<td>cur_term</td>
<td>Current terminal</td>
</tr>
<tr>
<td>curscr</td>
<td>Current screen image</td>
</tr>
<tr>
<td>numcodes[]</td>
<td>termcap capability codes</td>
</tr>
<tr>
<td>numfnames[]</td>
<td>Full C names</td>
</tr>
<tr>
<td>numfnames[]</td>
<td>terminfo capability codes</td>
</tr>
<tr>
<td>stdscr</td>
<td>Standard screen supplied by initscr()</td>
</tr>
<tr>
<td>strcodes[]</td>
<td>termcap capability name</td>
</tr>
<tr>
<td>strfnames[]</td>
<td>Full C names</td>
</tr>
<tr>
<td>strnames[]</td>
<td>terminfo capability names</td>
</tr>
<tr>
<td>ttytype</td>
<td>Terminal type</td>
</tr>
</tbody>
</table>

The boolcodes[], boolfnames[], boolnames[], numcodes[], numfnames[], strcodes[], strfnames[], strnames[], and ttytype constants conform to UNIX System V.

The curscr, stdscr, cur_term, COLS, LINES, COLORS, and COLOR_PAIRS, constants conform to UNIX System V and XPG4 version 2.
NAME
gmatch – shell global pattern matching

SYNOPSIS
c{[flag ...] file ... -lgen [library ...]}
#include <libgen.h>
int gmatch(const char *str, const char *pattern);

DESCRIPTION
gmatch() checks whether the null-terminated string str matches the null-terminated pattern string pattern. See the sh(1) section File Name Generation for a discussion of pattern matching. A backslash (\) is used as an escape character in pattern strings.

RETURN VALUES
gmatch() returns non-zero if the pattern matches the string, zero if the pattern does not.

EXAMPLE
In the following example, gmatch() returns non-zero (true) for all strings with “a” or “-” as their last character.

char *s;
gmatch (s, "*[a-]" )

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
sh(1), attributes(5)

NOTES
When compiling multi-thread applications, the _REENTRANT flag must be defined on the compile line. This flag should only be used in multi-thread applications.

modified 29 Dec 1996
NAME    grantpt – grant access to the slave pseudo-terminal device

SYNOPSIS  

```c
#include <stdlib.h>

int grantpt(int fildes);
```

DESCRIPTION  The `grantpt()` function changes the mode and ownership of the slave pseudo-terminal device associated with its master pseudo-terminal counter part. `fildes` is the file descriptor returned from a successful open of the master pseudo-terminal device. A `setuid` root program (see `setuid(2)`) is invoked to change the permissions. The user ID of the slave is set to the real UID of the calling process and the group ID is set to a reserved group. The permission mode of the slave pseudo-terminal is set to readable and writable by the owner and writable by the group.

RETURN VALUES  Upon successful completion, `grantpt()` returns 0. Otherwise, it returns -1 and sets `errno` to indicate the error.

ERRORS  The `grantpt()` function may fail if:

- **EBADF**  The `fildes` argument is not a valid open file descriptor.
- **EINVAL**  The `fildes` argument is not associated with a master pseudo-terminal device.
- **EACCES**  The corresponding slave pseudo-terminal device could not be accessed.

USAGE  The `grantpt()` function will fail if it is unable to successfully invoke the `setuid` root program. It may also fail if the application has installed a signal handler to catch `SIGCHLD` signals.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  `open(2)`, `setuid(2)`, `ptsname(3C)`, `unlockpt(3C)`, `attributes(5)`

STREAMS Programming Guide
<table>
<thead>
<tr>
<th>NAME</th>
<th>halfdelay – enable/disable half-delay mode</th>
</tr>
</thead>
</table>
| SYNOPSIS | `#include <curses.h>`
`int halfdelay(int tenths);`
| ARGUMENTS | `tenths` Is the number of tenths of seconds for which to block input (1 to 255).
| DESCRIPTION | The `halfdelay()` function is similar to `cbreak(3XC)` in that when set, characters typed by the user are immediately processed by the program. The difference is that `ERR` is returned if no input is received after `tenths` tenths seconds.
The `nocbreak(3XC)` function should be used to leave half-delay mode.
| RETURN VALUES | On success, the `halfdelay()` function returns `OK`. Otherwise, it returns `ERR`.
| ERRORS | None.
| SEE ALSO | `cbreak(3XC)` |
### NAME
has_ic, has_il – determine insert/delete character/line capability

### SYNOPSIS
```c
#include <curses.h>

bool has_ic(void);
bool has_il(void);
```

### DESCRIPTION
The `has_ic()` function determines whether or not the terminal has insert/delete character capability.
The `has_il()` function determines whether or not the terminal has insert/delete line capability.

### RETURN VALUES
The `has_ic()` function returns `TRUE` if the terminal has insert/delete character capability and `FALSE` otherwise.
The `has_il()` function returns `TRUE` if the terminal has insert/delete line capability and `FALSE` otherwise.

### ERRORS
None.
NAME
hline, mvhline, mvvline, mvwhline, mvwvline, vline, whline, wvline – use single-byte characters (and renditions) to draw lines

SYNOPSIS
#include <curses.h>

int hline(chtype ch, int n);
int mvhline(int y, int x, chtype ch, int n);
int mvvline(int y, int x, chtype ch, int n);
int mvwhline(WINDOW *win, int y, int x, chtype ch,
int n);
int mvwvline(WINDOW *win, int y, int x, chtype ch,
int n);
int vline(chtype ch, int n);
int whline(WINDOW *win, chtype ch, int n);
int wvline(WINDOW *win, chtype ch, int n);

ARGUMENTS
ch Is the character used to draw the line.
n Is the maximum number of characters in the line.
y Is the y (row) coordinate for the start of the line.
x Is the x (column) coordinate for the start of the line.
win Is a pointer to a window.

DESCRIPTION
The hline(), vline(), whline(), wvline() functions draw a horizontal or vertical line, in either the window stdscr or win starting at the current cursor position. The line is drawn using the character ch and is a maximum of n positions long, or as many as will fit into the window. If ch is 0 (zero), the default horizontal or vertical character is used.

The mvhline(), mvvline(), mvwhline(), mvwvline() functions are similar to the previous group of functions but the line begins at cursor position specified by x and y.

The functions with names ending with hline() draw horizontal lines proceeding towards the last column of the same line. The functions with names ending with vline() draw vertical lines proceeding towards the last column of the same line.

These functions do not change the position of the cursor.

RETURN VALUES
On success, these functions return OK. Otherwise, they return ERR.

ERRORS
None

SEE ALSO
border(3XC), border_set(3XC), hline_set(3XC)

modified 1 Jun 1996

SunOS 5.6

3XC-815
NAME  
hline_set, mvhline_set, mvvline_set, mvwhline_set, mvwvline_set, vline_set, whline_set,
wvline_set – use complex characters (and renditions) to draw lines

SYNOPSIS  
#include <curses.h>

int hline_set(const cchar_t *ch, int n);
int mvhline_set(int y, int x, const cchar_t *wch, int n);
int mvvline_set(int y, int x, const cchar_t *wch, int n);
int mvwhline_set(WINDOW *win, int y, int x, const cchar_t *wch, int n);
int mvwvline_set(WINDOW *win, int y, int x, const cchar_t *wch, int n);
int vline_set(const cchar_t *wch, int n);
int whline_set(WINDOW *win, const cchar_t *wch, int n);
int wvline_set(WINDOW *win, const cchar_t *wch, int n);

ARGUMENTS  
wch     Is the complex character used to draw the line.
n     Is the maximum number of characters in the line.
y     Is the y (row) coordinate for the start of the line.
x     Is the x (column) coordinate for the start of the line.
win     Is a pointer to a window.

DESCRIPTION  
The hline_set(), vline_set(), whline_set(), wvline_set() functions draw a line, in either
the window stdscr or win starting at the current cursor position. The line is drawn using
the character wch and is a maximum of n positions long, or as many as will fit into the
window. If wch is a null pointer, the default horizontal or vertical character is used.
The mvhline_set(), mvvline_set(), mvwhline_set(), mvwvline_set() functions are similar
to the previous group of functions but the line begins at cursor position specified by x
and y.
The functions with names ending with hline_set() draw horizontal lines proceeding
towards the last column of the same line. The functions with names ending with
vline_set() draw vertical lines proceeding towards the last column of the same line.
These functions do not change the position of the cursor.

RETURN VALUES  
On success, these functions return OK. Otherwise, they return ERR.

ERRORS  
None.

SEE ALSO  
border(3XC), border_set(3XC), hline(3XC)
NAME
hsearch, hcreate, hdestroy – manage hash search tables

SYNOPSIS
#include <search.h>
ENTRY ∗hsearch(ENTRY item, ACTION action);
int hcreate (size_t nelments);
void hdestroy(void);

DESCRIPTION
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)). item is a structure of type ENTRY (defined in the <search.h> header) containing two pointers: item.key points to the comparison key, and item.data points to any other data to be associated with that key. (Pointers to types other than void should be cast to pointer-to-void.) action 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.

hcreate() allocates sufficient space for the table, and must be called before hsearch() is used. nel 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.

hdestroy() destroys the search table, and may be followed by another call to hcreate().

RETURN VALUES
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.

hcreate() returns zero if it cannot allocate sufficient space for the table.

EXAMPLE
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.

#include <stdio.h>
#include <search.h>
#include <string.h>
#include <stdlib.h>
struct info {
    /* this is the info stored in table */
    int age, room; /* other than the key */
};
define NUM_EMPL 5000 /* # of elements in search table */
main()
{
    /* space to store strings */

modified 29 Dec 1996
SunOS 5.6
3C-817
char string_space[NUM_EMPL+20]; /* space to store employee info */
struct info info_space[NUM_EMPL]; /* next avail space in string_space */
char *str_ptr = string_space; /* next avail space in info_space */
struct info *info_ptr = info_space;
ENTRY item, *found_item; /* name to look for in table */
char name_to_find[30];
int i = 0;

/* create table */
(void) hcreate(NUM_EMPL);
while (scanf("%s%d%d", str_ptr, &info_ptr->age,
&info_ptr->room) != EOF && i++ < NUM_EMPL) {
    /* put info in structure, and structure in item */
    item.key = str_ptr;
    item.data = (void *)info_ptr;
    str_ptr += strlen(str_ptr) + 1;
    info_ptr++;
    /* put item into table */
    (void) hsearch(item, ENTER);
}

/* access table */
item.key = name_to_find;
while (scanf("%s", item.key) != EOF) {
    if ((found_item = hsearch(item, FIND)) != NULL) {
        /* if item is in the table */
        (void)printf("found %s, age = %d, room = %d\n",
        found_item->key,
        ((struct info *)found_item->data)->age,
        ((struct info *)found_item->data)->room);
    } else {
        (void)printf("no such employee %s\n", name_to_find)
    }
}
return 0;

ATTRIBUTES See attributes(5) for descriptions of the following attributes:
## SEE ALSO

bsearch(3C), lsearch(3C), malloc(3C), string(3C), tsearch(3C), malloc(3X), attributes(5)


## NOTES

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

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

---

**ATTRIBUTES**

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>
htonl (3XN) X/Open Networking Services Library Functions

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

SYNOPSIS
cc [-f ... ] file ... -lxnet [ library ... ]
#include <arpa/inet.h>
in_addr_t htonl(in_addr_t hostlong);
in_port_t htons(in_port_t hostshort);
in_addr_t ntohl(in_addr_t netlong);
in_port_t ntohs(in_port_t netshort);

DESCRIPTION
These functions convert 16-bit and 32-bit quantities between network byte order and host byte order.

RETURN VALUES
The htonl() and htons() functions return the argument value converted from host to network byte order.
The ntohl() and ntohs() functions return the argument value converted from network to host byte order.

ERRORS
No errors are defined.

USAGE
These functions are most often used in conjunction with Internet addresses and ports as returned by gethostent(3XN) and getservent(3XN).
On some architectures these functions are defined as macros that expand to the value of their argument.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
endhostent(3XN),endservent(3XN), attributes(5), inet(5)
NAME
hypot – Euclidean distance function

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

DESCRIPTION
The hypot() function computes the length of the hypotenuse of a right-angled triangle:
\[ \sqrt{x^2 + y^2} \]

RETURN VALUES
Upon successful completion, hypot() returns the length of the hypotenuse of a right angled triangle with sides of length \( x \) and \( y \).
If the result would cause overflow, HUGE_VAL is returned and errno may be set to ERANGE.
If \( x \) or \( y \) is NaN, NaN is returned.

ERRORS
The hypot() function may fail if:
ERANGE  The result overflows.

USAGE
The hypot() function takes precautions against underflow and overflow during intermediate steps of the computation.
An application wishing to check for error situations should set errno to 0 before calling hypot(). If errno is non-zero on return, or the return value is HUGE_VAL or NaN, an error has occurred.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
isnan(3M), sqrt(3M), attributes(5)
NAME
iconv – code conversion function

SYNOPSIS
#include <iconv.h>

size_t iconv(iconv_t cd, const char **inbuf, size_t *inbytesleft, char **outbuf,
size_t *outbytesleft);

DESCRIPTION
The iconv() function converts the sequence of characters from one code set, in the array specified by inbuf, into a sequence of corresponding characters in another code set, in the array specified by outbuf. The code sets are those specified in the iconv_open() call that returned the conversion descriptor, cd. The inbuf argument points to a variable that points to the first character in the input buffer and inbytesleft indicates the number of bytes to the end of the buffer to be converted. The outbuf argument points to a variable that points to the first available byte in the output buffer and outbytesleft indicates the number of the available bytes to the end of the buffer.

For state-dependent encodings, the conversion descriptor cd is placed into its initial shift state by a call for which inbuf is a null pointer, or for which inbuf points to a null pointer. When iconv() is called in this way, and if outbuf is not a null pointer or a pointer to a null pointer, and outbytesleft 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 inbuf 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.

If a sequence of input bytes does not form a valid character in the specified code set, conversion stops after the previous successfully converted character. If the input buffer ends with an incomplete character or shift sequence, conversion stops after the previous successfully converted bytes. If the output buffer is not large enough to hold the entire converted input, conversion stops just prior to the input bytes that would cause the output buffer to overflow. The variable pointed to by inbuf is updated to point to the byte following the last byte successfully used in the conversion. The value pointed to by inbytesleft is decremented to reflect the number of bytes still not converted in the input buffer. The variable pointed to by outbuf is updated to point to the byte following the last byte of converted output data. The value pointed to by outbytesleft 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.

If iconv() encounters a character in the input buffer that is legal, but for which an identical character does not exist in the target code set, iconv() performs an implementation-defined conversion on this character.

RETURN VALUES
The iconv() function updates the variables pointed to by the arguments to reflect the extent of the conversion and returns the number of non-identical conversions performed. If the entire string in the input buffer is converted, the value pointed to by inbytesleft will...
be 0. If the input conversion is stopped due to any conditions mentioned above, the value pointed to by inbytesleft will be non-zero and errno is set to indicate the condition. If an error occurs iconv() returns (size_t) −1 and sets errno to indicate the error.

**ERRORS**

The iconv() function will fail if:

- **EILSEQ** Input conversion stopped due to an input byte that does not belong to the input code set.
- **E2BIG** Input conversion stopped due to lack of space in the output buffer.
- **EINVAL** Input conversion stopped due to an incomplete character or shift sequence at the end of the input buffer.

The iconv() function may fail if:

- **EBADF** The cd argument is not a valid open conversion descriptor.

**FILES**

/usr/lib/iconv/*.so conversion modules

**ATTRIBUTES**

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**SEE ALSO**

iconv(1), iconv_close(3), iconv_open(3), attributes(5), iconv(5), iconv_unicode(5)
NAME     iconv_close – code conversion deallocation function

SYNOPSIS  
#include <iconv.h>
int iconv_close(iconv_t cd);

DESCRIPTION  The iconv_close() function deallocates the conversion descriptor cd and all other associated resources allocated by the iconv_open(3) function.
If a file descriptor is used to implement the type iconv_t, that file descriptor will be closed.

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.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  iconv(3), iconv_open(3), attributes(5)
NAME
iconv_open – code conversion allocation function

SYNOPSIS
#include <iconv.h>
iconv_t iconv_open(const char *tocode, const char *fromcode);

DESCRIPTION
The iconv_open() function returns a conversion descriptor that describes a conversion from the codeset specified by the string pointed to by the fromcode argument to the codeset specified by the string pointed to by the tocode argument. For state-dependent encodings, the conversion descriptor will be in a codeset-dependent initial shift state, ready for immediate use with the iconv(3) function.

Settings of fromcode and tocode and their permitted combinations are implementation-dependent.

A conversion descriptor remains valid in a process until that process closes it.

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
The iconv_open function may fail if:

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 fromcode and tocode is not supported by the implementation.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
iconv(3), iconv_close(3), malloc(3C), attributes(5)

NOTES
iconv_open() uses malloc(3C) to allocate space for internal buffer areas. iconv_open() may fail if there is insufficient storage space to accommodate these buffers.

Portable applications must assume that conversion descriptors are not valid after a call to one of the exec functions.

modified 29 Dec 1996 SunOS 5.6 3-825
### NAME
idcok – enable/disable hardware insert-character and delete-character features

### SYNOPSIS
```c
#include <curses.h>

void idcok(WINDOW *win, bool bf);
```

### ARGUMENTS
- **win**: Is a pointer to a window.
- **bf**: Is a Boolean expression.

### DESCRIPTION
The idcok() function enables or disables the use of hardware insert-character and delete-character features in `win`. If `bf` is set to `TRUE`, the use of these features in `win` is enabled (if the terminal is equipped). If `bf` is set to `FALSE`, their use in `win` is disabled.

### RETURN VALUES
The idcok() function does not return a value.

### ERRORS
None.

### SEE ALSO
- clearok(3XC), doupdate(3XC)
NAME
ilogb – returns an unbiased exponent

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

DESCRIPTION
The ilogb() function returns the exponent part of x. Formally, the return value is the
integral part of \( \log_r |x| \) as a signed integral value, for non-zero finite x, where \( r \) is the
radix of the machine’s floating point arithmetic.

RETURN VALUES
Upon successful completion, ilogb() returns the exponent part of x.
If x is 0, ilogb() returns -INT_MAX.
If x is NaN or ±Inf, ilogb() returns INT_MAX.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
logb(3M), attributes(5)
NAME  
immedok – call refresh on changes to window

SYNOPSIS  
#include <curses.h>
int immedok(WINDOW *win, bool bf);

ARGUMENTS  
win Is a pointer to the window that is to be refreshed.
bf Is a Boolean expression.

DESCRIPTION  
If bf is TRUE, immedok() calls refresh(3XC) if any change to the window image is made
(for example, through functions such as addch(3XC), clrtobot(3XC), and scrl(3XC)).
Repeated calls to refresh() may affect performance negatively. The immedok() function
is disabled by default.

RETURN VALUES  
The immedok() function does not return a value.

ERRORS  
None.

SEE ALSO  
addch(3XC), clearok(3XC), clrtobot(3XC), doupdate(3XC), scrl(3XC)
NAME
inch, mvinch, mvwinch, winch – return a single-byte character (with rendition)

SYNOPSIS
#include <curses.h>

ctype inch(void);
ctype mvinch(int y, int x);
ctype mvwinch(WINDOW *win, int y, int x);
ctype winch(WINDOW *win);

ARGUMENTS
y Is the y (row) coordinate of the position of the character to be returned.
x Is the x (column) coordinate of the position of the character to be returned.
win Is a pointer to the window that contains the character to be returned.

DESCRIPTION
The inch() and winch() functions return the ctype character located at the current cursor position of the stdscr window and window win, respectively. The mvinch() and mvwinch() functions return the ctype character located at the position indicated by the x (column) and y (row) parameters (the former in the stdscr window; the latter in window win).

The complete character/attribute pair will be returned. The character or attributes can be extracted by performing a bitwise AND on the returned value, using the constants A_CHARTEXT, A_ATTRIBUTES, and A_COLOR.

RETURN VALUES
On success, these functions return the specified character and rendition. Otherwise, they return ERR.

ERRORS
None.

SEE ALSO
addch(3XC), attroff(3XC)
NAME
inchnstr, inchstr, mvinchnstr, mvinchstr, mvwinchnstr, mvwinchstr, winchnstr, winchstr

− retrieve a single-byte character string (with rendition)

SYNOPSIS
#include <curses.h>

int inchnstr(chtype *chstr, int n);
int inchstr(chtype *chstr);
int mvinchnstr(int y, int x, chtype *chstr, int n);
int mvinchstr(int y, int x, chtype *chstr);
int mvwinchnstr(WINDOW *win, int y, int x, chtype *chstr, int n);
int mvwinchstr(WINDOW *win, int y, int x, chtype *chstr);
int winchnstr(WINDOW *win, chtype *chstr, int n);
int winchstr(WINDOW *win, chtype *chstr);

ARGUMENTS
chstr Is a pointer to an object that can hold the retrieved character string.
n Is the number of characters not to exceed when retrieving chstr.
y Is the y (row) coordinate of the starting position of the string to be retrieved.
x Is the x (column) coordinate of the starting position of the string to be retrieved.
win Is a pointer to the window in which the string is to be retrieved.

DESCRIPTION
The inchstr() and winchstr() functions retrieve the character string (with rendition) starting at the current cursor position of the stdscr window and window win, respectively, and ending at the right margin. The mvinchstr() and mvwinchstr() functions retrieve the character string located at the position indicated by the x (column) and y (row) parameters (the former in the stdscr window; the latter in window win).

The inchnstr(), winchnstr(), mvinchnstr(), and mvwinchnstr() functions retrieve at most n characters from the window stdscr and win, respectively. The former two functions retrieve the string, starting at the current cursor position; the latter two commands retrieve the string, starting at the position specified by the x and y parameters.

All these functions store the retrieved character string in the object pointed to by chstr.

The complete character/attribute pair is retrieved. The character or attributes can be extracted by performing a bitwise AND on the retrieved value, using the constants A_CHARTEXT, A_ATTRIBUTES, and A_COLOR. The character string can also be retrieved without attributes by using instr(3XC) set of functions.

RETURN VALUES
On success, these functions return OK. Otherwise, they return ERR.

ERRORS
None.
SEE ALSO  inch(3XC), innstr(3XC)

modified 1 Jun 1996

SunOS 5.6

3XC-831
NAME
index, rindex – string operations

SYNOPSIS
```
#include <strings.h>
char *index(const char *s, int c);
char *rindex(const char *s, int c);
```

DESCRIPTION
These functions operate on null-terminated strings.

index() returns a pointer to the first occurrence of character c in string s, and rindex() returns a pointer to the last occurrence of character c in string s. Both index() and rindex() return a null pointer if c does not occur in the string. The null character terminating a string is considered to be part of the string.

SEE ALSO
bstring(3C), malloc(3C), string(3C)

NOTES
On most modern computer systems, you can not 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.
NAME
inet, inet_addr, inet_network, inet_makeaddr, inet_lnaof, inet_netof, inet_ntoa – Internet address manipulation

SYNOPSIS
cc [flag ...] file ... -lssocket -lsocket [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);

DESCRIPTION
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.

The routine inet_ntoa() returns a pointer to a string in the base 256 notation d.d.d.d. See INTERNET ADDRESSES.

Internet addresses are returned in network order (bytes ordered from left to right). Network numbers and local address parts are returned as machine format integer values.

INTERNET ADDRESSES
Values specified using '.' notation take one of the following forms:
   a.b.c.d
   a.b.c
   a.b
   a

When four parts are specified, each is interpreted as a byte of data and assigned, from left to right, to the four bytes of an Internet address.

When a three part address is specified, the last part is interpreted as a 16-bit quantity and placed in the right most two bytes of the network address. This makes the three part address format convenient for specifying Class B network addresses as 128.net.host.
When a two part address is supplied, the last part is interpreted as a 24-bit quantity and placed in the right most three bytes of the network address. This makes the two part address format convenient for specifying Class A network addresses as net.host.

When only one part is given, the value is stored directly in the network address without any byte rearrangement.

Numbers supplied as parts in ‘.’ notation may be decimal, octal, or hexadecimal, as specified in the C language. For example, a leading 0x or 0X implies hexadecimal; otherwise, a leading 0 implies octal; otherwise, the number is interpreted as decimal.

**RETURN VALUES**

The value −1 is returned by `inet_addr()` and `inet_network()` for malformed requests.
The routines `inet_netof()` and `inet_lnaof()` break apart Internet host addresses, returning the network number and local network address part, respectively.
The routine `inet_ntoa()` returns a pointer to a string in the base 256 notation d.d.d.d described in **INTERNET ADDRESSES**.

**ATTRIBUTES**

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

**SEE ALSO**

`gethostbyname(3N)`, `getnetbyname(3N)`, `hosts(4)`, `networks(4)`, `attributes(5)`, `inet(5)`

**NOTES**

The return value from `inet_ntoa()` points to a buffer which is overwritten on each call. This buffer is implemented as thread-specific data in multithreaded applications.

**BUGS**

The problem of host byte ordering versus network byte ordering is confusing. A simple way to specify Class C network addresses in a manner similar to that for Class B and Class A is needed.
NAME
inet_addr, inet_network, inet_makeaddr, inet_lnaof, inet_netof, inet_ntoa – Internet address manipulation

SYNOPSIS
cc [flag ...] file ... -lxnet [ library ... ]
#include <arpa/inet.h>
in_addr_t inet_addr(const char *cp);
in_addr_t inet_lnaof(struct in_addr in);
struct in_addr inet_makeaddr(in_addr_t net, in_addr_t lna);
in_addr_t inet_netof(struct in_addr in);
in_addr_t inet_network(const char *cp);
char *inet_ntoa(struct in_addr in);

DESCRIPTION
The inet_addr() function converts the string pointed to by cp, in the Internet standard dot notation, to an integer value suitable for use as an Internet address.
The inet_lnaof() function takes an Internet host address specified by in and extracts the local network address part, in host byte order.
The inet_makeaddr() function takes the Internet network number specified by net and the local network address specified by lna, both in host byte order, and constructs an Internet address from them.
The inet_netof() function takes an Internet host address specified by in and extracts the network number part, in host byte order.
The inet_network() function converts the string pointed to by cp, in the Internet standard dot notation, to an integer value suitable for use as an Internet network number.
The inet_ntoa() function converts the Internet host address specified by in to a string in the Internet standard dot notation.
All Internet addresses are returned in network order (bytes ordered from left to right).
Values specified using dot notation take one of the following forms:

a.b.c.d When four parts are specified, each is interpreted as a byte of data and assigned, from left to right, to the four bytes of an Internet address.
a.b.c When a three-part address is specified, the last part is interpreted as a 16-bit quantity and placed in the rightmost two bytes of the network address. This makes the three-part address format convenient for specifying Class B network addresses as 128.net.host.
a.b When a two-part address is supplied, the last part is interpreted as a 24-bit quantity and placed in the rightmost three bytes of the network address. This makes the two-part address format convenient for specifying Class A network addresses as net.host.
a When only one part is given, the value is stored directly in the network address without any byte rearrangement.
inet_addr (3XN) X/Open Networking Services Library Functions

All numbers supplied as parts in dot notation may be decimal, octal, or implies hexadecimal; otherwise, a leading 0 implies octal; otherwise, the number is interpreted as decimal).

RETURN VALUES

Upon successful completion, inet_addr() returns the Internet address. Otherwise, it returns (in_addr_t)−1.

Upon successful completion, inet_network() returns the converted Internet network number. Otherwise, it returns (in_addr_t)−1.

The inet_makeaddr() function returns the constructed Internet address.

The inet_lnaof() function returns the local network address part.

The inet_netof() function returns the network number.

The inet_ntoa() function returns a pointer to the network address in Internet-standard dot notation.

ERRORS

No errors are defined.

USAGE

The return value of inet_ntoa() may point to static data that may be overwritten by subsequent calls to inet_ntoa().

ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO endhostent(3XN), endnetent(3XN), attributes(5), inet(5)
NAME  initgroups – initialize the supplementary group access list

SYNOPSIS  #include <grp.h>
#include <sys/types.h>

int initgroups(const char *name, gid_t basegid);

DESCRIPTION  initgroups() reads the group database to get the group membership for the user specified by name and then initializes the supplementary group access list of the calling process (see getgrnam(3C) and getgroups(2)). The basegid group id is also included in the supplementary group access list. This is typically the real group id from the user database.

While scanning the group database, if the number of groups, including the basegid entry, exceeds [NGROUPS_MAX], subsequent group entries are ignored.

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.

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

ATTRIBUTE TYPE | ATTRIBUTE VALUE
--- | ---
MT-Level | Unsafe

SEE ALSO  getgroups(2), getgrnam(3C), attributes(5)
NAME

initscr, newterm – screen initialization functions

SYNOPSIS

#include <curses.h>
WINDOW *initscr(void);
SCREEN *newterm(char *type, FILE *outfp, FILE *infp);

ARGUMENTS

type Is a string defining the terminal type to be used in place of TERM.
outfp Is a pointer to a file to be used for output to the terminal.
infp Is the pointer to a file to be used for input to the terminal.

DESCRIPTION

The initscr() function initializes X/Open Curses data structures, determines the terminal type, and ensures the first call to refresh(3XC) clears the screen.

The newterm() function opens a new terminal with each call. It should be used instead of initscr() when the program interacts with more than one terminal. It returns a variable of type SCREEN, which should be used for later reference to that terminal. Before program termination, endwin() should be called for each terminal.

The only functions that you can call before calling initscr() or newterm() are filter(3XC), ripofline(3XC), slk_init(3XC), and use_env(3XC).

RETURN VALUES

On success, the initscr() function returns a pointer to stdscr; otherwise, initscr() does not return.

On success, the newterm() function returns a pointer to the specified terminal; otherwise, a null pointer is returned.

ERRORS

None.

SEE ALSO
del_curterm(3XC), delscreen(3XC), douupdate(3XC), endwin(3XC), filter(3XC),
slk_attroff(3XC), use_env(3XC)
NAME  innstr, instr, mvinnstr, mvinstr, mvwinnstr, mvwinstr, winnstr, winstr ─ retrieve a multi-byte character string (without rendition)

SYNOPSIS

```
#include <curses.h>

int innstr(char *str, int n);

int instr(char *str);

int mvinnstr(int y, int x, char *str, int n);

int mvinstr(int y, int x, char *str);

int mvwinnstr(WINDOW *win, int y, int x, char *str, int n);

int mvwinstr(WINDOW *win, int x, char *str);

int winstr(WINDOW *win, char *str);

int winnstr(WINDOW *win, char *str, int n);
```

ARGUMENTS

- **str**  Is a pointer to an object that can hold the retrieved multibyte character string.
- **n**  Is the number of characters not to exceed when retrieving **str**.
- **y**  Is the y (row) coordinate of the starting position of the string to be retrieved.
- **x**  Is the x (column) coordinate of the starting position of the string to be retrieved.
- **win**  Is a pointer to the window in which the string is to be retrieved.

DESCRIPTION

The **instr()** and **winstr()** functions retrieve a multibyte character string (without attributes) starting at the current cursor position of the **stdscr** window and window **win**, respectively, and ending at the right margin. The **mvinstr()** and **mvwinstr()** functions retrieve a multibyte character string located at the position indicated by the **x** (column) and **y** (row) parameters (the former in the **stdscr** window; the latter in window **win**). The **innstr()**, **winnstr()**, **mvinnstr()**, and **mvwinnstr()** functions retrieve at most **n** characters from the window **stdscr** and **win**, respectively. The former two functions retrieve the string starting at the current cursor position; the latter two commands return the string, starting at the position specified by the **x** and **y** parameters.

All these functions store the retrieved string in the object pointed to by **str**. They only store complete multibyte characters. If the area pointed to by **str** is not large enough to hold at least one character, these functions fail.

Only the character portion of the character/rendition pair is returned. To return the complete character/rendition pair, use **winchstr()**.

ERRORS

- **OK**  Successful completion.
- **ERR**  An error occurred.
| **USAGE**   | All functions except `winnstr()` may be macros. |
| **SEE ALSO** | `inch(3XC)`, `inchstr(3XC)` |
NAME
innwstr, inwstr, mvinnwstr, mvinwstr, mvwinwstr, mvwinwstr, winwstr, winwstr –
retrieve a wide character string (without rendition)

SYNOPSIS
#include <curses.h>

int innwstr(wchar_t *wstr, int n);
int inwstr(wchar_t *wstr);
int mvinnwstr(int y, int x, wchar_t *wstr, int n);
int mvinwstr(int y, int x, wchar_t *wstr);
int mvwinwstr(WINDOW *win, int y, int x, wchar_t *wstr, int n);
int mvwinwstr(WINDOW *win, int y, int x, wchar_t *wstr);
int winwstr(WINDOW *win, wchar_t *wstr);
int winwstr(WINDOW *win, wchar_t *wstr, int n);

ARGUMENTS
wstr Is a pointer to an object that can hold the retrieved multibyte character string.
n Is the number of characters not to exceed when retrieving wstr.
y Is the y (row) coordinate of the starting position of the string to be retrieved.
x Is the x (column) coordinate of the starting position of the string to be retrieved.
win Is a pointer to the window in which the string is to be retrieved.

DESCRIPTION
The inwstr() and winwstr() functions retrieve a wide character string (without attributes) starting at the current cursor position of the stdscr window and window win, respectively, and ending at the right margin. The mvinwstr() and mvwinwstr() functions retrieve a wide character string located at the position indicated by the x (column) and y (row) parameters (the former in the stdscr window; the latter in window win).

The innwstr(), winwstr(), mvinnwstr(), and mvwinwstr() functions retrieve at most n characters from the window stdscr and win, respectively. The former two functions retrieve the string starting at the current cursor position; the latter two commands return the string, starting at the position specified by the x and y parameters.

All these functions store the retrieved string in the object pointed to by wstr. They only store complete wide characters. If the area pointed to by wstr is not large enough to hold at least one character, these functions fail.

Only the character portion of the character/rendition pair is returned. To return the complete character/rendition pair, use win_wchstr(3XC).

RETURN VALUES
On success, the inwstr(), mvinwstr(), mvwinwstr(), and winwstr() functions return OK. Otherwise, they return ERR.

On success, the innwstr(), mvinnwstr(), mvwinwstr(), and winwstr() functions return the number of characters read into the string. Otherwise, they return ERR.

modified 1 Jun 1996
SunOS 5.6
3XC-841
ERRORS
None.

SEE ALSO
in_wch(3XC), in_wchnstr(3XC)
NAME
insch, winsch, mvinsch, mvwinsch – insert a character

SYNOPSIS
#include <curses.h>

int insch(chtype ch);
int mvinsch(int y, int x, chtype ch);
int mvwinsch(WINDOW *win, int y, int x, chtype ch);
int winsch(WINDOW *win, chtype ch);

ARGUMENTS
ch Is the character to be inserted.
y Is the y (row) coordinate of the position of the character.
x Is the x (column) coordinate of the position of the character.
win Is a pointer to the window in which the character is to be inserted.

DESCRIPTION
The insch() function inserts the chtype character ch at the current cursor position of the stdscr window. The winsch() function performs the identical action but in window win. The mvinsch() and mvwinsch() functions insert the character at the position indicated by the x (column) and y (row) parameters (the former in the stdscr window; the latter in window win). The cursor position does not change.

All characters to the right of the inserted character are moved right one character. The last character on the line is deleted.

Insertions and deletions occur at the character level. The cursor is adjusted to the first column of the character prior to the operation.

RETURN VALUES
On success, these functions return OK. Otherwise, they return ERR.

ERRORS
None.

SEE ALSO
delch(3XC), insnstr(3XC)
# NAME
insdelln, winsdelln – insert/delete lines to/from the window

## SYNOPSIS

```c
#include <curses.h>

int insdelln(int n);
int winsdelln(WINDOW *win, int n);
```

## ARGUMENTS

- `n` is the number of lines to insert or delete (positive `n` inserts; negative `n` deletes).
- `win` is a pointer to the window in which to insert or delete a line.

## DESCRIPTION
The `insdelln()` and `windsdelln()` functions insert or delete blank lines in `stdscr` or `win`, respectively. When `n` is positive, `n` lines are added before the current line and the bottom `n` lines are lost; when `n` is negative, `n` lines are deleted starting with the current line, the remaining lines are moved up, and the bottom `n` lines are cleared. The position of the cursor does not change.

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

## ERRORS
None.

## SEE ALSO
deleteln(3XC), insertln(3XC)
NAME
insertln, wininsertln – insert a line in a window

SYNOPSIS
#include <curses.h>
int insertln(void);
int wininsertln(WINDOW *win);

ARGUMENTS
win Is a pointer to the window in which to insert the line.

DESCRIPTION
The insertln() and wininsertln() functions insert a blank line before the current line in stdscr or win, respectively. The new line becomes the current line. The current line and all lines after it in the window are moved down one line. The bottom line in the window is discarded.

RETURN VALUES
On success, these functions return OK. Otherwise, they return ERR.

ERRORS
None.

SEE ALSO
bkgdset(3XC), deleteln(3XC), insdelln(3XC)
NAME
insnstr, insstr, mvinsnstr, mvinsstr, mvwinsnstr, mvwinsstr, winsnstr, winsstr – insert a multibyte character string

SYNOPSIS
#include <curses.h>

int insnstr(const char *str, int n);
int insstr(const char *str);
int mvinsnstr(int y, int x, const char *str, int n);
int mvinsstr(int y, int x, const char *str);
int mvwinsnstr(WINDOW *win, int y, int x, const char *str, int n);
int mvwinsstr(WINDOW *win, int y, int x, const char *str);
int winsnstr(WINDOW *win, const char *str, int n);
int winsstr(WINDOW *win, const char *str);

ARGUMENTS
str Is a pointer to the string to be inserted.
n Is the number of characters not to exceed when inserting str. If n is less than 1, the entire string is inserted.
y Is the y (row) coordinate of the starting position of the string.
x Is the x (column) coordinate of the starting position of the string.
win Is a pointer to the window in which the string is to be inserted.

DESCRIPTION
The insnstr() function inserts str at the current cursor position of the stdscr window. The winsstr() function performs the identical action, but in window win. The mvinsnstr() and mvwinsnstr() functions insert the character string at the starting position indicated by the x (column) and y (row) parameters (the former to the stdscr window; the latter to window win).

The insnstr(), winsnstr(), mvinsnstr(), and mvwinsnstr() functions insert n characters to the window or as many as will fit on the line. If n is less than 1, the entire string is inserted or as much of it as fits on the line. The former two functions place the string at the current cursor position; the latter two commands use the position specified by the x and y parameters.

All characters to the right of inserted characters are moved to the right. Characters that don’t fit on the current line are discarded. The cursor is left at the point of insertion.

If a character in str is a newline, carriage return, backspace, or tab, the cursor is moved appropriately. The cursor is moved to the next tab stop for each tab character (by default, tabs are eight characters apart). If the character is a control character other than those previously mentioned, the character is inserted using \"x notation, where x is a printable character. clrtoeol(3XC) is automatically done before a newline.
### RETURN VALUES
On success, these functions return **OK**. Otherwise, they return **ERR**.

### ERRORS
None.

### SEE ALSO
- `addchstr(3XC)`, `addstr(3XC)`, `clrtoeol(3XC)`, `ins_nwstr(3XC)`, `insch(3XC)`
NAME

ins_nwstr, ins_wstr, mvins_nwstr, mvins_wstr, mvwins_nwstr, mvwins_nstr,
wins_nwstr, wins_wstr – insert a wide character string

SYNOPSIS

#include <curses.h>

int ins_nwstr(const wchar_t *wstr, int n);
int ins_wstr(const wchar_t *wstr);
int mvins_nwstr(int y, int x, const wchar_t *wstr, int n);
int mvins_wstr(int y, int x, const wchar_t *wstr);
int mvwins_nwstr(WINDOW *win, int y, int x,
                 const wchar_t *wstr, int n);
int mvwins_wstr(WINDOW *win, int y, int x,
                 const wchar_t *wstr);
int wins_nwstr(WINDOW *win, const wchar_t *wstr, int n);
int wins_wstr(WINDOW *win, const wchar_t *wstr);

ARGUMENTS

wstr  Is a pointer to the string to be inserted.
n     Is the number of characters not to exceed when inserting wstr. If n is less than 1,
     the entire string is inserted.
y     Is the y (row) coordinate of the starting position of the string.
x     Is the x (column) coordinate of the starting position of the string.
win   Is a pointer to the window in which the string is to be inserted.

DESCRIPTION

The ins_wstr() function inserts wstr at the current cursor position of the stdscr window. The
wins_wstr() function performs the identical action, but in window win. The
mvins_wstr() and mvwins_wstr() functions insert wstr string at the starting position indicated by the x (column) and y (row) parameters (the former in the stdscr window; the
latter in window win).

The ins_nwstr(), wins_nwstr(), mvins_nwstr(), and mvwins_nwstr() functions insert n
characters to the window or as many as will fit on the line. If n is less than 1, the entire
string is inserted or as much of it as fits on the line. The former two functions place the
string at the current cursor position; the latter two commands use the position specified
by the x and y parameters.

All characters to the right of inserted characters are moved to the right. Characters that
don’t fit on the current line are discarded. The cursor is left at the point of insertion.
If a character in wstr is a newline, carriage return, backspace, or tab, the cursor is moved
appropriately. The cursor is moved to the next tab stop for each tab character (by default,
tabs are eight characters apart). If the character is a control character other than those
previously mentioned, the character is inserted using "x notation, where x is a printable
character. clrtoeol(3XC) is automatically done before a newline.
### RETURN VALUES
On success, these functions return **OK**. Otherwise, they return **ERR**.

### ERRORS
None.

### SEE ALSO
- `add_wchnstr(3XC)`, `addnwstr(3XC)`, `clrtoeol(3XC)`, `ins_wch(3XC)`, `insnstr(3XC)`
NAME  insque, remque – insert/remove element from a queue

SYNOPSIS  include <search.h>
void insque(struct qelem *elem, struct qelem *pred);
void remque(struct qelem *elem);

DESCRIPTION  insque() and remque() manipulate queues built from doubly linked lists. Each element in the queue must be in the following form:

  struct qelem {
    struct qelem *q_forw;
    struct qelem *q_back;
    char q_data[ ];
  };

insque() inserts elem in a queue immediately after pred. remque() removes an entry elem from a queue.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO  attributes(5)
NAME

ins_wch, wins_wch, mvins_wch, mvwins_wch – insert a complex character

SYNOPSIS

```
#include <curses.h>

int ins_wch(const cchar_t *wch);
int mvins_wch(int y, int x, const cchar_t *wch);
int mvwins_wch(WINDOW *win, int y, int x,
               const cchar_t *wch);
int wins_wch(WINDOW *win, const cchar_t *wch);
```

ARGUMENTS

- `wch` Is the complex character to be inserted.
- `y` Is the y (row) coordinate of the position of the character.
- `x` Is the x (column) coordinate of the position of the character.
- `win` Is a pointer to the window in which the character is to be inserted.

DESCRIPTION

The `ins_wch()` function inserts the complex character `wch` at the current cursor position of the `stdscr` window. The `wins_wch()` function performs the identical action but in window `win`. The `mvins_wch()` and `mvwins_wch()` functions insert the character at the position indicated by the `x` (column) and `y` (row) parameters (the former in the `stdscr` window; the latter in window `win`). The cursor position does not change.

All characters to the right of the inserted character are moved right one character. The last character on the line is deleted.

Insertions and deletions occur at the character level. The cursor is adjusted to the first column of the character prior to the operation.

RETURN VALUES

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

ERRORS

None.

SEE ALSO

`add_wch(3XC)`, `ins_nwstr(3XC)`
NAME
intrflush – flush output in tty on interrupt

SYNOPSIS
#include <curses.h>
int intrflush(WINDOW *win, bool bf);

ARGUMENTS
win Is ignored.
bf Is a Boolean expression.

DESCRIPTION
If this option is enabled (bf is TRUE), intrflush() flushes all output in the terminal driver
when an interrupt, quit, or suspend character is sent to the terminal. This increases inter-
rupt response time but causes X/Open Curses to lose track of what currently exists on
the screen. If this option is disabled (bf is FALSE), intrflush() does not flush output on an
interrupt, quit, or suspend character. Whether this option is enabled or disabled by
default depends on the tty driver.

RETURN VALUES
On success, these functions return OK. Otherwise, they return ERR.

ERRORS
None.

SEE ALSO
flushinp(3XC), qiflush(3XC)
NAME

in_wch, mvin_wch, mvwin_wch, win_wch – retrieve a complex character (with rendition)

SYNOPSIS

```c
#include <curses.h>

int in_wch(cchar_t *wcval);
int mvin_wch(int y, int x, cchar_t *wcval);
int mvwin_wch(WINDOW *win, int y, int x, cchar_t *wcval);
int win_wch(WINDOW *win, cchar_t *wcval);
```

ARGUMENTS

- `wcval` is a pointer to an object that can store a complex character and its rendition.
- `y` is the y (row) coordinate of the position of the character to be returned.
- `x` is the x (column) coordinate of the position of the character to be returned.
- `win` is a pointer to the window that contains the character to be returned.

DESCRIPTION

The `in_wch()` and `win_wch()` functions retrieve the complex character and its rendition located at the current cursor position of the `stdscr` window and window `win`, respectively. The `mvin_wch()` and `mvwin_wch()` functions retrieve the complex character and its rendition located at the position indicated by the `x` (column) and `y` (row) parameters (the former in the `stdscr` window; the latter in window `win`).

All these functions store the retrieved character and its rendition in the object pointed to by `wcval`.

RETURN VALUES

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

ERRORS

None.

SEE ALSO

`add_wch(3XC)`, `inch(3XC)`

modified 1 Jun 1996

SunOS 5.6

3XC-853
#include <curses.h>

int in_wchnstr(cchar_t *wchstr, int n);
int in_wchstr(cchar_t *wchstr);
int mvin_wchnstr(int y, int x, cchar_t *wchstr, int n);
int mvin_wchstr(int y, int x, cchar_t *wchstr);
int mvwin_wchnstr(WINDOW *win, int y, int x, cchar_t *wchstr, int n);
int mvwin_wchstr(WINDOW *win, int y, int x, cchar_t *wchstr);
int win_wchnstr(WINDOW *win, cchar_t *wchstr, int n);
int win_wchstr(WINDOW *win, cchar_t *wchstr);

ARGUMENTS

wchstr Is a pointer to an object where the retrieved complex character string can be stored.

n Is the number of characters not to exceed when retrieving wchstr.

y Is the y (row) coordinate of the starting position of the string to be retrieved.

x Is the x (column) coordinate of the starting position of the string to be retrieved.

win Is a pointer to the window in which the string is to be retrieved.

DESCRIPTION

The in_wchstr() and win_wchstr() functions retrieve a complex character string (with rendition) starting at the current cursor position of the stdscr window and window win, respectively, and ending at the right margin. The mvin_wchstr() and mvwin_wchstr() functions retrieve a complex character string located at the position indicated by the x (column) and y (row) parameters (the former in the stdscr window; the latter in window win).

The in_wchnstr(), win_wchnstr(), mvin_wchnstr(), and mvwin_wchnstr() functions retrieve at most n characters from the window stdscr and win, respectively. The former two functions retrieve the string, starting at the current cursor position; the latter two commands retrieve the string, starting at the position specified by the x and y parameters. The retrieved character string (with renditions) is stored in the object pointed to by wcval.

RETURN VALUES

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

ERRORS

None.

SEE ALSO

in_wch(3XC)
NAME
isastream – test a file descriptor

SYNOPSIS
#include <stropts.h>
int isastream(int fildes);

DESCRIPTION
The function isastream() determines if a file descriptor represents a STREAMS file. fildes refers to an open file descriptor.

RETURN VALUES
If successful, isastream() returns 1 if fildes 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 fildes is not a valid file descriptor.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
attributes(5), streamio(7I)
STREAMS Programming Guide
isatty (3C) C Library Functions

NAME isatty – test for a terminal device

SYNOPSIS #include <unistd.h>
int isatty(int fildes);

DESCRIPTION The isatty() function tests whether fildes, an open file descriptor, is associated with a terminal device.

RETURN VALUES The isatty() function returns 1 if fildes is associated with a terminal; otherwise it returns 0 and may set errno to indicate the error.

ERRORS The isatty() function may fail if:
EBADF The fildes argument is not a valid open file descriptor.
ENOTTY The fildes argument is not associated with a terminal.

USAGE The isatty() function does not necessarily indicate that a human being is available for interaction via fildes. It is quite possible that non-terminal devices are connected to the communications line.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe in multi-thread applications</td>
</tr>
</tbody>
</table>

SEE ALSO ttyname(3C), attributes(5)

3C-856 SunOS 5.6 modified 29 Dec 1996
NAME
isencrypt – determine whether a buffer of characters is encrypted

SYNOPSIS
cc [ flag . . . ] file . . . -lgen [ library . . . ]
#include <libgen.h>
int isencrypt(const char *fbuf, size_t ninbuf);

DESCRIPTION
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.

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 ninbuf characters, isen-
crypt() assumes that the buffer is encrypted if the setlocale() LC_CTYPE category is set to
C or ascii.

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 ninbuf 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.

RETURN VALUES
If the buffer is encrypted, 1 is returned; otherwise zero is returned.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
setlocale(3C), attributes(5)

NOTES
When compiling multi-thread applications, the _REENTRANT flag must be defined on
the compile line. This flag should only be used in multi-thread applications.
NAME  
is_linetouched, is_wintouched, touchline, touchwin, untouchwin, wtouchln – control window refresh

SYNOPSIS  
#include <curses.h>

bool is_linetouched(WINDOW *win, int line);
bool is_wintouched(WINDOW *win);
int touchline(WINDOW *win, int start, int count);
int touchwin(WINDOW *win);
int untouchwin(WINDOW *win);
int wtouchln(WINDOW *win, int y, int n, int changed);

ARGUMENTS  
win  Is a pointer to the window in which the refresh is to be controlled or monitored.
line  Is the line to be checked for change since refresh.
start  Is the starting line number of the portion of the window to make appear changed.
count  Is the number of lines in the window to mark as changed.
y  Is the starting line number of the portion of the window to make appear changed or not changed.
n  Is the number of lines in the window to mark as changed.
changed  Is a flag indicating whether to make lines look changed (0) or not changed (1).

DESCRIPTION  
The touchwin() function marks the entire window as dirty. This makes it appear to X/Open Curses as if the whole window has been changed, thus causing the entire window to be rewritten with the next call to refresh(3XC). This is sometimes necessary when using overlapping windows; the change to one window will not be reflected in the other and, hence will not be recorded.

The touchline() function marks as dirty a portion of the window starting at line start and continuing for count lines instead of the entire window. Consequently, that portion of the window is updated with the next call to refresh().

The untouchwin() function marks all lines in the window as unchanged since the last refresh, ensuring that it is not updated.

The wtouchln() function marks n lines starting at line y as either changed (changed=1) or unchanged (changed=0) since the last refresh.

To find out which lines or windows have been changed since the last refresh, use the is_linetouched() and is_wintouched() commands, respectively. These return TRUE if the specified line or window have been changed since the last call to refresh() or FALSE if no changes have been made.
RETURN VALUES
On success, these functions return OK. Otherwise, they return ERR.

ERRORS
None.

SEE ALSO
doupdate(3XC)
NAME  isnan, isnand, isnanf, finite, fpclass, unordered – determine type of floating-point number

SYNOPSIS  
#include <ieeefp.h>

int isnand(double dsrsrc);
int isnanf(float fsrc);
int finite(double dsrsrc);
fpclass_t fpclass(double dsrsrc);
int unordered(double dsrsrc1, double dsrsrc2);

#include <math.h>

int isnan(double dsrsrc);

DESCRIPTION  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 dsrsrc belongs to. The 10 possible classes are as follows:

| 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 |

None of these routines generate any exception, even for signaling NaNs.

RETURN VALUES  isnan(), isnand(), and isnanf() return true (1) if the argument dsrsrc or fsrc is a NaN; otherwise they return false (0).

finite() returns true (1) if the argument dsrsrc is neither infinity nor NaN; otherwise it returns false (0).

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.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

3C-860  SunOS 5.6  modified 29 Dec 1996
SEE ALSO  fpgetround(3C), attributes(5)
NAME
isnan – test for NaN

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

DESCRIPTION
The isnan() function tests whether $x$ is NaN.

RETURN VALUES
The isnan() function returns non-zero if $x$ is NaN. Otherwise, 0 is returned.

USAGE
On systems not supporting NaN, isnan() always returns 0.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
attributes(5)
**NAME**

iswalpha, iswupper, iswlower, iswdigit, iswxdigit, iswalnum, iswspace, iswpunct, iswprint, iswcntrl, iswascii, iswgraph, isphonogram, isideogram, isenglish, isnumber, isspecial – wide-character code classification functions

**SYNOPSIS**

```
#include <wchar.h>

int iswalpha(wint_t wc);
```

**DESCRIPTION**

These functions test whether `wc` is a wide-character code representing a character of a particular class defined in the LC_CTYPE category of the current locale.

In all cases, `wc` is a `wint_t`, the value of which must be a wide-character code corresponding to a valid character in the current locale or must equal the value of the macro `WEOF`. If the argument has any other values, the behavior is undefined.

- `iswalpha(wc)` tests whether `wc` is a wide-character code representing a character of class "alpha" in the program’s current locale.
- `iswupper(wc)` tests whether `wc` is a wide-character code representing a character of class "upper" in the program’s current locale.
- `iswlower(wc)` tests whether `wc` is a wide-character code representing a character of class "lower" in the program’s current locale.
- `iswdigit(wc)` tests whether `wc` is a wide-character code representing a character of class "digit" in the program’s current locale.
- `iswxdigit(wc)` tests whether `wc` is a wide-character code representing a character of class "xdigit" in the program’s current locale.
- `iswalnum(wc)` tests whether `wc` is a wide-character code representing a character of class "alpha" or "digit" in the program’s current locale.
- `iswspace(wc)` tests whether `wc` is a wide-character code representing a character of class "space" in the program’s current locale.
- `iswpunct(wc)` tests whether `wc` is a wide-character code representing a character of class "punct" in the program’s current locale.
- `iswprint(wc)` tests whether `wc` is a wide-character code representing a character of class "print" in the program’s current locale.
- `iswgraph(wc)` tests whether `wc` is a wide-character code representing a character of class "graph" in the program’s current locale.
- `iswcntrl(wc)` tests whether `wc` is a wide-character code representing a character of class "cntrl" in the program’s current locale.
- `iswascii(wc)` tests whether `wc` is a wide-character code representing an ASCII character.
- `isphonogram(wc)` tests whether `wc` is a wide-character code representing a phonetic language character, excluding ASCII characters.
iseogram(wc) tests whether wc is a wide-character code representing an ideographic language character, excluding ASCII characters.

isenglish(wc) tests whether wc is a wide-character code representing an English language character, excluding ASCII characters.

isnumber(wc) tests whether wc is a wide-character code representing digit [0–9], excluding ASCII characters.

isspecial(wc) tests whether wc is a wide-character code representing a special language character, excluding ASCII characters.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe with exceptions</td>
</tr>
<tr>
<td>CSI</td>
<td>Enabled</td>
</tr>
</tbody>
</table>

SEE ALSO localedef(1), setlocale(3C), stdio(3S), ascii(5), attributes(5)
NAME       iswctype – test character for specified class

SYNOPSIS   #include <wchar.h>
            int iswctype(wint_t wc, wctype_t charclass);

DESCRIPTION The iswctype() function determines whether the wide-character code wc has the character class charclass, 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 wc argument is not in the domain of the function, the result is undefined. If the value of charclass is invalid (that is, not obtained by a call to wctype(3C) or charclass is invalidated by a subsequent call to setlocale(3C) that has affected category LC_CTYPE), the result is indeterminate.

RETURN VALUES iswctype() returns 0 for FALSE and non-zero for TRUE.

USAGE There are twelve strings that are reserved for the standard character classes:

"alnum"   "alpha"   "blank"
"cntrl"    "digit"   "graph"
"lower"    "print"   "punct"
"space"    "upper"   "xdigit"

In the table below, the functions in the left column are equivalent to the functions in the right column.

| iswalnum(wc) | iswctype(wc, wctype("alnum")) |
| iswalpha(wc) | iswctype(wc, wctype("alpha")) |
| iswcntrl(wc) | iswctype(wc, wctype("cntrl")) |
| iswdigit(wc) | iswctype(wc, wctype("digit")) |
| iswgraph(wc) | iswctype(wc, wctype("graph")) |
| iswlower(wc) | iswctype(wc, wctype("lower")) |
| iswprint(wc) | iswctype(wc, wctype("print")) |
| iswpunct(wc) | iswctype(wc, wctype("punct")) |
| iswspace(wc) | iswctype(wc, wctype("space")) |
| iswupper(wc) | iswctype(wc, wctype("upper")) |
| iswxdigit(wc) | iswctype(wc, wctype("xdigit")) |

The call iswctype(wc, wctype("blank"))
does not have an equivalent isw∗() function.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe with exceptions</td>
</tr>
<tr>
<td>CSI</td>
<td>Enabled</td>
</tr>
</tbody>
</table>

modified 20 Dec 1996  SunOS 5.6  3C-865
### SEE ALSO

- iswalnum(3C), iswalpha(3C), iswcntrl(3C), iswdigit(3C), iswgraph(3C), iswlower(3C), iswprint(3C), iswpunct(3C), iswspace(3C), iswupper(3C), iswxdigit(3C), setlocale(3C), wctype(3C), attributes(5)
NAME
j0, j1, jn – Bessel functions of the first kind

SYNOPSIS
cc [ flag ... ] file ... -lm [ library ... ]
#include <math.h>
double j0(double x);
double j1(double x);
double jn(int n, double x);

DESCRIPTION
The j0(), j1() and jn() functions compute Bessel functions of x of the first kind of orders 0, 1 and n respectively.

RETURN VALUES
Upon successful completion, j0(), j1() and jn() return the relevant Bessel value of x of the first kind.
If the x argument is too large in magnitude, 0 is returned and errno may be set to ERANGE.
If x is NaN, NaN is returned.
For exceptional cases, matherr(3M) tabulates the values to be returned as dictated by Standards other than XPG4.

ERRORS
The j0(), j1() and jn() functions may fail if:
ERANGE The value of x was too large in magnitude.

USAGE
An application wishing to check for error situations should set errno to 0 before calling j0(), j1() or jn(). If errno is non-zero on return, or the return value is NaN, an error has occurred.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
isnan(3M), matherr(3M), y0(3M), attributes(5), standards(5)
### NAME
Kerberos authentication library

### SYNOPSIS
```
cc [flag ...] file ... -lkrb [ library ... ]
#include <kerberos/krb.h>

extern char *krb_err_txt[];

int krb_mk_req(KTEXT authent, const char *service, const char *instance,
               const char *realm, const long checksum);

int krb_rd_req(const KTEXT authent, const char *service, char *instance,
               const long from_addr, AUTH_DAT *ad, const char *fn);

int krb_kntoln(const AUTH_DAT *ad, char *lname);

int krb_set_key(const char *key, const int cvt);

int krb_get_cred(const char *service, const char *instance, const char *realm,
                  CREDENTIALS *c);

long krb_mk_safe(const u_char *in, u_char *out, const u_long in_length,
                 const des_cblock *key, const struct sockaddr_in *sender,
                 const struct sockaddr_in *receiver);

long krb_rd_safe(const u_char *in, const u_long length, const des_cblock *key,
                 const struct sockaddr_in *sender, const struct sockaddr_in *receiver,
                 MSG_DAT *msg_data);

long krb_mk_err(u_char *out, const long code, const char *string);

long krb_rd_err(const u_char *in, const u_long length, long *code,
                 MSG_DAT *msg_data);
```

### DESCRIPTION
This library supports network authentication and various related operations. The library contains many routines beyond those described in this man page, but they are not intended to be used directly. Instead, they are called by the routines that are described, the authentication server and the login program.

`krb_err_txt[]` contains text string descriptions of various Kerberos error codes returned by some of the routines below.

`krb_mk_req()` takes a pointer to a text structure in which an authenticator is to be built. It also takes the name, instance, and realm of the service to be used and an optional checksum. It is up to the application to decide how to generate the checksum. `krb_mk_req()` then retrieves a ticket for the desired service and creates an authenticator. The authenticator is built in `authent` and is accessible to the calling procedure.

It is up to the application to get the authenticator to the service where it will be read by `krb_rd_req()`. Unless an attacker possesses the session key contained in the ticket, it will be unable to modify the authenticator. Thus, the checksum can be used to verify the authenticity of the other data that will pass through a connection.
krb_mk_req() returns KSUCCESS if successful, otherwise a Kerberos error code as defined in `<kerberos/krb.h>`.

krb_rd_req() takes an authenticator of type KTEXT, a service name, an instance, the address of the host originating the request, and a pointer to a structure of type AUTH_DAT which is filled in with information obtained from the authenticator. It also optionally takes the name of the file in which it will find the secret key(s) for the service. If the supplied instance is "*", then the first service key with the same service name found in the service key file will be used, and the instance argument will be filled in with the chosen instance. This means that the caller must provide space for such an instance name. If the last argument is the null string (""), krb_rd_req() will use the file /etc/srvtab to find its keys. If the last argument is NULL, it will assume that the key has been set by krb_set_key() and will not bother looking further.

krb_rd_req() is used to find out information about the principal when a request has been made to a service. It is up to the application protocol to get the authenticator from the client to the service. The authenticator is then passed to krb_rd_req() to extract the desired information.

krb_rd_req() returns zero (RD_AP_OK) upon successful authentication. If a packet was forged, modified, or replayed, authentication will fail. If the authentication fails, a non-zero value is returned indicating the particular problem encountered. See `<kerberos/krb.h>` for the list of error codes.

krb_kntoln() converts a Kerberos name to a local name. It takes a structure of type AUTH_DAT and uses the name, instance, and realm to determine the corresponding local name. A valid local name is returned if the instance is NULL and the realm is the same as the local realm. The local name returned is the Kerberos name and can be used by an application to change uids, directories, or other parameters. This routine is not an integral part of Kerberos, but is provided to support the use of Kerberos in existing utilities. This routine returns KSUCCESS or KFAILURE.

krb_set_key() takes as an argument a DES key. It then creates a key schedule from it and saves the original key to be used as an initialization vector. It is used to set the server’s key which must be used to decrypt tickets.

If called with a non-zero second argument, krb_set_key() will first convert the input from a string of arbitrary length to a DES key by encrypting it with a one-way function. In most cases it should not be necessary to call krb_set_key(). The necessary keys will usually be obtained and set inside krb_rd_req(). krb_set_key() is provided for those applications that do not wish to place the application keys on disk. It returns 0 for success, otherwise a non-zero value.

krb_get_cred() searches the caller’s ticket file for a ticket for the given service, instance, and realm. If a ticket is found, the given CREDENTIALS structure is filled in with the ticket information.

If the ticket was found, krb_get_cred() returns GC_OK. If the ticket file cannot be found, cannot be read, does not belong to the user (other than root), is not a regular file, or is in the wrong mode, the error GC_TKFIL is returned.
**krb_mk_safe()** creates an authenticated, but unencrypted message from any arbitrary application data, pointed to by `in` and `in_length` bytes long. The private session key, pointed to by `key`, is used to seed the `quad_cksum()` checksum algorithm used as part of the authentication. `sender` and `receiver` point to the Internet address of the two parties. This message does not provide privacy, but does protect (via detection) against modifications, insertions or replays. The encapsulated message and header are placed in the area pointed to by `out` and the routine returns the length of the output, or −1 indicating an error.

**krb_rd_safe()** authenticates a received `krb_mk_safe()` message. `in` points to the beginning of the received message, whose length is specified in `in_length`. The private session key, pointed to by `key`, is used to seed the `quad_cksum()` routine as part of the authentication. `msg_data` is a pointer to a MSG_DAT struct, defined in `<kerberos/krb.h>`. The routine fills in these MSG_DAT fields: the `app_data` field with a pointer to the application data, `app_length` with the length of the `app_data` field, `time_sec` and `time_5ms` with the timestamps in the message, and `swap` with a 1 if the byte order of the receiver is different than that of the sender. (The application must still determine if it is appropriate to byte-swap application data; the Kerberos protocol fields are already taken care of.) The routine returns zero if successful, or a Kerberos error code. Modified messages and old messages cause errors, but it is up to the caller to check the time sequence of messages, and to check against recently replayed messages.

**krb_mk_err()** constructs an application level error message that may be used along with `krb_mk_safe()`. `out` is a pointer to the output buffer, `code` is an application specific error code, and `string` is an application specific error string. This routine returns the length of the error reply.

**krb_rd_err()** unpacks a received `krb_mk_err()` message. `in` points to the beginning of the received message, whose length is specified in `in_length`. `code` is a pointer to a value to be filled in with the error value provided by the application. `msg_data` is a pointer to a MSG_DAT struct, defined in `<kerberos/krb.h>`. The routine fills in these MSG_DAT fields: the `app_data` field with a pointer to the application error text, `app_length` with the length of the `app_data` field, and `swap` with a 1 if the byte order of the receiver is different than that of the sender. (The application must still determine if it is appropriate to byte-swap application data; the Kerberos protocol fields are already taken care of). The routine returns zero if the error message has been successfully received, or a Kerberos error code.

The KTEXT structure is used to pass around text of varying lengths. It consists of a buffer for the data, and a length. **krb_rd_req()** takes an argument of this type containing the authenticator, and **krb_mk_req()** returns the authenticator in a structure of this type. KTEXT itself is really a pointer to the structure. The actual structure is of type KTEXT_ST.

The AUTH_DAT structure is filled in by **krb_rd_req()**. It must be allocated before calling **krb_rd_req()**, and a pointer to it is passed. The structure is filled in with data obtained from Kerberos. The MSG_DAT structure is filled in by either **krb_rd_safe()** or **krb_rd_err()**. It must be allocated before the call and a pointer to it is passed. The structure is filled in with data obtained from Kerberos.
FILES
/usr/lib/libkrb.*
/etc/aname
/etc/srvtab
/tmp/tktuid

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
kerberos(1), kerberos_rpc(3N), krb_realmofhost(3N), krb_sendaught(3N),
krb_set_tkt_string(3N), krb.conf(4), krb.realms(4), attributes(5)

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

BUGS
The caller of krb_rd_req() and krb_rd_safe() must check time order and for replay
attempts.

AUTHORS
Clifford Neuman, MIT Project Athena
Steve Miller, MIT Project Athena/Digital Equipment Corporation

RESTRICTIONS
COPYRIGHT 1985,1986,1989 Massachusetts Institute of Technology

modified 30 Dec 1996

SunOS 5.6

3N-871
NAME kerberos_rpc, authkerb_getucred, authkerb_seccreate, svc_kerb_reg – library routines for remote procedure calls using Kerberos authentication

DESCRIPTION RPC library routines allow C programs to make procedure calls on other machines across the network.

RPC supports various authentication flavors. Among them are:

- **AUTH_NONE** (none) no authentication.
- **AUTH_SYS** Traditional UNIX-style authentication.
- **AUTH_DES** DES encryption-based authentication.
- **AUTH_KERB** Kerberos encryption-based authentication.

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).

Routines See rpc(3N) for the definition of the AUTH data structure.

```
cc [ flag ... ] file ... -lkrb [ library ... ]
#include <rpc/rpc.h>
#include <sys/types.h>

int authkerb_getucred(const struct svc_req *rqst, uid_t *uidp, gid_t *gidp, short *gidlenp, int gidlist[NGROUPS]);
```

authkerb_getucred() is used on the server side for converting an AUTH_KERB credential received in an RPC request, which is operating system independent, into an AUTH_SYS credential. This routine returns 1 if it succeeds, 0 if it fails.

*uidp is set to the numerical ID of the user associated with the RPC request referenced by rqst. *gidp is set to the numerical ID of the user’s group. The numerical IDs of the other groups to which the user belongs are stored in gidlist[]. *gidlenp is set to the number of valid group ID entries returned in gidlist[]. All information returned by this routine is based on the Kerberos principal name contained in rqst. This principal name is taken to be the login name of the user, and the IDs returned are the same as if that user had physically logged in to the system.
AUTH *authkerb_seccreate(const char *service, const char *srv_inst,
const char *realm, const 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 resynchronization
due to 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);

csvc_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.

ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO

kerberos(1), kinit(1), kerbd(1M), rpc(3N), rpc_clnt_auth(3N), secure_rpc(3N), svc_run(3N), attributes(5)

NOTES

These interfaces are unsafe in multithreaded applications. Unsafe interfaces should be called only from the main thread.
NAME
keyname, key_name – return character string used as key name

SYNOPSIS
#include <curses.h>
char *keyname(int c);
char *key_name(wchar_t wc);

ARGUMENTS
c Is an 8 bit-character or a key code.
wc Is a wide character key name.

DESCRIPTION
The keyname() function returns a string pointer to the key name. Make a duplicate copy of the returned string if you plan to modify it.
The key_name() function is similar except that it accepts a wide character key name.
The following table shows the format of the key name based on the input.

<table>
<thead>
<tr>
<th>Input</th>
<th>Format of Key Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Visible character</td>
<td>The same character</td>
</tr>
<tr>
<td>Control character</td>
<td>'X</td>
</tr>
<tr>
<td>Meta-character (keyname() only)</td>
<td>M-X</td>
</tr>
<tr>
<td>Key value defined in &lt;curses.h&gt;</td>
<td>KEY_name</td>
</tr>
<tr>
<td>(keyname() only)</td>
<td></td>
</tr>
<tr>
<td>None of the above</td>
<td>UNKNOWN KEY</td>
</tr>
</tbody>
</table>

In the preceding table, X can be either a visible character with the high bit cleared or a control character.

RETURN VALUES
On success, these functions return a pointer to the string used as the key’s name. Otherwise, they return a null pointer.

ERRORS
None.

SEE ALSO
meta(3XC)
NAME         keypad – enable/disable keypad handling
SYNOPSIS     #include <curses.h>
             int keypad(WINDOW *win, bool bf);
ARGUMENTS    win     Is a pointer to the window in which to enable keypad handling.
             bf      Is a Boolean expression.
DESCRIPTION   If bf is TRUE, getch(3XC) handles special keys from the keyboard on the terminal associated with win as single values instead of character sequences. For example, if the user presses the right arrow key, getch() returns a single value, KEY_RIGHT, that represents the function key; otherwise, X/Open Curses handles the special keys as normal text. See getch() for a list of tokens for function keys that are returned by getch().
RETURN VALUES On success, the keypad() function returns OK. Otherwise, it returns ERR.
ERRORS        None.
SEE ALSO      getch(3XC)
NAME    killpg – send signal to a process group

SYNOPSIS  
\#include <signal.h>

    int killpg(pid_t pgrp, int sig);

DESCRIPTION  killpg() sends the signal sig to the process group pgrp. See signal(5) for a list of signals. The real or effective user ID of the sending process must match the real or saved set-user ID of the receiving process, unless the effective user ID of the sending process is the privileged user. A single exception is the signal SIGCONT, which may always be sent to any descendant of the current process.

RETURN VALUES  Upon successful completion, 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  killpg() will fail and no signal will be sent if any of the following occur:

    EINVAL    sig 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), setpgid(2), sigaction(2), signal(5)

modified 12 Feb 1993
NAME
krb_realmofhost, krb_get_phost, krb_get_krbhst, krb_get_admhst, krb_get_lrealm – additional Kerberos utility routines

SYNOPSIS
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);

DESCRIPTION
krb_realmofhost() returns the Kerberos realm of the host host, as determined by the translation table /etc/krb.realms. host 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.

The format of the translation file is described by krb.realms(4). If host exactly matches a host_name line, the corresponding realm is returned. Otherwise, if the domain portion of host matches a domain_name line, the corresponding realm is returned. If host contains a domain, but no translation is found, host’s domain is converted to upper-case and returned. If host contains no discernible domain, or an error occurs, the local realm name, as supplied by krb_get_lrealm(), is returned.

krb_get_phost() converts the hostname alias (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.

krb_get_krbhst() fills in host with the hostname of the nth host running a Kerberos key distribution center (KDC) for realm realm, as specified in the configuration file /etc/krb.conf or krb.conf NIS map. The configuration format is described by krb.conf(4). If the host is successfully filled in, the routine returns KSUCCESS. If the file (or NIS map) cannot be accessed, and n equals 1, then the hostname kerberos is filled in, and KSUCCESS is returned. If there are fewer than n hosts running a Kerberos KDC for the requested realm, or the configuration file is malformed, the routine returns KFAILURE.

When there is both a local /etc/krb.conf and a krb.conf NIS map, then the entries are counted starting first with the local file, then continuing with the NIS map. For example, if the local /etc/krb.conf file contains two entries which match realm, and the NIS map contains one matching entry, then there are three possible matches that krb_get_krbhst() can return. The first two (for n values 1 and 2) come from the file, and the third (for n equal to 3) comes from the map.
**krb_realmofhost (3N)**

**krb_get_admhst**

fills in `host` with the hostname of the `n`th host running a Kerberos KDC database administration server for realm `realm`, as specified in `/etc/krb.conf`. If the file cannot be opened or is malformed, or there are fewer than `n` hosts running a Kerberos KDC database administration server, the routine returns `KFAILURE`.

The character arrays used as return values for `krb_get_krbhst()` and `krb_get_admhst()` should be large enough to hold any hostname.

**krb_get_lrealm()**

fills in `realm` with the `n`th realm of the local host, as specified in the configuration file. `realm` should be at least `REALM_SZ` (from `<kerberos/krb.h>`) characters long. The return value is either `KSUCCESS` or `KFAILURE`.

**ATTRIBUTES**

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

**SEE ALSO**

`kerberos(3N), krb.conf(4), krb.realms(4), attributes(5)`

**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.

modified 30 Dec 1996 SunOS 5.6 3N-879
NAME krb_sendauth, krb_recvauth, krb_net_write, krb_net_read – Kerberos routines for sending authentication via network stream sockets

SYNOPSIS 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_DAT *msg_data, CREDENTIALS *cred, Key_schedule schedule,
const struct sockaddr_in *laddr, const struct sockaddr_in *faddr,
const char *version);

int krb_recvauth(const long options, const int fd, KTEXT ktext, const char *service,
char *inst, const struct sockaddr_in *faddr, const struct sockaddr_in *laddr,
AUTH_DAT *auth_data, const char *filename, Key_schedule schedule,
char *version);

int krb_net_write(const int fd, const char *buf, const int len);
int krb_net_read(const int fd, char *buf, const int len);

DESCRIPTION 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.

The krb_sendauth() function sends an authenticated ticket from the client program to the server program by writing the ticket to a network socket.

The krb_recvauth() function receives the ticket from the client by reading from a network socket.

krb_sendauth() This function writes the ticket to the network socket specified by the file descriptor fd, returning KSUCCESS if the write proceeds successfully, and an error code if it does not.

The ktext argument should point to an allocated KTEXT_ST structure. The service, inst, and realm 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 realm argument to NULL.

The version argument allows the client program to pass an application-specific version string that the server program can then match against its own version string. The version string can be up to KSEND_VNO_LEN (see <kerberos/krb.h>) characters in length.

The checksum argument can be used to pass checksum information to the server program. The client program is responsible for specifying this information. This checksum information is difficult to corrupt because krb_sendauth() passes it over the network in encrypted form. The checksum argument is passed as the checksum argument to krb_mk_req() (see kerberos(3N)).
You can set `krb_sendauth()`’s other arguments to NULL unless you want the client and server programs to mutually authenticate themselves. In the case of mutual authentication, the client authenticates itself to the server program, and demands that the server in turn authenticate itself to the client.

### krb_sendauth() and Mutual Authentication

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.
The `filename` argument specifies the filename which the service program should use to obtain its service key. The `krb_recvauth()` function passes `filename` to the `krb_rd_req()` function, see `kerberos(3N)`. If you set this argument to "", `krb_rd_req()` looks for the service key in the file `/etc/srvtab`.

If the client and server are performing mutual authentication, the `schedule` argument should point to an allocated `Key_schedule`. Otherwise it is ignored and may be `NULL`.

The `version` argument should point to a character array of at least `KSEND_VNO_LEN` characters. It is filled in with the version string passed by the client to `krb_sendauth()`.

The `krb_net_write()` function emulates the `write(2)` system call, but guarantees that all data specified is written to `fd` before returning, unless an error condition occurs.

The `krb_net_read()` function emulates the `read(2)` system call, but guarantees that the requested amount of data is read from `fd` before returning, unless an error condition occurs.

**ATTRIBUTES**

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

**SEE ALSO**

`read(2)`, `write(2)`, `kerberos(3N)`, `kerberos_rpc(3N)`, `krb_realmofhost(3N)`, `attributes(5)`

**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    cc [ flag ...] file ... -lkrb [ library ...]
#include <kerberos/krb.h>
void krb_set_tkt_string(const char *filename);

DESCRIPTION    krb_set_tkt_string() sets the name of the file that holds the user’s cache of Kerberos
server tickets and associated session keys.

The string filename passed in is copied into local storage. Only MAXPATHLEN-1 (see
<sys/param.h>) characters of the filename are copied in for use as the cache file name.

This routine should be called during initialization, before other Kerberos routines are
called; otherwise the routines which fetch the ticket cache file name may be called and
return an undesired ticket file name until this routine is called.

FILES    /tmp/tktuid  default ticket file name, unless the environment variable
          KRBTKFILE is set. uid denotes the user’s uid, in decimal.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO    kerberos(3N), attributes(5)

NOTES    This interface is unsafe in multithreaded applications. Unsafe interfaces should be called
          only from the main thread.
NAME
kstat – kernel statistics facility

DESCRIPTION
The **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:

```c
typedef int 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 */
  u_int     ks_ndata; /* # of data records */
  size_t    ks_data_size; /* size of kstat data section */
  hrtime_t  ks_snaptime; /* time of last data snapshot */
  
  /* Fields relevant to kernel only */
  
  int      (*ks_update)(struct kstat *, int);
  void     *ks_private;
  int      (*ks_snapshot)(struct kstat *, void *, int);
  void     *ks_lock;
}
kstat_t;
```

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 (such as creation time, last snapshot time, **kstat_timer_t** and **kstat_io_t** timestamps, and the like) 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, such as disk, tape, net, vm, and streams. This field can be used as a filter to extract related kstats. The following values are currently in use: disk, tape, controller, net, rpc, vm, kvm, hat, streams, kmem, kmem_cache, kstat, and misc. (The kstat class encompasses things like kstat_types.)

ks_data, ksndata, ks_data_size  
ks_data is a pointer to the kstat’s data section. The type of data stored there depends on ks_type.

ksndata 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:

\[
\text{rate} = \frac{\text{new\_count} - \text{old\_count}}{\text{new\_snaptime} - \text{old\_snaptime}};
\]

kstat data types  The following types of kstats are currently available:

```c
#define KSTAT_TYPE_RAW 0 /* can be anything */
#define KSTAT_TYPE_NAMED 1 /* name/value pairs */
#define KSTAT_TYPE_INTR 2 /* interrupt statistics */
#define KSTAT_TYPE_IO 3 /* I/O statistics */
#define KSTAT_TYPE_TIMER 4 /* event timers */
```

To get a list of all kstat types currently supported in the system, tools can read out the standard system kstat kstat_types (full name spec is "<unix'"", 0, "kstat_types"). This is a KSTAT_TYPE_NAMED kstat in which the name field describes the type of kstat, and the value field is the kstat type number (for example, KSTAT_TYPE_IO is type 3 -- see above).

modified 7 Nov 1996  SunOS 5.6  3K-885
Raw kstat: KSTAT_TYPE_RAW: raw data
The "raw" kstat type is just treated as an array of bytes. This is generally used to export well-known structures, like sysinfo.

Name=value kstat: KSTAT_TYPE_NAMED: A list of arbitrary name=value statistics.

typedef struct kstat_named {
  char name[KSTAT_STRLEN]; /* name of counter */
  uchar_t data_type; /* data type */
  union {
    char c[16]; /* enough for 128-bit ints */
    int32_t i32;
    uint32_t ui32;
    int64_t i64;
    uint64_t ui64;
  } value; /* value of counter */
} kstat_named_t;

#define KSTAT_DATA_CHAR 0
#define KSTAT_DATA_INT32 1
#define KSTAT_DATA_UINT32 2
#define KSTAT_DATA_INT64 3
#define KSTAT_DATA_UINT64 4

/* These types are obsolete */

#define KSTAT_DATA_LONG 1
#define KSTAT_DATA_ULONG 2
#define KSTAT_DATA_LONGLONG 3
#define KSTAT_DATA_ULONGLONG 4
#define KSTAT_DATA_FLOAT 5
#define KSTAT_DATA_DOUBLE 6

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).

```c
#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
```

```c
typedef struct kstat_intr {
    uint_t intrs[KSTAT_NUM_INTRS]; /* interrupt counters */
} kstat_intr_t;
```

**Event timer kstat**

KSTAT_TYPE_TIMER: Event timer statistics.

These provide basic counting and timing information for any type of event.

```c
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.

```c
typedef struct kstat_io {
    /*
     * Basic counters.
     */
    u_longlong_t nread;            /* number of bytes read */
    u_longlong_t nwritten;         /* number of bytes written */
    uint_t        reads;           /* number of read operations */
    uint_t        writes;          /* number of write operations */
    /*
     * Accumulated time and queue length statistics.
     */
    /*
     * Time statistics are kept as a running sum of "active" time.
     * Queue length statistics are kept as a running sum of the
     * product of queue length and elapsed time at that length --
     * that is, a Riemann sum for queue length integrated against time.
     */
    /*
     *                ^
     *                |           
     */
} kstat_io_t;
```
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;   /* cumulative wait length*time product */
hrtime_t wlastupdate;  /* last time wait queue changed */
hrtime_t rtime;  /* cumulative run (service) time */
hrtime_t rlen;   /* cumulative run length*time product */
hrtime_t rlastupdate;  /* last time run queue changed */
uint_t wcnt;    /* count of elements in wait state */
uint_t rcnt;    /* count of elements in run state */
```
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:

```c
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:

```c
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)

modified 7 Nov 1996 SunOS 5.6 3K-889
**NAME**
kstat_chain_update – update the kstat header chain

**SYNOPSIS**
```
c [ flag ... ] file ... -lkstat [ library... ]
#include <kstat.h>
kid_t *kstat_chain_update(kstat_ctl_t *kc);
```

**DESCRIPTION**
kstat_chain_update() brings the user’s kstat header chain in sync with the kernel’s. The kstat chain is a linked list of kstat headers (kstat_t’s), pointed to by kc->kc_chain, which is initialized by kstat_open(3K). This chain constitutes a list of all kstats currently in the system. During normal operation, the kernel will occasionally create new kstats and delete old ones, as various device instances come and go. When this happens, the user’s copy of the kstat chain becomes out of date.

kstat_chain_update() detects this by comparing the kernel’s current kstat chain ID (KCID), which is incremented every time the kstat chain changes, to the user’s KCID, kc->kc_chain_id. If the KCID’s match, kstat_chain_update() does nothing. Otherwise, it deletes any invalid kstat headers from the user’s kstat chain, and adds any new ones, and sets kc->kc_chain_id to the new KCID. All other kstat headers in the user’s kstat chain are unmodified.

**RETURN VALUES**
kstat_chain_update() returns the new KCID if the kstat chain has changed, 0 if it hasn’t, or -1 on failure.

**FILES**
/dev/kstat kernel statistics driver

**SEE ALSO**
kstat(3K), kstat_close(3K), kstat_data_lookup(3K), kstat_lookup(3K), kstat_open(3K), kstat_read(3K), kstat_write(3K)
NAME
kstat_lookup, kstat_data_lookup – find a kstat by name

SYNOPSIS
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);

DESCRIPTION
kstat_lookup() walks down the kstat chain, kc->kc_chain, looking for a kstat with the same ks_module, ks_instance, and ks_name fields; this triplet uniquely identifies a kstat. If ks_module is NULL, ks_instance is -1, or ks_name 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".

kstat_data_lookup() searches the kstat’s data section for the record with the specified name. 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.

RETURN VALUES
kstat_lookup() returns a pointer to the requested kstat if it is found, or NULL if it isn’t.

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.

FILES
/dev/kstat kernel statistics driver

SEE ALSO
kstat(3K), kstat_chain_update(3K), kstat_close(3K), kstat_open(3K), kstat_read(3K), kstat_write(3K)

modified 26 May 1994 SunOS 5.6 3K-891
NAME  
kstat_open, kstat_close – initialize kernel statistics facility

SYNOPSIS  
cc [ flag . . . ] file . . . -lkstat [ library . . . ]
#include <kstat.h>
kstat_ctl_t *kstat_open(void);
int kstat_close(kstat_ctl_t *kc);

DESCRIPTION  
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 kc
argument in subsequent libkstat function calls.
kstat_close() frees all resources that were associated with kc. This is done automatically
on exit(2) and execve() (see exec(2)).

RETURN VALUES  
kstat_open() returns a pointer to a kstat control structure. On failure, it returns NULL
and no resources are allocated.
kstat_close() returns 0 on success, −1 on failure.

FILES  
/dev/kstat kernel statistics driver

SEE ALSO  
 kstat(3K), kstat_chain_update(3K), kstat_data_lookup(3K), kstat_lookup(3K),
           kstat_read(3K), kstat_write(3K)
NAME  kstat_read, kstat_write – read or write kstat data

SYNOPSIS  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);

DESCRIPTION  kstat_read() gets data from the kernel for the kstat pointed to by ksp. ksp->ks_data is automatically allocated (or reallocated) to be large enough to hold all of the data. ksp->ks_ndata is set to the number of data fields, ksp->ks_data_size is set to the total size of the data, and ksp->ks_snaptime is set to the high-resolution time at which the data snapshot was taken. If buf is non-NULL, the data is copied from ksp->ks_data into buf.

    kstat_write() writes data from buf, or from ksp->ks_data if buf is NULL, to the corresponding kstat in the kernel. Only the superuser can use kstat_write().

RETURN VALUES  On success, kstat_read() and kstat_write() return the current kstat chain ID (KCID). On failure, they return −1.

FILES  /dev/kstat  kernel statistics driver

SEE ALSO  kstat(3K), kstat_chain_update(3K), kstat_close(3K), kstat_data_lookup(3K), kstat_lookup(3K), kstat_open(3K)
NAME
kvm_getu, kvm_getcmd – get the u-area or invocation arguments for a process

SYNOPSIS
#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);

DESCRIPTION
kvm_getu() reads the u-area of the process specified by proc to an area of static storage associated with kd and returns a pointer to it. Subsequent calls to kvm_getu() will overwrite this static area.

kd is a pointer to a kernel descriptor returned by kvm_open(3K). proc is a pointer to a copy (in the current process’ address space) of a proc structure (obtained, for instance, by a prior kvm_nextproc(3K) call).

kvm_getcmd() constructs a list of string pointers that represent the command arguments and environment that were used to initiate the process specified by proc.

kd is a pointer to a kernel descriptor returned by kvm_open(3K). u is a pointer to a copy (in the current process’ address space) of a user structure (obtained, for instance, by a prior kvm_getu() call). If arg 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 arg. If env 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 env.

The pointers returned in arg and env 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.

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.

RETURN VALUES
On success, kvm_getu() returns a pointer to a copy of the u-area of the process specified by proc. On failure, it returns NULL.

kvm_getcmd() returns:
0 on success.
−1 on failure.
ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO kvm_nextproc(3K), kvm_open(3K), kvm_read(3K), malloc(3C), attributes(5)

NOTES
If kvm_getcmd() returns −1, the caller still has the option of using the command line fragment that is stored in the u-area.
### NAME
kvm_nextproc, kvm_getproc, kvm_setproc – read system process structures

### SYNOPSIS
```c
#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);
```

### DESCRIPTION
=kvm_nextproc()= may be used to sequentially read all of the system process structures from the kernel identified by =kd= (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.

For performance reasons, many implementations will cache a set of system process structures. Since the system state is liable to change between calls to =kvm_nextproc(),= and since the cache may contain obsolete information, there is no guarantee that =every= process structure returned refers to an active process, nor is it certain that =all= processes will be reported.

=kvm_setproc()= renews the process list, enabling =kvm_nextproc()= to re-scan 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.

=kvm_getproc()= locates the =proc= structure of the process specified by =pid= 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.

### RETURN VALUES
On success, =kvm_nextproc()= returns a pointer to a copy of the next valid process table entry. On failure, it returns NULL.

On success, =kvm_getproc()= returns a pointer to the =proc= structure of the process specified by =pid=. On failure, it returns NULL.

=kvm_setproc()= returns:

- 0 on success.
- −1 on failure.
ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO

kvm_getu(3K), kvm_open(3K), kvm_read(3K), attributes(5)
NAME  
kvm_nlist – get entries from kernel symbol table

SYNOPSIS  
#include <kvm.h>
#include <nlist.h>

int kvm_nlist(kvm_t *kd, struct nlist *nl);

DESCRIPTION  
kvm_nlist() examines the symbol table from the kernel image identified by kd (see
kvm_open(3K)) and selectively extracts a list of values and puts them in the array of nlist
structures pointed to by nl. The name list pointed to by nl consists of an array of struc-
tures 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 ter-
ninated by an entry with a NULL pointer (or a pointer to a null string) in the n_name
field. For each entry in nl, if the named symbol is present in the kernel 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 nl is set to zero.

RETURN VALUES  
kvm_nlist() returns the value of nlist(3B) or nlist(3E), depending on the library used.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO  
nlist(3B), nlist(3E), kvm_open(3K), kvm_read(3K), attributes(5)
NAME
kvm_open, kvm_close – specify a kernel to examine

SYNOPSIS
#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);

DESCRIPTION
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 kd argument in subsequent kernel VM function calls.

The namelist argument specifies an unstripped executable file whose symbol table will be used to locate various offsets in corefile. If namelist 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 /dev/ksyms as a default namelist file).

corefile specifies a file that contains an image of physical memory, for instance, a kernel crash dump file (see savecore(1M)) or the special device /dev/mem. If corefile is NULL, the currently running kernel is accessed (using /dev/mem and /dev/kmem).

swapfile specifies a file that represents the swap device. If both corefile and swapfile are NULL, the swap device of the “currently running kernel” is accessed. Otherwise, if swapfile is NULL, kvm_open() may succeed but subsequent kvm_getu(3K) function calls may fail if the desired information is swapped out.

flag is used to specify read or write access for corefile and may have one of the following values:

O_RDONLY open for reading
O_RDWR open for reading and writing

errstr 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 errstr would be argv[0]. This has the effect of printing the process name in front of any error messages.

kvm_close() closes all file descriptors that were associated with kd. These files are also closed on exit(2) and execve() (see exec(2)). kvm_close() also resets the proc pointer associated with kvm_nextproc(3K) and flushes any cached kernel data.

RETURN VALUES
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.

kvm_close() returns:
0 on success.
−1 on failure.
FILES
/dev/kmem
/dev/ksyms
/dev/mem

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
savecore(1M), exec(2), exit(2), kvm_getu(3K), kvm_nextproc(3K), kvm_nlist(3K),
kvm_read(3K), attributes(5)

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, kvm_uread, kvm_uwrite, kvm_kread, kvm_kwrite – copy data to or from a kernel image or running system

SYNOPSIS  
```
#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);
int kvm_uread(kvm_t *kd, unsigned long addr, char *buf, unsigned nbytes);
int kvm_uwrite(kvm_t *kd, unsigned long addr, char *buf, unsigned nbytes);
int kvm_kread(kvm_t *kd, unsigned long addr, char *buf, unsigned nbytes);
int kvm_kwrite(kvm_t *kd, unsigned long addr, char *buf, unsigned nbytes);
```

DESCRIPTION  

kvm_read() transfers data from the kernel image specified by kd (see kvm_open(3K)) to the address space of the process. nbytes bytes of data are copied from the kernel virtual address given by addr to the buffer pointed to by buf.

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 kd must have specified write access. If a user virtual address is given, it is resolved in the address space of the process specified in the most recent kvm_getu(3K) call.

kvm_uread() transfers data from the address space of the processes specified in the most recent kvm_getu(3K) call. nbytes bytes of data are copied from the user virtual address given by addr to the buffer pointed to by buf.

kvm_uwrite() is like kvm_uread(), except that the direction of the transfer is reversed. In order to use this function, the kvm_open(3K) call that returned kd must have specified write access. The address is resolved in the address space of the process specified in the most recent kvm_getu(3K) call.

kvm_kread() transfers data from the kernel address space to the address space of the process. nbytes bytes of data are copied from the kernel virtual address given by addr to the buffer pointed to by buf.

kvm_kwrite() is like kvm_kread(), except that the direction of the transfer is reversed. In order to use this function, the kvm_open(3K) call that returned kd must have specified write access.

Note: The use of kvm_uread(), kvm_uwrite(), kvm_kread() and kvm_kwrite() is encouraged over the use of kvm_read() and kvm_write() since these are more clearly defined interfaces.

RETURN VALUES  

All the above functions return the following values:
```
<number of bytes actually transferred>  Success.
-1  Failure.
```

modified 29 Dec 1996  SunOS 5.6  3K-901
ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO

kvm_getu(3K), kvm_nlist(3K), kvm_open(3K), attributes(5)

3K-902

SunOS 5.6

modified 29 Dec 1996
NAME  lckpwdf, ulckpwdf – manipulate shadow password database lock file

SYNOPSIS  #include <shadow.h>
            int lckpwdf(void);
            int ulckpwdf(void);

DESCRIPTION  lckpwdf() and ulckpwdf() are routines that are used to gain modification access to the password databases, through the lock file. A process first uses lckpwdf() 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 ulckpwdf(). 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.

lckpwdf() 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.

ulckpwdf() 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  lckpwdf() and ulckpwdf() return −1 on failure, and 0 otherwise.

FILES  /etc/shadow
       /etc/passwd
       /etc/.pwd.lock

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  getpwnam(3C), getspnam(3C), attributes(5)

NOTES  These routines are for internal use only; compatibility is not guaranteed.
NAME 
ldexp – load exponent of a floating point number

SYNOPSIS 
#include <math.h>
double ldexp(double x, int exp);

DESCRIPTION 
The ldexp() function computes the quantity \( x \times 2^{\exp} \).

RETURN VALUES 
Upon successful completion, ldexp() returns a double representing the value \( x \times 2^{\exp} \).
If the value of \( x \) is NaN, NaN is returned.
If ldexp() would cause overflow, ±HUGE_VAL is returned (according to the sign of \( x \)), and errno is set to ERANGE.
If ldexp() would cause underflow to 0.0, 0 is returned and errno may be set to ERANGE.

ERRORS 
The ldexp() function will fail if:
ERANGE The value to be returned would have caused overflow.
The ldexp() function may fail if:
ERANGE The value to be returned would have caused underflow.

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

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO 
frexp(3C), isnan(3M), attributes(5)
NAME
lfmt – display error message in standard format and pass to logging and monitoring services

SYNOPSIS
#include <pfmt.h>
int lfmt(FILE *stream, long flags, char *format, .../* arg */);  

DESCRIPTION
lfmt() retrieves a format string from a locale-specific message database (unless MM_NOGET is specified) and uses it for printf() style formatting of args. The output is displayed on stream. If stream is NULL no output is displayed.
lfmt() encapsulates the output in the standard error message format (unless MM_NOSTD is specified, in which case the output is simply printf() like).
lfmt() forwards its output to the logging and monitoring facility, even if stream is NULL. Optionally, lfmt() will display the output on the console, with a date and time stamp.
lfmt() for the printf() format string is to be retrieved from a message database, the format argument must have the following structure:
<catalog>:<msgnum>:<defmsg>.
lfmt() will output Message not found!!\n as format string if <catalog> is not a valid catalog name, if no catalog is specified (either explicitly or via setcat()), if <msgnum> is not a valid number, or if no message could be retrieved from the message databases, and <defmsg> was omitted.
The flags determine the type of output (i.e. whether the format should be interpreted as is or encapsulated in the standard message format), and the access to message catalogs to retrieve a localized version of format.
The flags are composed of several groups, and can take the following values (one from each group):
Output format control

modified 29 Dec 1996

SunOS 5.6

3C-905
<table>
<thead>
<tr>
<th>Catalog access control</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>MM_NOP</td>
<td>Do not retrieve a localized version of <em>format</em>. In this case, only the <code>&lt;defmsg&gt;</code> part of the <em>format</em> is specified.</td>
</tr>
<tr>
<td>MM_GET</td>
<td>Retrieve a localized version of <em>format</em>, from the <code>&lt;catalog&gt;</code>, using <code>&lt;msgid&gt;</code> as the index and <code>&lt;defmsg&gt;</code> as the default message (default, value 0).</td>
</tr>
<tr>
<td>Severity (standard message format only)</td>
<td></td>
</tr>
<tr>
<td>MM_HALT</td>
<td>generates a localized version of <strong>HALT</strong>, but does not halt the machine.</td>
</tr>
<tr>
<td>MM_ERROR</td>
<td>generates a localized version of <strong>ERROR</strong> (default, value 0).</td>
</tr>
<tr>
<td>MM_WARNING</td>
<td>generates a localized version of <strong>WARNING</strong>.</td>
</tr>
<tr>
<td>MM_INFO</td>
<td>generates a localized version of <strong>INFO</strong>.</td>
</tr>
<tr>
<td>Additional severities can be defined. Add-on severities can be defined with number-string pairs with numeric values from the range [5-255], using <code>addsev()</code>. The numeric value ORed with other <em>flags</em> will generate the specified severity.</td>
<td></td>
</tr>
<tr>
<td>If the severity is not defined, <code>lfmt()</code> used the string <strong>SEV=N</strong> where <em>N</em> is replaced by the integer severity value passed in <em>flags</em>.</td>
<td></td>
</tr>
<tr>
<td>Multiple severities passed in <em>flags</em> 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 <strong>SEV=N</strong> (if undefined).</td>
<td></td>
</tr>
<tr>
<td>Action</td>
<td></td>
</tr>
<tr>
<td>MM_ACTION</td>
<td>specifies an action message. Any severity value is superseded and replaced by a localized version of <strong>TO FIX</strong>.</td>
</tr>
<tr>
<td>Console display control</td>
<td></td>
</tr>
<tr>
<td>MM_CONSOLE</td>
<td>display the message to the console in addition to the specified stream.</td>
</tr>
<tr>
<td>MM_NOCONSOLE</td>
<td>do not display the message to the console in addition to the specified stream (default, value 0).</td>
</tr>
</tbody>
</table>
Logging information

Major classification

Identifies the source of the condition. Identifiers are:
- **MM_HARD** (hardware)
- **MM_SOFT** (software)
- **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)
- **MM_OPSYS** (operating system).

STANDARD ERROR MESSAGE FORMAT

If `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.
ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-safe</td>
</tr>
</tbody>
</table>

SEE ALSO

addsev(3C), gettext(3C), pfmt(3C), printf(3S), setcat(3C), setlabel(3C), setlocale(3C), attributes(5), environ(5)
NAME
lgamma, lgamma_r, gamma, gamma_r – log gamma function

SYNOPSIS
cc [ flag ...] file ... -lm [ library ...]
#include <math.h>
extern int signgam;
double lgamma(double x);
double lgamma_r(double x, int *signgamp);

DESCRIPTION
Both lgamma() and lgamma_r() return

\ln |\Gamma(x)|

where

\Gamma(x) = \int_0^\infty t^{x-1}e^{-t}dt

for x > 0 and

\Gamma(x) = \pi/(\Gamma(1-x) \sin(\pi x))

for x < 1.

lgamma() uses the external integer signgam to return the sign of \Gamma(x) while lgamma_r() uses the user-allocated space addressed by signgamp.

IDIOSYNCRASIES
In the case of lgamma(), do not use the expression signgam*exp(lgamma(x)) to compute ‘g := \Gamma(x)’. Instead compute lgamma() first:

lg = lgamma(x); g = signgam*exp(lg);

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.

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

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>See NOTES below.</td>
</tr>
</tbody>
</table>

SEE ALSO
matherr(3M), attributes(5)

NOTES
Although lgamma_r() is not mentioned by POSIX.4a Draft 6, it was added to complete the functionality provided by similar thread-safe functions. This interface is subject to change to be compatible with the "spirit" of POSIX.4a when it is approved as a standard.

When compiling multi-thread applications, the _REENTRANT flag must be defined on the compile line. This flag should only be used in multi-thread applications.
The `lgamma()` function is unsafe in multithreaded applications. `lgamma_r()` should be used instead.
NAME
libthread_db – library of interfaces for monitoring and manipulating threads-related aspects of multithreaded programs

SYNOPSIS
cc [ flag ... ] file ... /lib/libthread_db.so.1 [ library ... ]
#include <proc_service.h>
#include <thread_db.h>
void td_event_addset(td_thr_events_t *events, td_thr_events_e n);
void td_event_delset(td_thr_events_t *events, td_thr_events_e n);
void td_event_emptyset(td_thr_events_t *events);
void td_event_fillset(td_thr_events_t *events);
void td_eventisempty(td_thr_events_t *events);
void td_eventismember(td_thr_events_t *events, td_thr_events_e n);
td_err_e td_init();
void td_log();
td_err_e td_sync_get_info(const td_synchandle_t *sh_p, td_syncinfo_t *si_p);
td_err_e td_sync_setstate(const td_synchandle_t *sh_p, int value);
td_err_e td_sync_waiters(const td_synchandle_t *sh_p, td_thr_iter_f *cb, void *cb_data_p);
td_err_e td_thr_clear_event(const td_thrhandle_t *th_p, td_thr_events_t *events);
td_err_e td_ta_delete(const td_thragent_t *ta_p);
td_err_e td_ta_enable_stats(const td_thragent_t *ta_p, int on_off);
td_err_e td_ta_event_addr(const td_thragent_t *ta_p, u_long event, td_notify_t *notify_p);
td_err_e td_ta_event_getmsg(const td_thragent_t *ta_p, td_event_msg_t *msg);
td_err_e td_ta_get_nthreads(const td_thragent_t *ta_p, int *nthread_p);
td_err_e td_ta_get_ph(const td_thragent_t *ta_p, struct ps_prochandle **ph_pp);
td_err_e td_ta_get_stats(const td_thragent_t *ta_p, td_ta_stats_t *tstats);
td_err_e td_ta_map_addr2sync(const td_thragent_t *ta_p, psaddr_t addr td_synchandle_t *sh_p);
td_err_e td_ta_map_id2thr(const td_thragent_t *ta_p, thread_t tid, td_thrhandle_t *th_p);
td_err_e td_ta_map_lwp2thr(const td_thragent_t *ta_p, lwpid_t lwpid, td_thrhandle_t *th_p);
DESCRIPTION  libthread_db is a library that provides support for monitoring and manipulating threads-related aspects of a multithreaded program. There are at least two processes involved, the controlling process and one or more target processes. The controlling process is the libthread_db client, which links with libthread_db and uses libthread_db to inspect or modify threads-related aspects of one or more target processes. The target processes must be multithreaded processes that use libthread or libpthread. The
controlling process may or may not be multithreaded itself.

The most commonly anticipated use for libthread_db is that the controlling process will be a debugger for a multithreaded program, hence the "db" in libthread_db.

libthread_db is dependent on the internal implementation details of libthread. It is a "friend" of libthread in the C++ sense, which is precisely the "value added" by libthread_db. It encapsulates the knowledge of libthread internals that a debugger needs in order to manipulate the threads-related state of a target process.

To be able to inspect and manipulate target processes, libthread_db makes use of certain process control primitives that must be provided by the process using libthread_db. The imported interfaces are defined in proc_service(3T). In other words, the controlling process is linked with libthread_db, and it calls routines in libthread_db. libthread_db in turn calls certain routines that it expects the controlling process to provide. These process control primitives allow libthread_db to:

- Look up symbols in a target process.
- Stop and continue individual lightweight processes (LWPs) within a target process.
- Stop and continue an entire target process.
- Read and write memory and registers in a target process.

Initially, a controlling process obtains a handle for a target process. Through that handle it can then obtain handles for the component objects of the target process, its threads, its synchronization objects, and its thread-specific-data keys.

When libthread_db needs to return sets of handles to the controlling process, for example, when returning handles for all the threads in a target process, it uses an iterator function. An iterator function calls back a client-specified function once for each handle to be returned, passing one handle back on each call to the callback function. The calling function also passes another parameter to the iterator function, which the iterator function passes on to the callback function. This makes it easy to build a linked list of thread handles for a particular target process. The additional parameter is the head of the linked list, and the callback function simply inserts the current handle into the linked list.

Callback functions are expected to return an integer. Iteration terminates early if a callback function returns a non-zero value. Otherwise, iteration terminates when there are no more handles to pass back.

libthread_db relies on an "agent thread" in the target process for some of its operations. The "agent thread" is a system thread started when libthread_db attaches to a process through td_ta_new(3T). In the current implementation, a brief window exists after the agent thread has been started, but before it has completed its initialization, in which libthread_db routines that require the agent thread will fail, returning a TD_NOCAPAB error status. This is particularly troublesome if the target process was stopped when td_ta_new() was called, so that the agent thread cannot be initialized. To avoid this problem, the target process must be allowed to make some forward progress after td_ta_new() is called. This limitation will be removed in a future release.
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>td_event_addset()</td>
<td>Macro that adds a specific event type to an event set.</td>
</tr>
<tr>
<td>td_event_delset()</td>
<td>Macro that deletes a specific event type from an event set.</td>
</tr>
<tr>
<td>td_event_emptyset()</td>
<td>Macro that sets argument to NULL event set.</td>
</tr>
<tr>
<td>td_event_fillset()</td>
<td>Macro that sets argument to set of all events.</td>
</tr>
<tr>
<td>td_eventisempty()</td>
<td>Macro that tests whether an event set is the NULL set.</td>
</tr>
<tr>
<td>td_eventismember()</td>
<td>Macro that tests whether a specific event type is a member of an event set.</td>
</tr>
<tr>
<td>td_init()</td>
<td>Performs initialization for interfaces.</td>
</tr>
<tr>
<td>td_log()</td>
<td>Placeholder for future logging functionality.</td>
</tr>
<tr>
<td>td_sync_get_info()</td>
<td>Gets information for the synchronization object.</td>
</tr>
<tr>
<td>td_sync_setstate()</td>
<td>Sets the state of the synchronization object.</td>
</tr>
<tr>
<td>td_sync_waiters()</td>
<td>Iteration function used for return of synchronization object handles.</td>
</tr>
<tr>
<td>td_ta_clear_event()</td>
<td>Clears a set of event types in the process event mask.</td>
</tr>
<tr>
<td>td_ta_delete()</td>
<td>Deregisters target process and deallocates internal process handle.</td>
</tr>
<tr>
<td>td_ta_enable_stats()</td>
<td>Turns statistics gathering on or off for the target process.</td>
</tr>
<tr>
<td>td_ta_event_addr()</td>
<td>Returns event reporting address.</td>
</tr>
<tr>
<td>td_ta_event_getmsg()</td>
<td>Returns process event message.</td>
</tr>
<tr>
<td>td_ta_get_nthreads()</td>
<td>Gets the total number of threads in a process. .</td>
</tr>
<tr>
<td>td_ta_get_ph()</td>
<td>Returns corresponding external process handle.</td>
</tr>
<tr>
<td>td_ta_get_stats()</td>
<td>Gets statistics gathered for the target process.</td>
</tr>
<tr>
<td>td_ta_map_addr2sync()</td>
<td>Gets a synchronization object handles from a synchronization object’s address.</td>
</tr>
<tr>
<td>td_ta_map_id2thr()</td>
<td>Returns a thread handle for the given thread id.</td>
</tr>
<tr>
<td>td_ta_map_lwp2thr()</td>
<td>Returns a thread handle for the given LWP id.</td>
</tr>
<tr>
<td>td_ta_new()</td>
<td>Registers target process and allocates internal process handle.</td>
</tr>
<tr>
<td>td_ta_reset_stats()</td>
<td>Resets all counters for statistics gathering for the target process.</td>
</tr>
<tr>
<td>td_ta_setconcurrency()</td>
<td>Sets concurrency level for target process.</td>
</tr>
<tr>
<td>td_ta_set_event()</td>
<td>Sets a set of event types in the process event mask.</td>
</tr>
<tr>
<td>td_ta_sync_iter()</td>
<td>Returns handles of synchronization objects associated</td>
</tr>
</tbody>
</table>
Threads Library libthread_db (3T)

```
Threads Library

with a process.

td_ta_thr_iter() Returns handles for threads that are part of the target process.
td_ta_tsd_iter() Returns the thread-specific data keys in use by the current process.
td_thr_clear_event() Clears a set of event types in the threads event mask.
td_thr_dbresume() Resumes thread.
td_thr_dbsuspend() Suspends thread.
td_thr_event_enable() Enables or disables event reporting.
td_thr_event_getmsg() Returns a process event message.
td_thr_get_info() Gets thread information and updates.
td_thr_getfpregs() Gets the floating point registers for the given thread.
td_thr_getgregs() Gets the general registers for a given thread.
td_thr_getxregs() Gets the extra registers for the given thread.
td_thr_getxregsize() Gets the size of the extra register set for the given thread.
td_thr_lockowner() Iterates over the set of locks owned by a thread. 

struct.
td_thr_set_event() Sets a set of event types in the threads event mask.
td_thr_setfpregs() Sets the floating point registers for the given thread. 

ti_sigmask

td_thr_setgregs() Sets the general registers for a given thread.
td_thr_setprio() Sets the priority of a thread.
td_thr_setsigpending() Changes a thread's pending signal state.
td_thr_setxregs() Sets the extra registers for the given thread.
td_thr_sigsetmask() Sets the signal mask of the thread.
td_thr_sleepinfo() Returns the synchronization handle for the object on which a thread is blocked.
td_thr_tsd() Gets a thread's thread-specific data.
td_thr_validate() Tests a thread handle for validity.

FILES /usr/lib/libthread_db.so.1

modified 3 Feb 1997 SunOS 5.6 3T-915
```
### ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

### SEE ALSO

libthread(3T), proc_service(3T), td_event_addset(3T), td_event_delset(3T),
td_event_emptyset(3T), td_event_fillset(3T), td_eventisempty(3T),

See also:

- libthread(3T), proc_service(3T), td_event_addset(3T), td_event_delset(3T),
  td_event_emptyset(3T), td_event_fillset(3T), td_eventisempty(3T),
  td_eventismember(3T), td_init(3T), td_log(3T), td_sync_get_info(3T),
  td_sync_waiters(3T), td_ta_delete(3T), td_ta_enable_stats(3T), td_ta_event_addr(3T),
  td_ta_event_getmsg(3T), td_ta_get_nthreads(3T), td_ta_get_ph(3T), td_ta_get_stats(3T),
  td_ta_map_addr2sync(3T), td_ta_map_id2thr(3T), td_ta_map_lwp2thr(3T),
  td_ta_new(3T), td_ta_reset_stats(3T), td_ta_set_event(3T), td_ta_setconcurrency(3T),
  td_ta_sync_iter(3T), td_ta_thr_iter(3T), td_ta_tsd_iter(3T), td_thr_clear_event(3T),
  td_thr_dbresume(3T), td_thr_dbsuspend(3T), td_thr_event_enable(3T),
  td_thr_event_getmsg(3T), td_thr_get_info(3T), td_thr_getfpregs(3T),
  td_thr_getxregs(3T), td_thr_getxregsize(3T), td_thr_lockowner(3T),
  td_thr_set_event(3T), td_thr_setfpregs(3T), td_thr_setgregs(3T), td_thr_setprio(3T),
  td_thr_setsigmask(3T), td_thr_setsigpending(3T), td_thr_setsigpending(3T),
  td_thr_sleepinfo(3T), td_thr_tsd(3T), td_thr_validate(3T), thr_getspecific(3T), lib-
  thread(4), libthread_db(4), attributes(5)
NAME
libtnfctl – library for TNF probe control in a process or the kernel

SYNOPSIS
#include <tnf/tnfctl.h>
c [ flag ... ] file ... -ltnfctl [ library ... ]

DESCRIPTION
libtnfctl is a library that provides an API to control TNF ("Trace Normal Form") probes within a process or the kernel. See tracing(3X) for an overview of the Solaris tracing architecture. The client of libtnfctl controls probes in one of four modes:

- **internal mode**: The target is the controlling process itself; that is, the client controls its own probes.
- **direct mode**: The target is a separate process; a client can either exec(2) a program or attach to a running process for probe control. libtnfctl uses proc(4) on the target process for probe and process control in this mode, and additionally provides basic process control features.
- **indirect mode**: The target is a separate process, but the controlling process is already using proc(4) to control the target, and hence libtnfctl cannot use those interfaces directly. Use this mode to control probes from within a debugger. In this mode, the client must provide a set of functions that libtnfctl can use to query and update the target process.
- **kernel mode**: The target is the Solaris kernel.

A process is controlled "externally" if it is being controlled in either direct mode or indirect mode. Alternatively, a process is controlled "internally" when it uses internal mode to control its own probes.

There can be only one client at a time doing probe control on a given process. Therefore, it is not possible for a process to be controlled internally while it is being controlled externally. It is also not possible to have a process controlled by multiple external processes. Similarly, there can be only one process at a time doing kernel probe control. Note, however, that while a given target may only be controlled by one libtnfctl client, a single client may control an arbitrary number of targets. That is, it is possible for a process to simultaneously control its own probes, probes in other processes, and probes in the kernel.

The following tables denote the modes applicable to all libtnfctl interfaces (INT = internal mode; D = direct mode; IND = indirect mode; K = kernel mode).

These interfaces create handles in the specified modes:

<table>
<thead>
<tr>
<th>Function</th>
<th>Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>tnfctl_internal_open()</td>
<td>INT</td>
</tr>
<tr>
<td>tnfctl_exec_open()</td>
<td>D</td>
</tr>
<tr>
<td>tnfctl_pid_open()</td>
<td>D</td>
</tr>
<tr>
<td>tnfctl_indirect_open()</td>
<td>IND</td>
</tr>
<tr>
<td>tnfctl_kernel_open()</td>
<td>K</td>
</tr>
</tbody>
</table>
These interfaces are used with the specified modes:

<table>
<thead>
<tr>
<th>Function</th>
<th>Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>tnfctl_continue()</td>
<td>D</td>
</tr>
<tr>
<td>tnfctl_probe_connect()</td>
<td>INT</td>
</tr>
<tr>
<td>tnfctl_trace_actrs_get()</td>
<td>INT</td>
</tr>
<tr>
<td>tnfctl_buffer_alloc()</td>
<td>INT</td>
</tr>
<tr>
<td>tnfctl_register_funcs()</td>
<td>INT</td>
</tr>
<tr>
<td>tnfctl_probe_apply()</td>
<td>INT</td>
</tr>
<tr>
<td>tnfctl_probe_apply_ids()</td>
<td>INT</td>
</tr>
<tr>
<td>tnfctl_probe_state_get()</td>
<td>INT</td>
</tr>
<tr>
<td>tnfctl_probe_enable()</td>
<td>INT</td>
</tr>
<tr>
<td>tnfctl_probe_disable()</td>
<td>INT</td>
</tr>
<tr>
<td>tnfctl_probe_trace()</td>
<td>INT</td>
</tr>
<tr>
<td>tnfctl_probe_untrace()</td>
<td>INT</td>
</tr>
<tr>
<td>tnfctl_check_libs()</td>
<td>INT</td>
</tr>
<tr>
<td>tnfctl_close()</td>
<td>INT</td>
</tr>
<tr>
<td>tnfctl_strerror()</td>
<td>INT</td>
</tr>
<tr>
<td>tnfctl_buffer_dealloc()</td>
<td>K</td>
</tr>
<tr>
<td>tnfctl_trace_state_set()</td>
<td>K</td>
</tr>
<tr>
<td>tnfctl_filter_state_set()</td>
<td>K</td>
</tr>
<tr>
<td>tnfctl_filter_list_get()</td>
<td>K</td>
</tr>
<tr>
<td>tnfctl_filter_list_add()</td>
<td>K</td>
</tr>
<tr>
<td>tnfctl_filter_list_delete()</td>
<td>K</td>
</tr>
</tbody>
</table>

When using libtnfctl, the first task is to create a handle for controlling probes. Function tnfctl_internal_open() creates an internal mode handle for controlling probes in the same process, as described above. Functions tnfctl_pid_open() and tnfctl_exec_open() create handles in direct mode. tnfctl_indirect_open() creates an indirect mode handle, and tnfctl_kernel_open() creates a kernel mode handle. A handle is required for use in nearly all other libtnfctl functions. tnfctl_close() releases the resources associated with a handle.

tnfctl_continue() is used in direct mode to resume execution of the target process.

tnfctl_buffer_alloc() allocates a trace file or, in kernel mode, a trace buffer.

tnfctl_probe_apply() and tnfctl_probe_apply_ids() call a specified function for each probe or for a designated set of probes.

tnfctl_register_funcs() registers functions to be called whenever new probes are seen or probes have disappeared, providing an opportunity to do one-time processing for each probe.
tnfctl_check_libs() is used primarily in indirect mode to check whether any new probes have appeared, that is, they have been made available by `dlopen(3X)`, or have disappeared, that is, they have disassociated from the process by `dlclose(3X)`.

`tnfctl_probe_enable()` and `tnfctl_probe_disable()` control whether the probe, when hit, will be ignored.

`tnfctl_probe_trace()` and `tnfctl_probe_untrace()` control whether an enabled probe, when hit, will cause an entry to be made in the trace file.

`tnfctl_probe_connect()` and `tnfctl_probe_disconnect_all()` control which functions, if any, are called when an enabled probe is hit.

`tnfctl_probe_state_get()` returns information about the status of a probe, such as whether it is currently enabled.

`tnfctl_trace_attrs_get()` returns information about the tracing session, such as the size of the trace buffer or trace file.

`tnfctl_strerror()` maps a `tnfctl` error code to a string, for reporting purposes.

The remaining interfaces apply only to kernel mode.

`tnfctl_trace_state_set()` controls the master switch for kernel tracing. See `prex(1)` for more details.

`tnfctl_filter_state_set()`, `tnfctl_filter_list_get()`, `tnfctl_filter_list_add()`, and `tnfctl_filter_list_delete()` allow a set of processes to be specified for which probes will not be ignored when hit. This prevents kernel activity caused by uninteresting processes from cluttering up the kernel’s trace buffer.

`tnfctl_buffer_dealloc()` deallocates the kernel’s internal trace buffer.

**RETURN VALUES**

`TNFCTL_ERR_NONE` is returned upon success.

**ERRORS**

The error codes for `libtnfctl` are:

- `TNFCTL_ERR_ACCES` - Permission denied.
- `TNFCTL_ERR_NOTARGET` - The target process completed.
- `TNFCTL_ERR_ALLOCFAIL` - A memory allocation failure occurred.
- `TNFCTL_ERR_INTERNAL` - An internal error occurred.
- `TNFCTL_ERR_SIZETOOSMALL` - The requested trace size is too small.
- `TNFCTL_ERR_SIZETOOBIG` - The requested trace size is too big.
- `TNFCTL_ERR_BADARG` - Bad input argument.
- `TNFCTL_ERR_NOTDYNAMIC` - The target is not a dynamic executable.
- `TNFCTL_ERR_NOLIBTNFPROBE` - `libtnfprobe.so` not linked in target.
- `TNFCTL_ERR_BUF BROKEN` - Tracing is broken in the target.
- `TNFCTL_ERR_BUF EXISTS` - A buffer already exists.
- `TNFCTL_ERR_NOBUF` - No buffer exists.
- `TNFCTL_ERR_BADDEALLOC` - Cannot deallocate buffer.

modified 4 Mar 1997 SunOS 5.6 3X-919
TNFCTL_ERR_NOPROCESS  No such target process exists.
TNFCTL_ERR_FILENOTFOUND File not found.
TNFCTL_ERR_BUSY  Cannot attach to process or kernel because it is already tracing.
TNFCTL_ERR_INVALIDPROBE Probe no longer valid.
TNFCTL_ERR_USR1  Error code reserved for user.
TNFCTL_ERR_USR2  Error code reserved for user.
TNFCTL_ERR_USR3  Error code reserved for user.
TNFCTL_ERR_USR4  Error code reserved for user.
TNFCTL_ERR_USR5  Error code reserved for user.

ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Availability</td>
<td>SUNWtnfc</td>
</tr>
<tr>
<td>MT Level</td>
<td>MT-Safe with exceptions</td>
</tr>
</tbody>
</table>

SEE ALSO

prex(1), exec(2), dlclose(3X), dlopen(3X), TNF_PROBE(3X), tnfctl_buffer_alloc(3X), tnfctl_buffer_dealloc(3X), tnfctl_check_libs(3X), tnfctl_close(3X), tnfctl_continue(3X), tnfctl_internal_open(3X), tnfctl_exec_open(3X), tnfctl_filter_list_add(3X), tnfctl_filter_list_delete(3X), tnfctl_filter_list_get(3X), tnfctl_filter_state_set(3X), tnfctl_kernel_open(3X), tnfctl_pid_open(3X), tnfctl_probe_apply(3X), tnfctl_probe_apply_ids(3X), tnfctl_probe_connect(3X), tnfctl_probe_disable(3X), tnfctl_probe_enable(3X), tnfctl_probe_state_get(3X), tnfctl_probe_trace(3X), tnfctl_probe_untrace(3X), tnfctl_indirect_open(3X), tnfctl_unregister_funcs(3X), tnfctl_strerror(3X), tnfctl_trace_attrs_get(3X), tnfctl_trace_state_set(3X), libtnfctl(4), proc(4), attributes(5)

Programming Utilities Guide
Linker and Libraries Guide

NOTES

This API is MT-Safe. Multiple threads may concurrently operate on independent tnfctl handles, which is the typical behavior expected. libtnfctl does not support multiple threads operating on the same tnfctl handle. If this is desired, it is the client’s responsibility to implement locking to ensure that two threads that use the same tnfctl handle are not simultaneously in a libtnfctl interface.
NAME    lio_listio – list directed I/O

SYNOPSIS    cc [ flag ...] file ... -lposix4 [ library ...]
#include <aio.h>

int lio_listio(int mode, struct aiocb const list[], int nent, struct sigevent *sig);

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 */
};

DESCRIPTION    The lio_listio() function allows the calling process, LWP, or thread, to initiate a list of I/O requests within a single function call.

If mode is set to LIO_WAIT, lio_listio() behaves synchronously, waiting until all I/O is completed, and the sig argument is ignored. If mode is set to LIO_NOWAIT, lio_listio() behaves asynchronously, returning immediately, and signal delivery will occur, according to the sig argument, when all the I/O operations from this function complete. If sig is NULL, or the sigev_signo member of the sigevent structure referenced by sig is zero, then no signal delivery will occur. Otherwise, the signal number indicated by sigev_signo will be delivered when all the requests in list have completed.

list is an array of pointers to aiocb structures. This array consists of nent elements. The array may contain null pointers, which will be ignored.

The aio_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).

modified 30 Dec 1996    SunOS 5.6    3R-921
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_NOWAIT, 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)).

For regular files, no data transfer will occur past the offset maximum established in the open file description associated with *aiocbp-* > aio_fildes.

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 *aiocbp-* > 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 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**

The lio_listio() function will fail if:

**EAGAIN**

The resources necessary to queue all the I/O requests were not available. The error status for each request is recorded in the 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 *nent* 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; \texttt{aio\_return\(3R\)} to determine if any request has completed; or \texttt{aio\_error\(3R\)} to determine if any request was canceled.

**EIO** One or more of the individual I/O operations failed. Using \texttt{aio\_error\(3R\)} with each \texttt{aiocb} structure will determine the individual request(s) that failed.

**ENOSYS** \texttt{lio\_listio\() is not supported by this implementation.

If either \texttt{lio\_listio\()} succeeds in queuing all of its requests, or \texttt{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 \texttt{aiocb} structure contains errors specific to the \texttt{read\(2\)} or \texttt{write\(2\)} function being performed:

**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 \texttt{aio\_cancel\(3R\)} request.

**EINPROGRESS** The requested I/O is in progress.

The following are additional error codes which may be set for each \texttt{aiocb} control block:

**EOVERFLOW** The \texttt{aiocbp->aio\_lio\_opcode} is \texttt{LIO\_READ}, the file is a regular file, \texttt{aiocbp->aio\_nbytes} is greater than 0, and the \texttt{aiocbp->aio\_offset} is before the end-of-file and is greater than or equal to the offset maximum in the open file description associated with \texttt{aiocbp->aio\_fildes}.

**EFBIG** The \texttt{aiocbp->aio\_lio\_opcode} is \texttt{LIO\_WRITE}, the file is a regular file, \texttt{aiocbp->aio\_nbytes} is greater than 0, and the \texttt{aiocbp->aio\_offset} is greater than or equal to the offset maximum in the open file description associated with \texttt{aiocbp->aio\_fildes}.

**USAGE** The \texttt{lio\_listio\()} function has an explicit 64-bit equivalent. See \texttt{interface64\(5\)}. 

**ATTRIBUTES** See \texttt{attributes\(5\)} for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**SEE ALSO** \texttt{close\(2\)}, \texttt{exec\(2\)}, \texttt{exit\(2\)}, \texttt{fork\(2\)}, \texttt{lseek\(2\)}, \texttt{read\(2\)}, \texttt{write\(2\)}, \texttt{aio\_cancel\(3R\)}, \texttt{aio\_fsync\(3R\)}, \texttt{aio\_read\(3R\)}, \texttt{aio\_return\(3R\)}, \texttt{attributes\(5\)}, \texttt{fcntl\(5\)}, \texttt{interface64\(5\)}, \texttt{siginfo\(5\)}, \texttt{standards\(5\)}

**NOTES** Applications compiled under Solaris 2.3 and 2.4 and using POSIX (see \texttt{standards\(5\)}) Asynchronous Input and Output option must be recompiled to work correctly when Solaris supports this option.

**BUGS** In Solaris 2.5, these functions always return \texttt{−1} and set \texttt{errno} to ENOSYS, because this release does not support the Asynchronous Input and Output option. Beginning with Solaris 2.6, these interfaces are supported.
NAME
listen – listen for connections on a socket

SYNOPSIS
c
cc [ flag ... ] file ... -ls
include <sys/types.h>
include <sys/socket.h>

int listen(int s, int backlog);

DESCRIPTION
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.

The backlog parameter defines the maximum length the queue of pending connections
may grow to.

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 backlog is not cleared by the time
the tcp times out, the connect will fail with ETIMEDOUT.

RETURN VALUES
A 0 return value indicates success; −1 indicates an error.

ERRORS
The call fails if:

EBADF The argument s is not a valid file descriptor.
ENOTSOCK The argument s is not a socket.
EOPNOTSUPP The socket is not of a type that supports the operation listen().

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
accept(3N), connect(3N), socket(3N), attributes(5), socket(5)

NOTES
There is currently no backlog limit.
NAME
listen – listen for socket connections and limit the queue of incoming connections

SYNOPSIS
cc [ flag ...] file ... -lxnet [ library ...]
#include <sys/socket.h>
int listen(int socket, int backlog);

DESCRIPTION
The listen() function marks a connection-mode socket, specified by the socket argument, as accepting connections, and limits the number of outstanding connections in the socket’s listen queue to the value specified by the backlog argument.
If listen() is called with a backlog argument value that is less than 0, the function sets the length of the socket’s listen queue to 0.
If backlog exceeds the maximum queue length, the length of the socket’s listen queue will be set to the maximum supported value.

RETURN VALUES
Upon successful completions, listen() returns 0. Otherwise, −1 is returned and errno is set to indicate the error.

ERRORS
The listen() function will fail if:
EBADF The socket argument is not a valid file descriptor.
ENOTSOC The socket argument does not refer to a socket.
EOPNOTSUP The socket protocol does not support listen().
EINVAL The socket is already connected.
EDESTADDRREQ The socket is not bound to a local address, and the protocol does not support listening on an unbound socket.

The listen() function may fail if:
EINVAL The socket has been shut down.
ENOBUFFS Insufficient resources are available in the system to complete the call.

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

ATTRIBUTE TYPE   ATTRIBUTE VALUE
MT-Level          MT-Safe

SEE ALSO
accept(3XN), connect(3XN), socket(3XN), attributes(5), socket(5)
NAME
localeconv – get numeric formatting information

SYNOPSIS
#include <locale.h>
struct lconv *localeconv(void);

DESCRIPTION
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).

```c
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 */
```

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:

- **char *decimal_point**
  The decimal-point character used to format non-monetary quantities.

- **char *thousands_sep**
  The character used to separate groups of digits to the left of the decimal-point character in formatted non-monetary quantities.
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 ele-
ments 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 for-
matted 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 quan-
tities.

<table>
<thead>
<tr>
<th>Country</th>
<th>Positive format</th>
<th>Negative format</th>
<th>International format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Italy</td>
<td>L.1234</td>
<td>-L.1234</td>
<td>ITL.1234</td>
</tr>
<tr>
<td>Netherlands</td>
<td>F 1.234,56</td>
<td>F -1.234,56</td>
<td>NLG 1.234,56</td>
</tr>
<tr>
<td>Norway</td>
<td>kr1.234,56</td>
<td>kr1.234,56-</td>
<td>NOK 1.234,56</td>
</tr>
<tr>
<td>Switzerland</td>
<td>SFrs.1,234,56</td>
<td>SFrs.1,234,56C</td>
<td>CHF 1,234,56</td>
</tr>
</tbody>
</table>

For these four countries, the respective values for the monetary members of the structure
returned by localeconv are as follows:

<table>
<thead>
<tr>
<th>int_curr_symbol</th>
<th>Italy</th>
<th>Netherlands</th>
<th>Norway</th>
<th>Switzerland</th>
</tr>
</thead>
<tbody>
<tr>
<td>currency_symbol</td>
<td>&quot;L.&quot;</td>
<td>&quot;NLG&quot;</td>
<td>&quot;kr&quot;</td>
<td>&quot;SFrs.&quot;</td>
</tr>
<tr>
<td>mon_decimal_point</td>
<td>&quot;.&quot;</td>
<td>&quot;.&quot;</td>
<td>&quot;.&quot;</td>
<td>&quot;.&quot;</td>
</tr>
<tr>
<td>mon_thousands_sep</td>
<td>&quot;\3&quot;</td>
<td>&quot;\3&quot;</td>
<td>&quot;\3&quot;</td>
<td>&quot;\3&quot;</td>
</tr>
<tr>
<td>mon_grouping</td>
<td>&quot;\3&quot;</td>
<td>&quot;\3&quot;</td>
<td>&quot;\3&quot;</td>
<td>&quot;\3&quot;</td>
</tr>
<tr>
<td>positive_sign</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
localeconv (3C)

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

ENVIRONMENT

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.

FILES
/usr/lib/locale/LC_MONETARY/monetary
   LC_MONETARY database for locale
/usr/lib/locale/LC_NUMERIC/numeric
   LC_NUMERIC database for locale

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe with exceptions</td>
</tr>
<tr>
<td>CSI</td>
<td>Enabled</td>
</tr>
</tbody>
</table>

SEE ALSO
setlocale(3C), attributes(5)

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

#include <unistd.h>

int lockf(int filedes, int function, off_t size);

The lockf() function allows sections of a file to be locked; advisory or mandatory write locks depending on the mode bits of the file (see chmod(2)). Locking calls from other processes that attempt to lock the locked file section will either return an error value or be put to sleep until the resource becomes unlocked. All the locks for a process are removed when the process terminates. See fcntl(2) for more information about record locking.

The filedes argument is an open file descriptor. The file descriptor must have O_WRONLY or O_RDWR permission in order to establish locks with this function call.

function is a control value that specifies the action to be taken. The permissible values for function are defined in <unistd.h> as follows:

#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 */

All other values of function are reserved for future extensions and will result in an error return if not implemented.

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.

The size argument is the number of contiguous bytes to be locked or unlocked. The resource to be locked or unlocked starts at the current offset in the file and extends forward for a positive size and backward for a negative size (the preceding bytes up to but not including the current offset). If size 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.

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 size bytes or to the end of file if size 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.
F_LOCK and F_TLOCK requests differ only by the action taken if the resource is not available. F_LOCK will cause the calling process to sleep until the resource is available. F_TLOCK will cause the function to return a −1 and set errno to EAGAIN if the section is already locked by another process.

File locks are released on first close by the locking process of any file descriptor for the file.

F_UNLOCK requests may, in whole or in part, release one or more locked sections controlled by the process. When sections are not fully released, the remaining sections are still locked by the process. Releasing the center section of a locked section requires an additional element in the table of active locks. If this table is full, an errno is set to EDEADLK and the requested section is not released.

An F_UNLOCK request in which size is non-zero and the offset of the last byte of the requested section is the maximum value for an object of type off_t, when the process has an existing lock in which size is 0 and which includes the last byte of the requested section, will be treated as a request to unlock from the start of the requested section with a size equal to 0. Otherwise, an F_UNLOCK request will attempt to unlock only the requested section.

A potential for deadlock occurs if a process controlling a locked resource is put to sleep by requesting another process’s locked resource. Thus calls to lockf() or fcntl(2) scan for a deadlock prior to sleeping on a locked resource. An error return is made if sleeping on the locked resource would cause a deadlock.

Sleeping on a resource is interrupted with any signal. The alarm(2) function may be used to provide a timeout facility in applications that require this facility.

RETURN VALUES
Upon successful completion, 0 is returned. Otherwise, −1 is returned and errno is set to indicate the error.

ERRORS
The lockf() function will fail if:

EBADF The fildes argument is not a valid open file descriptor; or function is F_LOCK or F_TLOCK and fildes is not a valid file descriptor open for writing.

EACCES or EAGAIN The function argument is F_TLOCK or F_TEST and the section is already locked by another process.

EDEADLK The function argument is F_LOCK and a deadlock is detected.

EINTR A signal was caught during execution of the function.

ECOMM The fildes argument is on a remote machine and the link to that machine is no longer active.

EINVAL The function argument is not one of F_LOCK, F_TLOCK, F_TEST, or F_UNLOCK; or size plus the current file offset is less than 0.
EOVERFLOW  The offset of the first, or if size is not 0 then the last, byte in the requested section cannot be represented correctly in an object of type off_t.

The lockf() function may fail if:

EAGAIN  The function argument is F_LOCK or F_TLOCK and the file is mapped with mmap(2).

EDEADLK or ENOLCK  The function argument is F_LOCK, F_TLOCK, or F_ULOCK, and the request would cause the number of locks to exceed a system-imposed limit.

EOPNOTSUPP or EINVAL  The locking of files of the type indicated by the fildes argument is not supported.

USAGE  Record-locking should not be used in combination with the fopen(3S), fread(3S), fwrite(3S) and other stdio functions. Instead, the more primitive, non-buffered functions (such as open(2)) should be used. Unexpected results may occur in processes that do buffering in the user address space. The process may later read/write data which is/was locked. The stdio functions are the most common source of unexpected buffering.

The alarm(2) function may be used to provide a timeout facility in applications requiring it.

The lockf(2) function has an explicit 64-bit equivalent. See interface64(5).

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  intro(2), alarm(2), chmod(2), close(2), creat(2), fcntl(2), mmap(2), open(2), read(2), write(2), attributes(5), interface64(5)
NAME    log10 – base 10 logarithm function

SYNOPSIS cc [ flag ... ] file ... -lm [ library ... ]
#include <math.h>

double log10(double x);

DESCRIPTION The log10() function computes the base 10 logarithm of \( x \), \( \log_{10}(x) \). The value of \( x \) must be positive.

RETURN VALUES Upon successful completion, log10() returns the base 10 logarithm of \( x \).
If \( x \) is NaN, NaN is returned.
If \( x \) is less than 0, -HUGE_VAL or NaN is returned, and errno is set to EDOM.
If \( x \) is 0, -HUGE_VAL is returned and errno may be set to ERANGE.
For exceptional cases, matherr(3M) tabulates the values to be returned as dictated by Standards other than XPG4.

ERRORS The log10() function will fail if:
EDOM The value of \( x \) is negative.
The log10() function may fail if:
ERANGE The value of \( x \) is 0.
No other errors will occur.

USAGE An application wishing to check for error situations should set errno to 0 before calling log10(). If errno is non-zero on return, or the return value is NaN, an error has occurred.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO isnan(3M), log(3M), matherr(3M), pow(3M), attributes(5), standards(5)
**NAME**
log1p – compute natural logarithm

**SYNOPSIS**
cc [ flag ... ] file ... -lm [ library ... ]
#include <math.h>
double log1p(double x);

**DESCRIPTION**
The log1p() function computes \( \log_e(1.0 + x) \). The value of \( x \) must be greater than \(-1.0\).

**RETURN VALUES**
Upon successful completion, log1p() returns the natural logarithm of \( 1.0 + x \).
If \( x \) is NaN, log1p() returns NaN.
If \( x \) is less than \(-1.0\), log1p() returns \(-\text{HUGE\_VAL}\) or NaN and sets errno to EDOM.
If \( x \) is \(-1.0\), log1p() returns \(-\text{HUGE\_VAL}\) and may set errno to ERANGE.
For exceptional cases, matherr(3M) tabulates the values to be returned as dictated by Standards other than XPG4.

**ERRORS**
The log1p() function will fail if:
EDOM The value of \( x \) is less than \(-1.0\).
The log1p() function may fail and set errno to:
ERANGE The value of \( x \) is \(-1.0\).

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**SEE ALSO**
log(3M), matherr(3M), attributes(5), standards(5)
NAME  
log – natural logarithm function

SYNOPSIS  
cc [ flag ... ] file ... -lm [ library ... ]
#include <math.h>

double log(double x);

DESCRIPTION  
The log() function computes the natural logarithm of x, \( \log_e(x) \). The value of x must be positive.

RETURN VALUES  
Upon successful completion, log() returns the natural logarithm of x.
If x is NaN, NaN is returned.
If x is less than 0, -HUGE_VAL or NaN is returned and errno is set to EDOM.
If x is 0, -HUGE_VAL is returned and errno may be set to ERANGE.
In IEEE754 mode (the -xlibmieee cc compilation option), if x is Inf or a quiet NaN, x is returned; if x is a signaling NaN, a quiet NaN is returned and the invalid operation exception is raised; if x is 1, 0 is returned; for all other positive x, a normalized number is returned and the inexact exception is raised.
For exceptional cases, matherr(3M) tabulates the values to be returned as dictated by Standards other than XPG4.

ERRORS  
The log() function will fail if:
EDOM The value of x is negative.
The log() function may fail if:
ERANGE The value of x is 0.
No other errors will occur.

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

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  
exp(3M), isnan(3M), log10(3M), log1p(3M), matherr(3M), attributes(5), standards(5)
NAME        logb – radix-independent exponent

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

DESCRIPTION The logb() function computes the exponent of x, which is the integral part of \log_r |x|, as a signed floating point value, for non-zero x, where r is the radix of the machine’s floating-point arithmetic.

RETURN VALUES Upon successful completion, logb() returns the exponent of x.
            If x is 0.0, logb() returns -HUGE_VAL and sets errno to EDOM.
            If x is \pm Inf, logb() returns +Inf.
            If x is NaN, logb() returns NaN.
            For exceptional cases, matherr(3M) tabulates the values to be returned as dictated by various Standards.

ERRORS      The logb() function will fail if:
            EDOM      The x argument is 0.0.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO     ilogb(3M), matherr(3M), attributes(5)
**NAME**  
_longjmp, _setjmp – non-local goto

**SYNOPSIS**  
```c
#include <setjmp.h>
void _longjmp(jmp_buf env, int val);
int _setjmp(jmp_buf env);
```

**DESCRIPTION**  
The _longjmp() and _setjmp() functions are identical to longjmp(3C) and setjmp(3C), respectively, with the additional restriction that _longjmp() and _setjmp() do not manipulate the signal mask.

If _longjmp() is called even though env was never initialized by a call to _setjmp(), or when the last such call was in a function that has since returned, the results are undefined.

**RETURN VALUES**  
Refer to longjmp(3C) and setjmp(3C).

**ERRORS**  
No errors are defined.

**USAGE**  
If _longjmp() is executed and the environment in which _setjmp() was executed no longer exists, errors can occur. The conditions under which the environment of the _setjmp() no longer exists include exiting the function that contains the _setjmp() call, and exiting an inner block with temporary storage. This condition might not be detectable, in which case the _longjmp() occurs and, if the environment no longer exists, the contents of the temporary storage of an inner block are unpredictable. This condition might also cause unexpected process termination. If the function has returned, the results are undefined.

Passing longjmp() a pointer to a buffer not created by setjmp(), passing _longjmp() a pointer to a buffer not created by _setjmp(), passing siglongjmp(3C) a pointer to a buffer not created by sigsetjmp(3C) or passing any of these three functions a buffer that has been modified by the user can cause all the problems listed above, and more.

The _longjmp() and _setjmp() functions are included to support programs written to historical system interfaces. New applications should use siglongjmp(3C) and sigsetjmp(3C) respectively.

**SEE ALSO**  
longjmp(3C), setjmp(3C), siglongjmp(3C), sigsetjmp(3C)

modified 28 Feb 1996
NAME longname – return full terminal type name

SYNOPSIS #include <curses.h>
const char *longname(void);

DESCRIPTION The longname() function returns a pointer to a static area containing a verbose description (128 characters or fewer) of the terminal. The area is defined after calls to initscr(3XC), newterm(3XC), or setupterm(3XC). The value should be saved if longname() is going to be used with multiple terminals since it will be overwritten with a new value after each call to newterm() or setupterm().

RETURN VALUES On success, the longname() function returns a pointer to a verbose description of the terminal. Otherwise, it returns a null pointer.

ERRORS None.

SEE ALSO initscr(3XC), newterm(3XC), setupterm(3XC)
NAME       lsearch, lfind – linear search and update

SYNOPSIS   #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 *));

DESCRIPTION lsearch() is a linear search routine generalized from Knuth (6.1) Algorithm S. (See The Art of Computer Programming, Volume 3, Section 6.1, by Donald E. Knuth.) 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. key points to the datum to be sought in the table. base points to the first element in the table. nelp 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. width is the size of an element in bytes. compar 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.

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.

Note that:

- 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   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.
            #include <search.h>
            #include <string.h>
            #include <stdlib.h>
            #include <stdio.h>
            #define TABSIZE 50
            #define ELSIZE 120

            main()
            {
                char line[ELSIZE];    /* buffer to hold input string */
                char tab[TABSIZE][ELSIZE]; /* table of strings */
                size_t nel = 0;        /* number of entries in tab */
            }

modified 29 Dec 1996
int i;

while (fgets(line, ELSIZE, stdin) != NULL &&
    nel < TABSIZE)
    (void) lsearch(line, tab, &nel, ELSIZE, mycmp);
for( i = 0; i < nel; i++)
    (void)fputs(tab[i], stdout);
return 0;

ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO bsearch(3C), hsearch(3C), string(3C), tsearch(3C), attributes(5)


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
#include <sys/types.h>
#include <sys/mman.h>

int madvise(caddr_t addr, size_t len, int advice);

DESCRIPTION
madvise() advises the kernel that a region of user mapped memory in the range \([addr, addr + len]\) 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.

Values for `advice` are defined in `<sys/mman.h>` as:

```c
#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 */
```

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.

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.

MADV_SEQUENTIAL
Tells the system that addresses in this range are likely to be accessed only once, so the system will free the resources mapping the address range as quickly as possible. This is used in the `cat(1)` and `cp(1)` utilities.

MADV_WILLNEED
Tells the system that a certain address range is definitely needed so the kernel will start reading the specified range into memory. This can benefit programs wanting to minimize the time needed to access memory the first time, as the kernel would need to read in from the file.

MADV_DONTNEED
Tells the kernel that the specified address range is no longer needed, so the system starts to free the resources associated with the address range.
madvise() should be used by programs with specific knowledge of their access patterns over a memory object, such as a mapped file, to increase system performance.

**RETURN VALUES**

madvise() returns:
- 0 on success.
- −1 on failure and sets **errno** to indicate the error.

**ERRORS**

- **EINVAL**
  - *addr* is not a multiple of the page size as returned by **sysconf(3C)**.
  - The length of the specified address range is less than or equal to 0, or the advice was invalid.
- **EIO**
  - An I/O error occurred while reading from or writing to the file system.
- **ENOMEM**
  - Addresses in the range [addr, addr + len) are outside the valid range for the address space of a process, or specify one or more pages that are not mapped.
- **ESTALE**
  - Stale nfs file handle.

**ATTRIBUTES**

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**SEE ALSO**

cat(1), cp(1), mmap(2), sysconf(3C), attributes(5)
NAME

maillock, mailunlock, touchlock – functions to manage lockfile(s) for user’s mailbox

SYNOPSIS

cc [ flag . . . ] file . . . -mail [ library . . . ]
#include <maillock.h>
int maillock(const char *user, int retrycnt);
int mailunlock(void);
void touchlock(void);

DESCRIPTION

The maillock() function attempts to create a lockfile for the user’s mailbox. If a lockfile
already exists, and it has not been modified in the last 5 minutes, maillock() will remove
the lockfile and set its own lockfile.

It is crucial that programs locking mail files refresh their locks at least every three
minutes to maintain the lock. Refresh the lockfile by calling the routine touchlock() with
no arguments.

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.

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 retrycnt.

When the lockfile is no longer needed, it should be removed by calling mailunlock().

user is the login name of the user for whose mailbox the lockfile will be created. mail-
lock() assumes that user’s mailfiles are in the “standard” place as defined in
<maillock.h>.

RETURN VALUES

The following return code definitions are contained in <maillock.h>.

#define L_SUCCESS 0 /* Lockfile created or removed */
#define L_NAMELEN 1 /* Recipient name > 13 chars */
#define L_TMPLOCK 2 /* Can’t create tmp file */
#define L_TMPWRITE 3 /* Can’t write pid into lockfile */
#define L_MAXTRY川 4 /* Failed after retrycnt attempts */
#define L_ERROR 5 /* Check errno for reason */

FILES

LIBDIR/lib-mail.ln
LIBDIR/mail.a
/var/mail/*
/var/mail/*.lock

modified 8 May 1997

SunOS 5.6

3X-943
ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
attributes(5)

NOTES
mailunlock() will only remove the lockfile created from the most previous call to maillock(). Calling maillock() for different users without intervening calls to mailunlock() will cause the initially created lockfile(s) to remain, potentially blocking subsequent message delivery until the current process finally terminates.
NAME  
makecontext, swapcontext – manipulate user contexts

SYNOPSIS  
#include <ucontext.h>

void makecontext(ucontext_t *ucp, void(*func)(), int argc, ...);
int swapcontext(ucontext_t *oucp, const ucontext_t *ucp);

DESCRIPTION  
These functions are useful for implementing user-level context switching between multiple threads of control within a process.

makecontext() modifies the context specified by ucp, 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 func, passing it the arguments that follow argc in the makecontext() call. The integer value of argc must be one-greater-than the number of arguments that follow argc; otherwise, the behavior is undefined. For 5 arguments, the value of argc must be 6.

Before a call is made to makecontext(), the context being modified should have a stack allocated for it. The value of argc must match the number of integer arguments passed to func(), otherwise the behavior is undefined.

The uc_link member is used to determine the context that will be resumed when the context being modified by makecontext() returns. The uc_link member should be initialized prior to the call to makecontext().

swapcontext() saves the current context in the context structure pointed to by oucp and sets the context to the context structure pointed to by ucp.

RETURN VALUES  
On successful completion, swapcontext() returns 0. Otherwise, −1 is returned and errno is set to indicate the error.

ERRORS  
The makecontext() and swapcontext() functions will fail if:

EFAULT     ucp or oucp points to an invalid address.
ENOMEM      ucp does not have enough stack left to complete the operation.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  
exit(2), getcontext(2), sigaction(2), sigprocmask(2), attributes(5), ucontext(5)

NOTES  
The size of the ucontext_t structure may change in future releases. To remain binary compatible, users of these features must always use makecontext() or getcontext() to create new instances of them.

modified 29 Dec 1996  
SunOS 5.6  
3C-945
NAME
makedev, major, minor – manage a device number

SYNOPSIS
#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);

DESCRIPTION
The makedev() routine returns a formatted device number on success and NODEV on failure. maj is the major number. min is the minor number. makedev() can be used to create a device number for input to mknod(2).
The major() routine returns the major number component from device.
The minor() routine returns the minor number component from device.

RETURN VALUES
On failure, NODEV is returned and errno is set to indicate the error.

ERRORS
makedev() will fail if one or more of the following are true:
EINVAL One or both of the arguments maj and min is too large.
EINVAL The device number created from maj and min is NODEV.
major() will fail if one or more of the following are true:
EINVAL The device argument is NODEV.
EINVAL The major number component of device is too large.
minor() will fail if the following is true:
EINVAL The device argument is NODEV.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
mknod(2), stat(2), attributes(5)
NAME
malloc, calloc, free, memalign, realloc, valloc, alloca – memory allocator

SYNOPSIS
#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);

DESCRIPTION
malloc() and free() provide a simple general-purpose memory allocation package. malloc() returns a pointer to a block of at least size bytes suitably aligned for any use. 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 ptr is a NULL pointer, no action occurs. Undefined results will occur if the space assigned by malloc() is overrun or if some random number is handed to free().
calloc() allocates space for an array of nelem elements of size elsize. The space is initialized to zeros.
memalign() allocates size 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 alignment. Note: the value of alignment must be a power of two, and must be greater than or equal to the size of a word.
realloc() changes the size of the block pointed to by ptr to size 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 ptr is NULL, realloc() behaves like malloc() for the specified size. If size is zero and ptr is not a null pointer, the object pointed to is freed.
valloc() is equivalent to memalign(sysconf(_SC_PAGESIZE),size).
Each of the allocation routines returns a pointer to space suitably aligned (after possible pointer coercion) for storage of any type of object.
malloc(), realloc(), memalign(), and valloc() will fail if there is not enough available memory.
alloca() allocates size bytes of space in the stack frame of the caller, and returns a pointer to the allocated block. This temporary space is automatically freed when the caller returns. If the allocated block is beyond the current stack limit, the resulting behavior is undefined.
RETURN VALUES
If there is no available memory, `malloc()`, `realloc()`, `memalign()`, `valloc()`, and `calloc()` return a null pointer. When `realloc()` returns NULL, the block pointed to by `ptr` is left intact. If `size`, `nelem`, or `elsize` is 0, a unique pointer to the arena is returned.

ERRORS
If `malloc()`, `calloc()`, or `realloc()` returns unsuccessfully, `errno` will be set to indicate the following:

- **ENOMEM**: `size` bytes of memory exceeds the physical limits of your system, and cannot be allocated.
- **EAGAIN**: There is not enough memory available at this point in time to allocate `size` bytes of memory; but the application could try again later.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
`brk(2)`, `getrlimit(2)`, `bsdmalloc(3X)`, `malloc(3X)`, `mapmalloc(3X)`, `watchmalloc(3X)`, attributes(5)

WARNINGS
Undefined results will occur if the size requested for a block of memory exceeds the maximum size of a process’s heap, which may be obtained with `getrlimit()`.

`malloc()` 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` routines are a trade-off between performance and space-efficiency.

`free()` does not set `errno`. 
### NAME
malloc, free, realloc, calloc, mallopt, mallinfo – memory allocator

### SYNOPSIS

```c
cc [ flag ...] file ... -lmalloc [ library ...]
#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);
```

### DESCRIPTION

malloc() and free() provide a simple general-purpose memory allocation package. malloc() returns a pointer to a block of at least size 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 ptr is a null pointer, no action occurs. Undefined results occur if the space assigned by malloc() is overrun or if some random number is handed to free().

realloc() changes the size of the block pointed to by ptr to size 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 ptr is a null pointer, realloc() behaves like malloc() for the specified size. If size is zero and ptr is not a null pointer, the object it points to is freed.

calloc() allocates space for an array of nelem elements of size elsize. The space is initialized to zeros.

mallopt() provides for control over the allocation algorithm. The available values for cmd are:

**M_MXFAST**

Set maxfast to value. The algorithm allocates all blocks below the size of maxfast in large groups and then doles them out very quickly. The default value for maxfast is 24.

**M_NLBLKS**

Set numlblks to value. The above mentioned “large groups” each contain numlblks blocks. numlblks must be greater than 0. The default value for numlblks is 100.

**M_GRAIN**

Set grain to value. The sizes of all blocks smaller than maxfast are considered to be rounded up to the nearest multiple of grain. grain must be greater than 0. The default value of grain is the smallest number of bytes that will allow alignment of any data type. Value will be rounded up to a multiple of the default when grain is set.

---

modified 31 Dec 1996

SunOS 5.6
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.

malloc() may be called repeatedly, but may not be called after the first small block is allocated.

mallocopt() 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 smbufs; /* number of small blocks */
- int hblkhd; /* space in holding block headers */
- int hblks; /* number of holding blocks */
- int usmbufs; /* 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 mallocopt() is called after any allocation or if cmd or value are invalid, non-zero is returned. Otherwise, it returns zero.

**ERRORS**

If malloc(), calloc(), or realloc() returns unsuccessfully, errno will be set to indicate the following:

- ENOMEM size bytes of memory exceeds the physical limits of your system, and cannot be allocated.
- EAGAIN There is not enough memory available AT THIS POINT IN TIME to allocate size bytes of memory; but the application could try again later.

**ATTRIBUTES**

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

**SEE ALSO**

brk(2), malloc(3C), bsdmalloc(3X), attributes(5)
NOTES

Note that unlike `malloc(3C)`, this package does not preserve the contents of a block when it is freed, unless the `M_KEEP` option of `mallopt()` is used. Undocumented features of `malloc(3C)` have not been duplicated.

Function prototypes for `malloc()`, `realloc()`, `calloc()`, and `free()` are also defined in the `<malloc.h>` header for compatibility with old applications. New applications should include `<stdlib.h>` to access the prototypes for these functions.

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

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

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

modified 31 Dec 1996
### NAME
mapmalloc, calloc, cfree, free, realloc, – memory allocator

### SYNOPSIS
c
flag ...] file ... -lmmapmalloc [ library ... ]
#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);

### DESCRIPTION
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. `malloc()` and `free()` provide a simple general-purpose memory allocation package. `malloc()` returns a pointer to a block of at least `size` bytes suitably aligned for any use. The argument to `free()` is a pointer to a block previously allocated by `malloc()`, `calloc()` or `realloc()`. If `ptr` is a NULL pointer, no action occurs. Undefined results will occur if the space assigned by `malloc()` is overrun or if some random number is handed to `free()`. `calloc()` allocates space for an array of `nelem` elements of size `elsize`. The space is initialized to zeros. `realloc()` changes the size of the block pointed to by `ptr` to `size` 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 `ptr` is NULL, `realloc()` behaves like `malloc()` for the specified size. If `size` is zero and `ptr` is not a null pointer, the object pointed to is freed. Each of the allocation routines returns a pointer to space suitably aligned (after possible pointer coercion) for storage of any type of object. `malloc()` and `realloc()` will fail if there is not enough available memory. Entry points for `malloc_debug()`, `mallocmap()`, `mallopt()`, `mallinfo()`, `memalign()`, and `valloc()`, are empty routines, and are provided only to protect the user from mixing `malloc()` functions from different implementations.

### RETURN VALUES
If there is no available memory, `malloc()`, `realloc()`, and `calloc()` return a null pointer. When `realloc()` returns NULL, the block pointed to by `ptr` is left intact. If `size`, `nelem`, or `elsize` is 0, a unique pointer to the arena is returned.
FILES
/usr/lib/libmalloc

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
brk(2), getrlimit(2), mmap(2), realloc(3C), attributes(5)
NAME
matherr – math library exception-handling function

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

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
In SVID3 mode (code compiled with cc −Xt), 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.

<table>
<thead>
<tr>
<th>SVID3 ERROR HANDLING PROCEDURES (compile with cc −Xt)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Types of Errors</td>
</tr>
<tr>
<td>&lt;math.h&gt; type</td>
</tr>
<tr>
<td>errno</td>
</tr>
<tr>
<td>IEEE Exception</td>
</tr>
<tr>
<td>fp_exception_type</td>
</tr>
<tr>
<td>ACOS, ASIN (</td>
</tr>
<tr>
<td>ACOSH (x &lt; 1), ATANH (</td>
</tr>
<tr>
<td>ATAN2 (0,0):</td>
</tr>
<tr>
<td>COSH, SINH:</td>
</tr>
<tr>
<td>EXP:</td>
</tr>
<tr>
<td>FMOD(x,0):</td>
</tr>
<tr>
<td>HYPOT:</td>
</tr>
<tr>
<td>J0, J1, JN (</td>
</tr>
<tr>
<td>LGAMMA: usual cases</td>
</tr>
<tr>
<td>(x = 0 or −integer)</td>
</tr>
<tr>
<td>LOG, LOG10: (x &lt; 0)</td>
</tr>
<tr>
<td>(x = 0)</td>
</tr>
<tr>
<td>POW: usual cases</td>
</tr>
<tr>
<td>(x &lt; 0) ** (y not an integer)</td>
</tr>
<tr>
<td>0 ** 0</td>
</tr>
<tr>
<td>0 ** (y &lt; 0)</td>
</tr>
<tr>
<td>REMAINDER (x,0):</td>
</tr>
<tr>
<td>SCALB:</td>
</tr>
<tr>
<td>SQRT (x &lt; 0):</td>
</tr>
<tr>
<td>Y0, Y1, YN: (x &lt; 0)</td>
</tr>
<tr>
<td>(x = 0)</td>
</tr>
<tr>
<td>(x &gt; X_TLOSS)</td>
</tr>
</tbody>
</table>

modified 29 Dec 1996  SunOS 5.6  3M-955
ABBREVIATIONS

<table>
<thead>
<tr>
<th>ABBREVIATION</th>
<th>MEANING</th>
</tr>
</thead>
<tbody>
<tr>
<td>Md</td>
<td>Message is printed (DOMAIN error).</td>
</tr>
<tr>
<td>Ms</td>
<td>Message is printed (SING error).</td>
</tr>
<tr>
<td>Mt</td>
<td>Message is printed (TLOSS error).</td>
</tr>
<tr>
<td>NaN</td>
<td>IEEE NaN result and invalid operation exception.</td>
</tr>
<tr>
<td>HUGE</td>
<td>Maximum finite single-precision floating-point number.</td>
</tr>
<tr>
<td>HUGE_VAL</td>
<td>IEEE ± result and division-by-zero exception.</td>
</tr>
<tr>
<td>X_TLOSS</td>
<td>The value X_TLOSS is defined in &lt;values.h&gt;.</td>
</tr>
</tbody>
</table>

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.

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`.

<table>
<thead>
<tr>
<th>Types of Errors</th>
<th>&lt;math.h&gt; type</th>
<th>&lt;math.h&gt; type</th>
<th>&lt;math.h&gt; type</th>
<th>&lt;math.h&gt; type</th>
<th>&lt;math.h&gt; type</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>errno</code></td>
<td>EDOM</td>
<td>EDOM</td>
<td>ERANGE</td>
<td>ERANGE</td>
<td>ERANGE</td>
</tr>
<tr>
<td>ACOS, ASIN (</td>
<td>x</td>
<td>&gt; 1):</td>
<td>0.0</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ATAN2 (0,0):</td>
<td>0.0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>COSH, SINH:</td>
<td>-</td>
<td>-</td>
<td>[+HUGE_VAL]</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>EXP:</td>
<td>-</td>
<td>-</td>
<td>[+HUGE_VAL]</td>
<td>[0.0]</td>
<td></td>
</tr>
<tr>
<td>FMOD (x,0):</td>
<td>[NaN]</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>HYPOT:</td>
<td>-</td>
<td>-</td>
<td>[+HUGE_VAL]</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>J0, J1, JN (</td>
<td>x</td>
<td>&gt; X_TLOSS):</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>LGAMMA:</td>
<td>-</td>
<td>-</td>
<td>[+HUGE_VAL]</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>(x = 0 or -integer)</td>
<td>-</td>
<td>+HUGE_VAL</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>LOG, LOG10:</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>(x &lt; 0)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>(x = 0)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>POW:</td>
<td>-</td>
<td>-</td>
<td>[+HUGE_VAL]</td>
<td>±0.0</td>
<td></td>
</tr>
<tr>
<td>usual cases</td>
<td>0.0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>(x &lt; 0) ** (y not an integer)</td>
<td>1.0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>0 ** 0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>0 ** (y &lt; 0)</td>
<td>[+HUGE_VAL]</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>SQRT (x &lt; 0):</td>
<td>0.0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Y0, Y1, YN:</td>
<td>-</td>
<td>[+HUGE_VAL]</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>(x &lt; 0)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>(x = 0)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>(x &gt; X_TLOSS)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.0</td>
<td></td>
</tr>
</tbody>
</table>
ABBREVIATIONS

| {...} | errno is not to be relied upon in all braced cases. |
| NaN | IEEE NaN result and invalid operation exception. |
| HUGE_VAL | IEEE \( \infty \) result and division-by-zero exception. |
| X_TLOSS | The value X_TLOSS is defined in <values.h>. |

ANSI/ISO-C STANDARD CONFORMANCE

The ANSI/ISO-C standard covers a small subset of XPG3. The following table summarizes the values returned in the exceptional cases.

<table>
<thead>
<tr>
<th>ANSI/ISO-C ERROR HANDLING PROCEDURES (compile with cc -Xc)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Types of Errors</td>
</tr>
<tr>
<td>&lt;math.h&gt; type</td>
</tr>
<tr>
<td>errno</td>
</tr>
<tr>
<td>ACOS, ASIN (</td>
</tr>
<tr>
<td>ATAN2(0,0):</td>
</tr>
<tr>
<td>EXP:</td>
</tr>
<tr>
<td>FMOD(x,0):</td>
</tr>
<tr>
<td>LOG, LOG10:</td>
</tr>
<tr>
<td>(x &lt; 0)</td>
</tr>
<tr>
<td>(x = 0)</td>
</tr>
<tr>
<td>POW:</td>
</tr>
<tr>
<td>(x &lt; 0) ** (y not an integer)</td>
</tr>
<tr>
<td>0 ** (y &lt; 0)</td>
</tr>
<tr>
<td>SQRT(x &lt; 0):</td>
</tr>
</tbody>
</table>

ABBREVIATIONS

| NaN | IEEE NaN result and invalid operation exception. |
| HUGE_VAL | IEEE \( \infty \) result and division-by-zero exception. |

EXAMPLES

```c
#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);
        break;
    case OVERFLOW:
        break;
    case UNDERFLOW:
        break;
    case SING:
        break;
    case DOMAIN:
        break;
    default:
        break;
    }
} /* matherr */
```

modified 29 Dec 1996

SunOS 5.6

3M-957
```c
    abort();
    break;
}
return (0); /* all other exceptions, execute default procedure */

ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
attributes(5), standards(5)
```
NAME
mblen – get number of bytes in a character

SYNOPSIS
#include <stdlib.h>
int mblen(const char *s, size_t n);

DESCRIPTION
If s is not a null pointer, mblen() determines the number of bytes constituting the character pointed to by s. It is equivalent to:

mbtowc((wchar_t *)0, s, n);

A call with s as a null pointer causes this function to return 0. The behavior of this function is affected by the LC_CTYPE category of the current locale.

RETURN VALUES
If s is a null pointer, mblen() returns a 0 value. If s is not a null pointer, mblen() either returns 0 (if s points to the null byte), or returns the number of bytes that constitute the character (if the next n or fewer bytes form a valid character), or returns −1 (if they do not form a valid character) and may set errno to indicate the error. In no case will the value returned be greater than n or the value of the MB_CUR_MAX macro.

ERRORS
The mblen() function may fail if:
EILSEQ Invalid character sequence is detected.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe with exceptions</td>
</tr>
<tr>
<td>CSI</td>
<td>Enabled</td>
</tr>
</tbody>
</table>

SEE ALSO
mbstowcs(3C), mbtowc(3C), setlocale(3C), wcstombs(3C), wctomb(3C), attributes(5)

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

modified 20 Dec 1996
SunOS 5.6
3C-959
NAME

mbstowcs – convert a character string to a wide-character string

SYNOPSIS

#include <stdlib.h>

size_t mbstowcs(wchar_t *pwcs, const char *s, size_t n);

DESCRIPTION

The mbstowcs() function converts a sequence of characters from the array pointed to by s into a sequence of corresponding wide-character codes and stores not more than n wide-character codes into the array pointed to by pwcs. No characters that follow a null byte (which is converted into a wide-character code with value 0) will be examined or converted. Each character is converted as if by a call to mbtowc(3C).

No more than n elements will be modified in the array pointed to by pwcs. If copying takes place between objects that overlap, the behavior is undefined.

The behavior of this function is affected by the LC_CTYPE category of the current locale. If pwcs is a null pointer, mbstowcs() returns the length required to convert the entire array regardless of the value of n, but no values are stored.

RETURN VALUES

If an invalid character is encountered, mbstowcs() returns (size_t)-1 and may set errno to indicate the error. Otherwise, mbstowcs() returns the number of the array elements modified (or required if pwcs is NULL), not including a terminating 0 code, if any. The array will not be zero-terminated if the value returned is n.

ERRORS

The mbstowcs() function may fail if the following error is detected:

EILSEC Invalid byte sequence

ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
<tr>
<td>CSI</td>
<td>Enabled</td>
</tr>
</tbody>
</table>

SEE ALSO mblen(3C), mbtowc(3C), setlocale(3C), wcstombs(3C), wctomb(3C), attributes(5)

3C-960 SunOS 5.6 modified 20 Dec 1996
NAME
mbtowc – convert a character to a wide-character code

SYNOPSIS
#include <stdlib.h>

int mbtowc(wchar_t *pwc, const char *s, size_t n);

DESCRIPTION
If s is not a null pointer, mbtowc() determines the number of the bytes that constitute the character pointed to by s. It then determines the wide-character code for the value of type wchar_t that corresponds to that character. (The value of the wide-character code corresponding to the null byte is 0.) If the character is valid and pwc is not a null pointer, mbtowc() stores the wide-character code in the object pointed to by pwc.

A call with s as a null pointer causes this function to return 0. The behavior of this function is affected by the LC_CTYPE category of the current locale. At most n bytes of the array pointed to by s will be examined.

RETURN VALUES
If s is a null pointer, mbtowc() returns a 0 value. If s is not a null pointer, mbtowc() either returns 0 (if s points to the null byte), or returns the number of bytes that constitute the converted character (if the next n or fewer bytes form a valid character), or returns -1 and may set errno to indicate the error (if they do not form a valid character).

In no case will the value returned be greater than n or the value of the MB_CUR_MAX macro.

ERRORS
The mbtowc() function may fail if the following is detected:
EILSEQ Invalid character sequence

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe with exceptions</td>
</tr>
<tr>
<td>CSI</td>
<td>Enabled</td>
</tr>
</tbody>
</table>

SEE ALSO
mblen(3C), mbstowcs(3C), setlocale(3C), wcstombs(3C), wctomb(3C), attributes(5)

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

modified 20 Dec 1996 SunOS 5.6 3C-961
**NAME**
mctl – memory management control

**SYNOPSIS**
/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;

**DESCRIPTION**
mctl() applies a variety of control functions over pages identified by the mappings established for the address range [addr, addr + len]. The function to be performed is identified by the argument function. Valid functions are defined in mman.h as follows:

**MC_LOCK**
Lock the pages in the range in memory. This function is used to support mlock(). See mlock(3C) for semantics and usage. arg is ignored.

**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. addr and len are ignored. arg is an integer built from the flags:

- MCL_CURRENT Lock current mappings
- MCL_FUTURE Lock future mappings

**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. arg is used to represent the flags argument to msync(). It is constructed from an OR of the following values:

- MS_SYNC Synchronized write
- MSASYNC Return immediately
- MS_INVALIDATE Invalidate mappings

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.

**MC_UNLOCK**
Unlock the pages in the range. This function is used to support munlock(). arg is ignored.
MC_UNLOCKAS
Remove address space memory lock, and locks on all current mappings. This function is used to support munlockall(). addr and len must have the value 0. arg is ignored.

RETURN VALUES
mctl() returns 0 on success, −1 on failure.

ERRORS
mctl() fails if:
EAGAIN Some or all of the memory identified by the operation could not be locked due to insufficient system resources.
EBUSY MS_INVALIDATE was specified and one or more of the pages is locked in memory.
EINVAL addr is not a multiple of the page size as returned by getpagesize().
EINVAL addr and/or len do not have the value 0 when MC_LOCKAS or MC_UNLOCKAS are specified.
EINVAL arg is not valid for the function specified.
EIO An I/O error occurred while reading from or writing to the file system.
ENOMEM Addresses in the range [addr, addr + len) are invalid for the address space of a process, or specify one or more pages which are not mapped.
EPERM The process’s effective user ID is not super-user and one of MC_LOCK, MC_LOCKAS, MC_UNLOCK, or MC_UNLOCKAS was specified.

SEE ALSO mmap(2), memcntl(2), getpagesize(3C), mlock(3C), mlockall(3C), msync(3C)

NOTES
Use of these interfaces should be restricted to only applications written on BSD platforms. Use of these interfaces with any of the system libraries or in multi-thread applications is unsupported.
NAME  media_findname – convert a supplied name into an absolute pathname that can be used to access removable media

SYNOPSIS  ccc [flag ...] file ... -lvolmgt [library ...]
#include <volmgt.h>
char *media_findname(char *start);

DESCRIPTION  media_findname() converts the supplied start string into an absolute pathname that can then be used to access a particular piece of media.

The start parameter can be one of the following types of specifications:

/dev/  ... An absolute pathname in /dev, such as /dev/rdiskette0, in which case a copy of that string is returned (see NOTES on this page).

/vol/  ... An absolute Volume Management pathname, such as /vol/dev/aliases/floppy0 or /vol/dsk/fred. If this supplied pathname is not a symbolic link, then a copy of that pathname is returned. If the supplied pathname is a symbolic link then it is dereferenced and a copy of that dereferenced pathname is returned.

volume_name  The Volume Management volume name for a particular volume, such as fred (see fdformat(1) for a description of how to label floppies). In this case a pathname in the Volume Management namespace is returned.

volmgt_symname  The Volume Management symbolic name for a device, such as floppy0 or cdrom2 (see volfs(7FS) for more information on Volume Management symbolic names), in which case a pathname in the Volume Management namespace is returned.

media_type  The Volume Management generic media type name. For example, floppy or cdrom. In this case media_findname() looks for the first piece of media that matches that media type, starting at 0 (zero) and continuing on until a match is found (or some fairly large maximum number is reached). In this case, if a match is found, a copy of the pathname to the volume found is returned.

RETURN VALUES  Upon successful completion media_findname() returns a pointer to the pathname found. In the case of an error a null pointer is returned.

ERRORS  For cases where the supplied start parameter is an absolute pathname, media_findname() can fail, returning a null string pointer, if an lstat(2) of that supplied pathname fails. Also, if the supplied absolute pathname is a symbolic link, media_findname() can fail if a readlink(2) of that symbolic link fails, or if a stat(2) of the pathname pointed to by that symbolic link fails, or if any of the following is true:

ENXIO  The specified absolute pathname was not a character special device, and

3X-964  SunOS 5.6  modified 31 Dec 1996
it was not a directory with a character special device in it.

EXAMPLES

The following example attempts to find what the Volume Management pathname is to a piece of media called fred. Notice that a volmgt_check() is done first (see the NOTES section on this page).

```c
(void) volmgt_check(NULL);
if ((nm = media_findname("fred")) != NULL) {
    (void) printf("media named \"fred\" is at \"%s\"\n", nm);
} else {
    (void) printf("media named \"fred\" not found\n");
}
```

This example looks for whatever volume is in the first floppy drive, letting media_findname() call volmgt_check() if and only if no floppy is currently known to be the first floppy drive.

```c
if ((nm = media_findname("floppy0")) != NULL) {
    (void) printf("path to floppy0 is \"%s\"\n", nm);
} else {
    (void) printf("nothing in floppy0\n");
}
```

ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO

cc(1B), fdformat(1), vold(1M), lstat(2), readlink(2), stat(2), free(3C), malloc(3C), volmgt_check(3X), volmgt_inuse(3X), volmgt_root(3X), volmgt_running(3X), volmgt_symname(3X), attributes(5), volfs(7FS)

NOTES

If media_findname() cannot find a match for the supplied name, it performs a volmgt_check(3X) and tries again, so it can be more efficient to perform volmgt_check() before calling media_findname().

Upon success media_findname() returns a pointer to string which has been allocated; this should be freed when no longer in use (see free(3C)).
media_getattr(3X)                  Miscellaneous Library Functions

NAME  media_getattr, media_setattr – get and set media attributes

SYNOPSIS  cc [ flag ... ] file... -lvolmgt [ library.... ]
            #include <volmgt.h>
            char *media_getattr(char *vol_path, char *attr);
            int media_setattr(char *vol_path, char *attr, char *value);

DESCRIPTION  media_setattr() and media_getattr() respectively set and get attribute-value pairs (called properties) on a per-volume basis.

Volume Management supports system properties and user properties. System properties are ones that Volume Management predefines. Some of these system properties are writable, but only by the user that owns the volume being specified, and some system properties are read only:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Writable</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>s-access</td>
<td>RO</td>
<td>&quot;seq&quot;, &quot;rand&quot;</td>
<td>sequential or random access</td>
</tr>
<tr>
<td>s-density</td>
<td>RO</td>
<td>&quot;low&quot;, &quot;medium&quot;, &quot;high&quot;</td>
<td>media density</td>
</tr>
<tr>
<td>s-parts</td>
<td>RO</td>
<td>comma separated list of slice numbers</td>
<td>list of partitions on this volume</td>
</tr>
<tr>
<td>s-location</td>
<td>RO</td>
<td>&quot;pathname&quot;</td>
<td>Volume Management pathname to media</td>
</tr>
<tr>
<td>s-mejectable</td>
<td>RO</td>
<td>&quot;true&quot;, &quot;false&quot;</td>
<td>whether or not media is manually ejectable</td>
</tr>
<tr>
<td>s-rmoneject</td>
<td>R/W</td>
<td>&quot;true&quot;, &quot;false&quot;</td>
<td>should media access points be removed from database upon ejection</td>
</tr>
<tr>
<td>s-enxio</td>
<td>R/W</td>
<td>&quot;true&quot;, &quot;false&quot;</td>
<td>if set return ENXIO when media access attempted</td>
</tr>
</tbody>
</table>

Properties can also be defined by the user. In this case the value can be any string the user wishes.

RETURN VALUES  Upon successful completion media_getattr() returns a pointer to the value corresponding to the specified attribute. A null pointer is returned if the specified volume doesn’t exist, if the specified attribute for that volume doesn’t exist, if the specified attribute is boolean and its value is false, or if malloc(3C) fails to allocate space for the return value. media_setattr() returns 1 upon success, and 0 upon failure.

ERRORS  Both media_getattr() and media_setattr() can fail returning a null pointer if an open(2) of the specified vol_path fails, if an fstat(2) of that pathname fails, or if that pathname is not a block or character special device.

3X-966  SunOS 5.6  modified 31 Dec 1996
media_getattr() can also fail if the specified attribute was not found, and media_setattr() can also fail if the caller doesn’t have permission to set the attribute, either because it’s is a system attribute, or because the caller doesn’t own the specified volume. Additionally, either routine can fail returning the following error values:

- **ENXIO**: The Volume Management daemon, vold, is not running
- **EINTR**: The routine was interrupted by the user before finishing

**EXAMPLES**

The following example checks to see if the volume called fred that Volume Management is managing can be ejected via software, or if it can only be manually ejected:

```c
if (media_getattr("/vol/rdsk/fred", "s-mejectable") != NULL) {
    (void) printf(""fred"") must be manually ejected"
}
else {
    (void) printf(software can eject "fred"");
}
```

This example shows setting the s-enxio property for the floppy volume currently in the first floppy drive:

```c
int res;

if ((res = media_setattr("/vol/dev/aliases/floppy0", "s-enxio",
    "true") == 0) {
    (void) printf("can’t set s-enxio flag for floppy0"");
}
```

**ATTRIBUTES**

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**SEE ALSO**

cc(1B), vold(1M), lstat(2), open(2), readdir(2), stat(2), free(3C), malloc(3C), media_findname(3X), volmgt_check(3X), volmgt_inuse(3X), volmgt_root(3X), volmgt_running(3X), volmgt_symname(3X), attributes(5)

**NOTES**

Upon success media_getattr() returns a pointer to a string which has been allocated, and should be freed when no longer in use (see free(3C)).
NAME  memory, memcpy, memchr, memcmp, memmove, memset – memory operations

SYNOPSIS  
```c
#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);
```

DESCRIPTION  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.

- **memccpy()**: copies bytes from memory area `s2` into `s1`, stopping after the first occurrence of `c` (converted to an `unsigned char`) has been copied, or after `n` bytes have been copied, whichever comes first. It returns a pointer to the byte after the copy of `c` in `s1`, or a null pointer if `c` was not found in the first `n` bytes of `s2`.

- **memchr()**: returns a pointer to the first occurrence of `c` (converted to an `unsigned char`) in the first `n` bytes (each interpreted as an `unsigned char`) of memory area `s`, or a null pointer if `c` does not occur.

- **memcmp()**: compares its arguments, looking at the first `n` bytes (each interpreted as an `unsigned char`), and returns an integer less than, equal to, or greater than 0, according as `s1` is lexicographically less than, equal to, or greater than `s2` when taken to be unsigned characters.

- **memcpy()**: copies `n` bytes from memory area `s2` to `s1`. It returns `s1`.

- **memmove()**: copies `n` bytes from memory areas `s2` to `s1`. Copying between objects that overlap will take place correctly. It returns `s1`.

- **memset()**: sets the first `n` bytes in memory area `s` to the value of `c` (converted to an `unsigned char`). It returns `s`.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  string(3C), attributes(5)
NAME
menus – character based menus package

SYNOPSIS
#include <menu.h>

DESCRIPTION
The menu library is built using the curses library, and any program using menus routines must call one of the curses initialization routines, such as initscr. A program using these routines must be compiled with −lmenu and −lcurses on the cc command line.
The 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.

<table>
<thead>
<tr>
<th>menus Routine Name</th>
<th>Manual Page Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>current_item</td>
<td>menu_item_current(3X)</td>
</tr>
<tr>
<td>free_item</td>
<td>menu_item_new(3X)</td>
</tr>
<tr>
<td>free_menu</td>
<td>menu_new(3X)</td>
</tr>
<tr>
<td>item_count</td>
<td>menu_items(3X)</td>
</tr>
<tr>
<td>item_description</td>
<td>menu_item_name(3X)</td>
</tr>
<tr>
<td>item_index</td>
<td>menu_item_current(3X)</td>
</tr>
<tr>
<td>item_init</td>
<td>menu_hook(3X)</td>
</tr>
<tr>
<td>item_name</td>
<td>menu_item_name(3X)</td>
</tr>
<tr>
<td>item_opts</td>
<td>menu_item_opts(3X)</td>
</tr>
<tr>
<td>item_opts_off</td>
<td>menu_item_opts(3X)</td>
</tr>
<tr>
<td>item_opts_on</td>
<td>menu_item_opts(3X)</td>
</tr>
<tr>
<td>item_term</td>
<td>menu_hook(3X)</td>
</tr>
<tr>
<td>item_userptr</td>
<td>menu_item_userptr(3X)</td>
</tr>
<tr>
<td>item_value</td>
<td>menu_item_value(3X)</td>
</tr>
<tr>
<td>item_visible</td>
<td>menu_item_visible(3X)</td>
</tr>
<tr>
<td>menu_back</td>
<td>menu_attributes(3X)</td>
</tr>
<tr>
<td>menu_driver</td>
<td>menu_driver(3X)</td>
</tr>
<tr>
<td>menu_fore</td>
<td>menu_attributes(3X)</td>
</tr>
<tr>
<td>menu_format</td>
<td>menu_format(3X)</td>
</tr>
<tr>
<td>menu_grey</td>
<td>menu_attributes(3X)</td>
</tr>
</tbody>
</table>

modified 31 Dec 1996
menu_init  menu_hook(3X)
menu_items  menu_items(3X)
menu_mark  menu_mark(3X)
menu_opts  menu_opts(3X)
menu_opts_off menu_opts(3X)
menu_opts_on  menu_opts(3X)
menu_pad  menu_attributes(3X)
menu_pattern menu_pattern(3X)
menu_sub  menu_win(3X)
menu_term  menu_hook(3X)
menu_userptr  menu_userptr(3X)
menu_win  menu_win(3X)
new_item  menu_item_new(3X)
new_menu  menu_new(3X)
pos_menu_cursor menu_cursor(3X)
post_menu menu_post(3X)
scale_menu menu_win(3X)
set_current_item menu_item_current(3X)
set_item_init menu_hook(3X)
set_item_opts menu_item_opts(3X)
set_item_term menu_hook(3X)
set_item_userptr menu_item_userptr(3X)
set_item_value menu_item_value(3X)
set_menu_back menu_attributes(3X)
set_menu_fore menu_attributes(3X)
set_menu_format menu_format(3X)
set_menu_grey menu_attributes(3X)
set_menu_init menu_hook(3X)
set_menu_items menu_items(3X)
set_menu_mark menu_mark(3X)
set_menu_opts menu_opts(3X)
set_menu_pad menu_attributes(3X)
set_menu_pattern menu_pattern(3X)
set_menu_sub menu_win(3X)
set_menu_term menu_hook(3X)
set_menu_userptr menu_userptr(3X)
set_menu_win menu_win(3X)
set_top_row menu_item_current(3X)
top_row menu_item_current(3X)
unpost_menu menu_post(3X)
RETURN VALUES

Routines that return pointers always return NULL on error. Routines that return an integer return one of the following:

- **E_OK**: The routine returned successfully.
- **E_SYSTEM_ERROR**: System error.
- **E_BAD_ARGUMENT**: An incorrect argument was passed to the routine.
- **E_POSTED**: The menu is already posted.
- **E_CONNECTED**: One or more items are already connected to another menu.
- **E_BAD_STATE**: The routine was called from an initialization or termination function.
- **E_NO_ROOM**: The menu does not fit within its subwindow.
- **E_NOT_POSTED**: The menu has not been posted.
- **E_UNKNOWN_COMMAND**: An unknown request was passed to the menu driver.
- **E_NO_MATCH**: The character failed to match.
- **E_NOT_SELECTABLE**: The item cannot be selected.
- **E_NOT_CONNECTED**: No items are connected to the menu.
- **E_REQUEST_DENIED**: The menu driver could not process the request.

ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO

curses(3X), attributes(5)

NOTES

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

cc [ flag ... ] file ... -lm menuseries [ library ... ]

#include <menu.h>

int set_menu_fore(MENU *menu, chtype attr);
chtdata menu_fore(MENU *menu);
int set_menu_back(MENU *menu, chtype attr);
chtdata menu_back(MENU *menu);
int set_menu_grey(MENU *menu, chtype attr);
chtdata menu_grey(MENU *menu);
int set_menu_pad(MENU *menu, int pad);
chtdata menu_pad(MENU *menu);

DESCRIPTION

set_menu_fore() sets the foreground attribute of menu — 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 menu.

set_menu_back() sets the background attribute of menu — the display attribute for
unselected, yet selectable, items. This display attribute is a curses library visual attribute.

set_menu_grey() sets the grey attribute of menu — 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 menu.

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 menu to pad. menu_pad() returns
the pad character of menu.

RETURN VALUES

These routines 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.

ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO

curses(3X), menus(3X), attributes(5)

3X-972 SunOS 5.6 modified 31 Dec 1996
| NOTES | The header `<menu.h>` automatically includes the headers `<eti.h>` and `<curses.h>`. |

modified 31 Dec 1996 | SunOS 5.6 | 3X-973 |
NAME  menu_cursor, pos_menu_cursor – correctly position a menus cursor

SYNOPSIS  cc [ flag ... ] file ... -lmenu -lcurses [ library ... ]
            #include <menu.h>
            int pos_menu_cursor(MENU *menu);

DESCRIPTION  pos_menu_cursor() moves the cursor in the window of menu to the correct position to resume menu processing. This is needed after the application calls a curses library I/O routine.

RETURN VALUES  This routine returns one of the following:
                E_OK                  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.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO  curses(3X), menus(3X), panel_update(3X), panels(3X), attributes(5)

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

3X-974        SunOS 5.6        modified 31 Dec 1996
NAME

menu_driver – command processor for the menus subsystem

SYNOPSIS

cc [flag ...] file ... -lmenu -lcurses [library ...]
#include <menu.h>
int menu_driver(MENU *menu, int c);

DESCRIPTION

menu_driver() is the workhorse of the menus subsystem. It checks to determine whether the character c is a menu request or data. If c is a request, the menu driver executes the request and reports the result. If c 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.

Menu driver requests:

- 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

menu_driver() 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_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.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO  curses(3X), menus(3X), attributes(5)

NOTES  Application defined commands should be defined relative to (greater than)
MAX_COMMAND, the maximum value of a request listed above.
The header <menu.h> automatically includes the headers <eti.h> and <curses.h>.
NAME
menu_format, set_menu_format – set and get maximum numbers of rows and columns in menus

SYNOPSIS
cc [ flag ... ] file ... -lmenu -lcurses [ library ... ]
#include <menu.h>
int set_menu_format(MENU *menu, int rows, int cols);
void menu_format(MENU *menu, int *rows, int *cols);

DESCRIPTION
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.

menu_format() returns the maximum number of rows and columns that may be displayed at one time on menu. rows and cols are pointers to the variables used to return these values.

RETURN VALUES
set_menu_format() 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_POSTED The menu is already posted.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
curses(3X), menus(3X), attributes(5)

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

modified 31 Dec 1996
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
cc [ flag ... ] file ... -lmenu -lcurses [ library ... ]
#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);

DESCRIPTION
set_item_init() assigns the application-defined function to be called when the menu is
posted and just after the current item changes. item_init() returns a pointer to the item
initialization routine, if any, called when the menu is posted and just after the current item
changes.

set_item_term() assigns an application-defined function to be called when the menu is
unposted and just before the current item changes. item_term() returns a pointer to the
termination function, if any, called when the menu is unposted and just before the current
item changes.

set_menu_init() assigns an application-defined function to be called when the menu 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 menu is posted and just
after the top row changes on a posted menu.

set_menu_term() assigns an application-defined function to be called when the menu 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 menu is unposted and
just before the top row changes on a posted menu.

RETURN VALUES
Routines that return pointers always return NULL on error. Routines that return an
integer return one of the following:
E_OK The routine returned successfully.
E_SYSTEM_ERROR System error.
ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO

curses(3X), menus(3X), attributes(5)

NOTES

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

menu_item_current, set_current_item, current_item, set_top_row, top_row, item_index –
set and get current menus items

SYNOPSIS

cc [ flag ... ] file ... -lm enu -lcurses [ library ... ]
#include <menu.h>
int set_current_item(MENU *menu, ITEM *item);
ITEM *current_item(MENU *menu);
int set_top_row(MENU *menu, int row);
int top_row(MENU *menu);
int item_index(ITEM *item);

DESCRIPTION

The current item of a menu is the item where the cursor is currently positioned.
set_current_item() sets the current item of menu to item. current_item() returns a pointer
to the the current item in menu.
set_top_row() sets the top row of menu to row. 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 menu.
item_index() returns the index to the item in the item pointer array. The value of this
index ranges from 0 through N-1, where N is the total number of items connected to the
menu.

RETURN VALUES

current_item() returns NULL on error.
top_row() and index_item() return -1 on error.
set_current_item() and set_top_row() 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_BAD_STATE The routine was called from an
initialization or termination function.
E_NOT_CONNECTED No items are connected to the menu.

ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO

curses(3X), menus(3X), attributes(5)

NOTES

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

3X-980 SunOS 5.6 modified 31 Dec 1996
NAME
menu_item_name, item_name, item_description – get menus item name and description

SYNOPSIS
cc [ flag ... ] file ... -lmenu -lcurses [ library ... ]
#include <menu.h>
char *item_name(ITEM *item);
char *item_description(ITEM *item);

DESCRIPTION
item_name() returns a pointer to the name of item.
item_description() returns a pointer to the description of item.

RETURN VALUES
These routines return NULL on error.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
curses(3X), menus(3X), menu_new(3X), attributes(5)

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

modified 31 Dec 1996 SunOS 5.6 3X-981
NAME menu_item_new, new_item, free_item – create and destroy menus items

SYNOPSIS cc [ flag … ] file … -lmenu -lcurses [ library … ]
#include <menu.h>
ITEM *new_item(char *name, char *desc);
int free_item(ITEM *item);

DESCRIPTION new_item() creates a new item from name and description, and returns a pointer to the new item.
free_item() frees the storage allocated for item. 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.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO curses(3X), menus(3X), attributes(5)

NOTES The header <menu.h> automatically includes the headers <eti.h> and <curses.h>.
NAME  menu_item_opts, set_item_opts, item_opts_on, item_opts_off, item_opts – menus item option routines

SYNOPSIS  cc [ flag ... ] file ... -lmenu -lcurses [ library ... ]
#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);

DESCRIPTION  set_item_opts() turns on the named options for item and turns off all other options. Options are boolean values that can be OR-ed together.
item_opts_on() turns on the named options for item; no other option is changed.
item_opts_off() turns off the named options for item; no other option is changed.
item_opts() returns the current options of item.

Item Options:
O_SELECTABLE The item can be selected during menu processing.

RETURN VALUES  Except for item_opts(), these routines return one of the following:
E_OK The routine returned successfully.
E_SYSTEM_ERROR System error.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO  curses(3X), menus(3X), attributes(5)

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

modified 31 Dec 1996  SunOS 5.6  3X-983
NAME  
menu_items, set_menu_items, item_count – connect and disconnect items to and from menus

SYNOPSIS  
cc [ flag ... ] file ... -lmenu -lcurses [ library ... ]
#include <menu.h>

int set_menu_items(MENU *menu, ITEM **items);
ITEM **menu_items(MENU *menu);
int item_count(MENU *menu);

DESCRIPTION  
set_menu_items() changes the item pointer array connected to menu to the item pointer array items.

menu_items() returns a pointer to the item pointer array connected to menu.

item_count() returns the number of items in menu.

RETURN VALUES  
menu_items() returns NULL on error.

item_count() returns -1 on error.

set_menu_items() 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_POSTED  The menu is already posted.
E_CONNECTED  One or more items are already connected to another menu.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO  
curses(3X), menus(3X), attributes(5)

NOTES  
The header <menu.h> automatically includes the headers <eti.h> and <curses.h>.
NAME    menu_item_userptr, set_item_userptr, item_userptr – associate application data with menus items

SYNOPSIS    cc [ flag ... ] file ... -lmenu -lcurses [ library ... ]
#include <menu.h>
int set_item_userptr(ITEM *item, char *userptr);
char *item_userptr(ITEM *item);

DESCRIPTION    Every item has an associated user pointer that can be used to store relevant information.
set_item_userptr() sets the user pointer of item. item_userptr() returns the user pointer of item.

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.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO    curses(3X), menus(3X), attributes(5)

NOTES    The header <menu.h> automatically includes the headers <eti.h> and <curses.h>.
NAME
menu_item_value, set_item_value, item_value — set and get menus item values

SYNOPSIS
cc [ flag ...] file ... -lmenu -l curses [ library ...]
#include <menu.h>
int set_item_value(ITEM *item, int bool);
int item_value(ITEM *item);

DESCRIPTION
Unlike single-valued menus, multi-valued menus enable the end-user to select one or more items from a menu. set_item_value() sets the selected value of the item — TRUE (selected) or FALSE (not selected). set_item_value() may be used only with multi-valued menus. To make a menu multi-valued, use set_menu_opts or menu_opts_off() to turn off the option O_ONEVALUE. (See menu_opts(3X)).

item_value() returns the select value of item, either TRUE (selected) or FALSE (unselected).

RETURN VALUES
set_item_value() returns one of the following:
E_OK The routine returned successfully.
E_SYSTEM_ERROR System error.
E_REQUEST_DENIED The menu driver could not process the request.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
curses(3X), menus(3X), menu_opts(3X), attributes(5)

NOTES
The header <menu.h> automatically includes the headers <eti.h> and <curses.h>.
### NAME
menu_item_visible, item_visible – tell if menus item is visible

### SYNOPSIS
```
cc [ flag ... ] file ... -lmenu -lcurses [ library ... ]
#include <menu.h>
int item_visible(ITEM *item);
```

### DESCRIPTION
A menu item is visible if it currently appears in the subwindow of a posted menu.

The `item_visible()` function returns `TRUE` if `item` is visible, otherwise it returns `FALSE`.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

### SEE ALSO
curses(3X), menus(3X), menu_new(3X), attributes(5)

### NOTES
The header `<menu.h>` automatically includes the headers `<eti.h>` and `<curses.h>`.

---

Modified 31 Dec 1996

SunOS 5.6

3X-987
NAME  menu_mark, set_menu_mark – menus mark string routines

SYNOPSIS  cc [ flag ... ] file ... -lmenu -lcurses [ library ... ]
#include <menu.h>
int set_menu_mark(MENU *menu, char *mark);
char *menu_mark(MENU *menu);

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 menu to mark.
menu_mark() returns a pointer to the mark string of menu.

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.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO  curses(3X), menus(3X), attributes(5)

NOTES  The header <menu.h> automatically includes the headers <eti.h> and <curses.h>.
NAME
menu_new, new_menu, free_menu – create and destroy menus

SYNOPSIS
cc [ flag ... ] file ... -lmenu -lcurses [ library ... ]
#include <menu.h>
MENU *new_menu(ITEM **items);
int free_menu(MENU *menu);

DESCRIPTION
new_menu() creates a new menu connected to the item pointer array items and returns a
pointer to the new menu.
free_menu() disconnects menu from its associated item pointer array and frees the
storage allocated for the menu.

RETURN VALUES
new_menu() returns NULL on error.
free_menu() 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_POSTED The menu is already posted.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
curses(3X), menus(3X), attributes(5)

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

modified 31 Dec 1996  SunOS 5.6  3X-989
NAME  menu_opts, set_menu_opts, menu_opts_on, menu_opts_off – menus option routines

SYNOPSIS  cc [ flag ... ] file ... -lmenu -lcurses [ library ... ]
            #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);

DESCRIPTION  Menu Options

set_menu_opts() turns on the named options for menu and turns off all other options. Options are boolean values that can be OR-ed together.

menu_opts_on() turns on the named options for menu; no other option is changed.

menu_opts_off() turns off the named options for menu; no other option is changed.

menu_opts() returns the current options of menu.

The following values can be OR’d together to create opts.

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  Except for menu_opts(), these routines return one of the following:
E_OK  The routine returned successfully.
E_SYSTEM_ERROR  System error.
E_POSTED  The menu is already posted.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO  curses(3X), menus(3X), attributes(5)

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

3X-990       SunOS 5.6       modified 31 Dec 1996
NAME menu_pattern, set_menu_pattern – set and get menus pattern match buffer

SYNOPSIS cc [ flag ... ] file ...-lmenu -lcurses [ library ... ]
#include <menu.h>
char *menu_pattern(MENU *menu);
int set_menu_pattern(MENU *menu, char *pat);

DESCRIPTION Every menu has a pattern buffer to match entered data with menu items. set_menu_pattern() sets the pattern buffer to pat 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 menu.

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.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO curses(3X), menus(3X), attributes(5)

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

modified 31 Dec 1996 SunOS 5.6 3X-991
NAME  menu_post, post_menu, unpost_menu – write or erase menus from associated subwindows

SYNOPSIS  cc [flag ...] file ... -lmenu -lcurses [ library ... ]
#include <menu.h>
int post_menu(MENU *menu);
int unpost_menu(MENU *menu);

DESCRIPTION  post_menu() writes menu 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.
unpost_menu() erases menu from its associated subwindow.

RETURN VALUES  These routines 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_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.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO  curses(3X), menus(3X), panels(3X), attributes(5)

NOTES  The header <menu.h> automatically includes the headers <eti.h> and <curses.h>.
NAME  menu_userptr, set_menu_userptr – associate application data with menus

SYNOPSIS  cc [ flag ... ] file ... -lmenu -lcurses [ library ... ]
#include <menu.h>

char *menu_userptr(MENU *menu);
int set_menu_userptr(MENU *menu, char *userptr);

DESCRIPTION  Every menu has an associated user pointer that can be used to store relevant information.
set_menu_userptr() sets the user pointer of menu. menu_userptr() returns the user pointer of menu.

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.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO  curses(3X), menus(3X), attributes(5)

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

modified 31 Dec 1996  SunOS 5.6  3X-993
NAME  menu_win, set_menu_win, set_menu_sub, menu_sub, scale_menu – menus window and subwindow association routines

SYNOPSIS  cc [ flag … ] file … −lmenu −lcurses [ library … ]
#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);

DESCRIPTION  set_menu_win() sets the window of menu to win. menu_win() returns a pointer to the window of menu.
set_menu_sub() sets the subwindow of menu to sub. menu_sub() returns a pointer to the subwindow of menu.
scale_window() returns the minimum window size necessary for the subwindow of menu. rows and cols are pointers to the locations used to return the values.

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_NOT_CONNECTED No items are connected to the menu.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO  curses(3X), menus(3X), attributes(5)

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

3X-994 SunOS 5.6 modified 31 Dec 1996
<table>
<thead>
<tr>
<th>NAME</th>
<th>meta – enable/disable meta keys</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYNOPSIS</td>
<td><code>#include &lt;curses.h&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>int meta(WINDOW *win, bool bf);</code></td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td><code>win</code> Is an ignored parameter.</td>
</tr>
<tr>
<td></td>
<td><code>bf</code> Is a Boolean expression.</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>Whether a terminal returns 7 or 8 significant bits initially depends on the control mode of the terminal driver. The <code>meta()</code> function forces the number of bits to be returned by <code>getch(3XC)</code> to be 7 (if <code>bf</code> is <code>FALSE</code>) or 8 (if <code>bf</code> is <code>TRUE</code>).</td>
</tr>
<tr>
<td></td>
<td>If the program handling the data can only pass 7-bit characters or strips the 8th bit, 8 bits cannot be handled.</td>
</tr>
<tr>
<td></td>
<td>If the <code>terminfo</code> capabilities <code>smm</code> (meta_on) and <code>rmm</code> (meta_off) are defined for the terminal, <code>smm</code> is sent to the terminal when <code>meta(win, TRUE)</code> is called, and <code>rmm</code> is sent when <code>meta(win, FALSE)</code> is called.</td>
</tr>
<tr>
<td></td>
<td>This function is useful when extending the non-text command set in applications where the META key is used.</td>
</tr>
<tr>
<td>RETURN VALUES</td>
<td>On success, the <code>meta()</code> function returns <code>OK</code>. Otherwise, it returns <code>ERR</code>.</td>
</tr>
<tr>
<td>ERRORS</td>
<td>None.</td>
</tr>
<tr>
<td>SEE ALSO</td>
<td><code>getch(3XC)</code></td>
</tr>
</tbody>
</table>

modified 1 Jun 1996
NAME  mkdirp, rmdirp – create, remove directories in a path

SYNOPSIS  
```c
cc [ flag ...] file ... -lgen [ library ...]
#include <libgen.h>
int mkdirp(const char *path, mode_t mode);
int rmdirp(char *dir, char *dir1);
```

DESCRIPTION  
mkdirp() creates all the missing directories in the given path with the given mode. See chmod(2) for the values of mode.

rmdirp() removes directories in path dir. This removal starts at the end of the path and moves back toward the root as far as possible. If an error occurs, the remaining path is stored in dir1. rmdirp() returns a 0 only if it is able to remove every directory in the path.

EXAMPLES  
```c
/* 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");
```

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.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  
mkdir(2), rmdir(2), attributes(5)

NOTES  
mkdirp() uses malloc(3C) to allocate temporary space for the string.

rmdirp() returns -2 if a "." or ".." is in the path and -3 if an attempt is made to remove the current directory. If an error occurs other than one of the above, -1 is returned.

When compiling multi-thread applications, the _REENTRANT flag must be defined on the compile line. This flag should only be used in multi-thread applications.
NAME  mkfifo – create a new FIFO

SYNOPSIS  #include <sys/types.h>
#include <sys/stat.h>

int mkfifo(const char *path, mode_t mode);

DESCRIPTION  The mkfifo() routine creates a new FIFO special file named by the pathname pointed to by path. The mode of the new FIFO is initialized from mode. The file permission bits of the mode argument are modified by the process’s file creation mask (see umask(2)). 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.

mkfifo() calls the mknod(2) function to make the file.

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.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  mkdir(1), chmod(2), exec(2), mknod(2), umask(2), fs_ufs(4), attributes(5), stat(5)

NOTES  Bits other than the file permission bits in mode are ignored.
mkstemp (3C)  C Library Functions

NAME  
mkstemp – make a unique file name

SYNOPSIS  
#include <stdlib.h>
int mkstemp(char *template);

DESCRIPTION  
The mkstemp() function replaces the contents of the string pointed to by template by a unique file name, and returns a file descriptor for the file open for reading and writing. The function thus prevents any possible race condition between testing whether the file exists and opening it for use. The string in template should look like a file name with six trailing ‘X’s; mkstemp() replaces each ‘X’ with a character from the portable file name character set. The characters are chosen such that the resulting name does not duplicate the name of an existing file.

RETURN VALUES  
Upon successful completion, mkstemp() returns an open file descriptor. Otherwise −1 is returned if no suitable file could be created.

ERRORS  
No errors are defined.

USAGE  
It is possible to run out of letters.

The mkstemp() function does not check to determine whether the file name part of template exceeds the maximum allowable file name length.

For portability with X/Open standards prior to XPG4v2, tmpfile(3S) is preferred over this function.

The mkstemp() function has an explicit 64-bit equivalent. See interface64(5).

SEE ALSO  
getpid(2), open(2), tmpfile(3S), tmpnam(3S), interface64(5), standards(5)
NAME
mktemp – make a unique file name

SYNOPSIS
#include <stdlib.h>
char *mktemp(char *template);

DESCRIPTION
mktemp() replaces the contents of the string pointed to by template with a unique file name, and returns template. The string in template should look like a file name with six trailing 'X's; mktemp() will replace the 'X's with a character string that can be used to create a unique file name.

RETURN VALUES
mktemp() will assign to template the empty string if it cannot create a unique name.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
mkstemp(3C), tmpfile(3S), tmpnam(3S), attributes(5)

NOTES
mktemp() can create only 26 unique file names per thread for each unique template.
NAME

mktime – converts a tm structure to a calendar time

SYNOPSIS

```c
#include <time.h>

time_t mktime(struct tm *timeptr);
```

DESCRIPTION

The `mktime()` function converts the time represented by the `tm` structure pointed to by `timeptr` into a calendar time (the number of seconds since 00:00:00 UTC, January 1, 1970).

The `tm` structure contains the following members:

```c
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 */
```

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.

The `tm_year` member 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.

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`.

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, `mktime()` attempts to determine whether the alternate timezone is in effect for the specified time.

Local timezone information is used as if `mktime()` had called `tzset()` (see `ctime(3C)`).
The **mktime()** function returns the specified calendar time. If the calendar time cannot be represented, the function returns the value $(\text{time}_t) - 1$.

**EXAMPLES**

What day of the week is July 4, 2001?

```c
#include <stdio.h>
#include <time.h>

static char *const wday[8] = {
    "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)==-(time_t)-1)
    time_str.tm_wday=7;

printf("%s\n", wday[time_str.tm_wday]);
```

**ATTRIBUTES**

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe with exceptions</td>
</tr>
</tbody>
</table>

**SEE ALSO**

ctime(3C), getenv(3C), TIMEZONE(4), attributes(5)

**NOTES**

The **mktime()** function is MT-Safe in multithread applications, as long as no user-defined function directly modifies one of the following variables: **timezone**, **altzone**, **daylight**, and **tzname**. See ctime(3C)).
NAME  mlock, munlock – lock (or unlock) pages in memory

SYNOPSIS  

#include <sys/types.h>

int mlock(caddr_t addr, size_t len);
int munlock(caddr_t addr, size_t len);

DESCRIPTION  

The function mlock() uses the mappings established for the address range \([addr, addr + len]\) 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.

munlock() removes locks established with mlock().

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.

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.

Locks established with mlock() are not inherited by a child process after a fork() and are not nested.

Because of the impact on system resources, the use of mlock() and munlock() is restricted to the super-user.

Attempts to mlock() more memory than a system-specific limit will fail.

RETURN VALUES  

Upon successful completion, the functions mlock() and munlock() return 0; otherwise, they return −1 and set errno to indicate the error.

ERRORS  

EINVAL  addr is not a multiple of the page size as returned by sysconf(3C).
ENOMEM  Addresses in the range \([addr, addr + len]\) are invalid for the address space of a process, or specify one or more pages which are not mapped.
EPERM  The process’s effective user ID is not superuser.

mlock(3C)  C Library Functions
ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO fork(2), memcntl(2), mmap(2), plock(3C), mlockall(3C), sysconf(3C), attributes(5)

NOTES mlock and munlock require super-user privileges.
mlockall (3C) C Library Functions

NAME
mlockall, munlockall – lock or unlock address space

SYNOPSIS
#include <sys/mman.h>

int mlockall(int flags);
int munlockall(void);

DESCRIPTION
The mlockall() function locks in memory all pages mapped by an address space.
The value of flags 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:

MCL_CURRENT Lock current mappings
MCL_FUTURE Lock future mappings

If MCL_FUTURE is specified for mlockall(), mappings are locked as they are added to the
address space (or replace existing mappings), provided sufficient memory is available.
Locking in this manner is not persistent across the exec family of functions (see exec(2)).
Mappings locked using mlockall() with any option may be explicitly unlocked with a
munlock() call (see mlock(3C)).
The munlockall() function removes address space locks and locks on mappings in the
address space.
All conditions and constraints on the use of locked memory that apply to mlock(3C) also
apply to mlockall().
Locks established with mlockall() are not inherited by a child process after a fork(2) call,
and are not nested.

RETURN VALUES
Upon successful completion, the functions mlockall() and munlockall() return 0; otherwise, they return −1
and set errno to indicate the error.

ERRORS
EAGAIN Some or all of the memory in the address space could not be locked due
to sufficient resources. This error condition applies to mlockall() only.
EINVAL The flags argument contains values other than MCL_CURRENT and
MCL_FUTURE.
EPERM The process’s effective user ID is not super-user.

USAGE
The mlockall() and munlockall() functions require super-user privileges.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
### SEE ALSO

exec(2), fork(2), memcntl(2), mmap(2), plock(3C), mlock(3C), sysconf(3C), attributes(5)
NAME
modf, modff – decompose floating-point number

SYNOPSIS
#include <math.h>

double modf(double x, double *iptr);

float modff(float x, float *iptr);

DESCRIPTION
The modf() and modff() functions break the argument x into integral and fractional parts, each of which has the same sign as the argument. modf() stores the integral part as a double in the object pointed to by iptr. modff() stores the integral part as a float in the object pointed to by iptr.

RETURN VALUES
Upon successful completion, modf() and modff() return the signed fractional part of x. If x is NaN, NaN is returned and *iptr is set to NaN.
If the correct value would cause underflow to 0.0, modf() returns 0 and errno may be set to ERANGE.

ERRORS
The modf() function may fail if:
ERANGE The result underflows.

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

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
frexp(3C), isnan(3M), ldexp(3C), attributes(5)
NAME

monitor – prepare process execution profile

SYNOPSIS

```c
#include <mon.h>

void monitor(int (*lowpc)(), int (*highpc)(), WORD *buffer, size_t bufsize,
             size_t nfunc);
```

DESCRIPTION

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().

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.

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 dirname, and monitor() is called automatically by compilation with cc -p, the file created is dirname/pid.progname where progname is the name of the program.

lowpc and highpc are the beginning and ending addresses of the region to be profiled.

buffer is the address of a user-supplied array of WORD (WORD is defined in the header <mon.h>). buffer is used by monitor() to store the histogram generated by profil() and the call counts.

bufsize identifies the number of array elements in buffer.

nfunc is the number of call count cells that have been reserved in buffer. Additional call count cells will be allocated automatically as they are needed.

bufsize should be computed using the following formula:

```c
size_of_buffer =
    sizeof(struct hdr) +
    nfunc * sizeof(struct cnt) +
    ((highpc-lowpc)/BARSIZE) * sizeof(WORD) +
    sizeof(WORD) - 1 ;

bufsize = (size_of_buffer / sizeof(WORD)) ;
```

where:

lowpc, highpc, nfunc are the same as the arguments to monitor();

BARSIZE is the number of program bytes that correspond to each histogram bar, or cell, of the profil() buffer;
the `hdr` and `cnt` structures and the type `WORD` are defined in the header `<mon.h>`.

The default call to `monitor()` is shown below:

```c
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:

```c
monitor((int (*)(()))0, (int (*)(()))0, (WORD *)0, 0, 0);
```

Use `prof` to examine the results.

### FILES
`mon.out`

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

### SEE ALSO
`cc(1B), profil(2), end(3C), attributes(5), prof(5)`

### NOTES
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     move, wmove – move cursor in window

SYNOPSIS  #include <curses.h>
           int move(int y, int x);
           int wmove(WINDOW *win, int y, int x);

ARGUMENTS  y     Is the y (row) coordinate of the position of the cursor in the window.
           x     Is the x (column) coordinate of the position of the cursor in the window.
           win   Is a pointer to the window in which the cursor is to be written.

DESCRIPTION  The move() function moves the logical cursor (for stdscr) to the position specified by y (row) and x (column), where the upper left corner of the window is row 0, column 0. The wmove() function performs the same action, but moves the cursor in the window specified by win. The physical cursor will not move until after a call to refresh(3XC) or doupdate(3XC).

RETURN VALUES  On success, these functions return OK. Otherwise, they return ERR.

ERRORS      None.

SEE ALSO    doupdate(3XC)
mp (3M) Mathematical Library

NAME

mp, mp_madd, mp_msub, mp_mult, mp_mdiv, mp_mcmp, mp_min, mp_mout,
mp_pow, mp_gcd, mp_rpow, mp_itom, mp_xtom, mp_mtox, mp_mfree – multiple pre-
ciaison integer arithmetic

SYNOPSIS

cc [ flag ...] file ... -lmp [ library ... ]
#include <mp.h>

void mp_madd(MINT *a, MINT *b, MINT *c);
void mp_msub(MINT *a, MINT *b, MINT *c);
void mp_mult(MINT *a, MINT *b, MINT *c);
void mp_mdiv(MINT *a, MINT *b, MINT *q, MINT *r);
int mp_mcmp(MINT *a, MINT *b);
int mp_min(MINT *a);
void mp_mout(MINT *a);
void mp_pow(MINT *a, MINT *b, MINT *c, MINT *d);
void mp_gcd(MINT *a, MINT *b, MINT *c);
int mp_msqrt(MINT *a, MINT *b, MINT *r);
void mp_sdiv(MINT *a, short n, MINT *q, short *r);
MINT * mp_itom(short n);
MINT * mp_xtom(char *a);
char * mp_mtox(MINT *a);
void mp_mfree(MINT *a);

DESCRIPTION

These routines perform arithmetic on integers of arbitrary length. The integers are stored
using the defined type MINT. Pointers to a MINT should be initialized using the function
mp_itom(n), which sets the initial value to n. Alternatively, mp_xtom(a) may be used to
initialize a MINT from a string of hexadecimal digits. mp_mfree(a) may be used to
release the storage allocated by the mp_itom(a) and mp_xtom(a) routines.

The mp_madd(a,b,c), mp_msub(a,b,c) and mp_mult(a,b,c) functions assign to their third
arguments the sum, difference, and product, respectively, of their first two arguments.
The mp_mdiv(a,b,q,r) function assigns the quotient and remainder, respectively, to its
third and fourth arguments. The mp_sdiv(a,n,q,r) function is similar to mp_mdiv(a,b,q,r)
except that the divisor is an ordinary integer. The mp_msqrt(a,b,r) function produces the
square root and remainder of its first argument. The mp_mcmp(a,b) function compares
the values of its arguments and returns 0 if the two values are equal, a value greater than
0 if the first argument is greater than the second, and a value less than 0 if the second
argument is greater than the first. The mp_rpow(a,n,b) function raises a to the n-th power
and assigns this value to b. The mp_pow(a,b,c,d) function raises a to the b-th power,
reduces the result modulo c and assigns this value to d. The mp_min(a) and
mp_mout(a) functions perform decimal input and output. The mp_gcd(a,b,c) function finds the greatest common divisor of the first two arguments, returning it in the third argument. The mp_mtox(a) function provides the inverse of mp_xtom(a). To release the storage allocated by mp_mtox(a), use free() (see malloc(3C)). Use the –lmp loader option to obtain access to these functions.

FILES
/usr/lib/libmp.a
/usr/lib/libmp.so

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
exp(3M), malloc(3C), libmp(4), attributes(5)

DIAGNOSTICS
Illegal operations and running out of memory produce messages and core images.

WARNINGS
The function pow() exists in both libmp and libm with widely differing semantics. This is why libmp.so.2 exists. libmp.so.1 exists solely for reasons of backward compatibility, and should not be used otherwise. Use the mp_*() functions instead. See libmp(4).
NAME  
 mq_close – close a message queue

SYNOPSIS  
 cc [ flag ... ] file ... -lposix4 [ library ... ]
 #include <mqueue.h>
 int mq_close(mqd_t mqdes);

DESCRIPTION 
 mq_close( ) removes the association between the message queue descriptor, mqdes, and
 its message queue.

If the process (or thread) has registered a notification request to the message queue via
this mqdes, this registration is removed and the message queue is available for another
process to attach for notification.

RETURN VALUES  
 Upon successful completion, mq_close( ) returns 0; otherwise, the function returns -1 and
 sets errno to indicate the error condition.

ERRORS  
 EBADF         mqdes is an invalid message queue descriptor.
 ENOSYS        sem_open( ) is not supported by this implementation.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  
 mq_notify(3R), mq_open(3R), mq_unlink(3R), attributes(5)
NAME  mq_notify – notify process (or thread) that a message is available on a queue

SYNOPSIS  cc [ flag . . . ] file . . . -lposix4 [ library . . . ]
#include <mqueue.h>

int mq_notify(mqd_t mqdes, const struct sigevent *notification);

struct sigevent {
  int sigev_notify; /* notification type */
  int sigev_signo; /* signal number */
  union sigval sigev_value; /* signal value */
};

union sigval {
  int sival_int; /* integer value */
  void *sival_ptr; /* pointer value */
};

DESCRIPTION  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).

If notification 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, mqdes. The notification specified by notification 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.

notification 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 notification->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 notification->sigev_notify is SIGEV_SIGNAL, then the signal specified in notification->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 notification->sigev_value will be the si_value component of the generated signal (see siginfo(5)).

If notification 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.

When the notification is sent to the registered process, its registration is removed. The message queue is then be available for registration.

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_notify(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.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  mq_close(3R), mq_open(3R), mq_receive(3R), mq_send(3R), attributes(5), siginfo(5)
NAME  mq_open – open a message queue

SYNOPSIS  cc [ flag ...] file ... -lposix4 [ library ... ]
#include <mqueue.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 */
    ...
};

DESCRIPTION  mq_open() establishes a connection to a named message queue, name, 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),
exi t(2), or exec(2).

name points to a string naming a message queue. The name argument must conform to
the construction rules for a path-name. If name is not the name of an existing message
queue and its creation is not requested, mq_open() fails and returns an error. The first
character of name must be a slash (/) character and the remaining characters of name cannot
include any slash characters. For maximum portability, name should include no more
than 14 characters, but this limit is not enforced.

oflag 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.

The value of oflag 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 oflag:

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() waits for message buffer space or a calling mq_receive() 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.

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

EINTR
The mq_open() operation was interrupted by a signal.
Realtime Library

mq_open(3R)

ERRNO

EINVAL
name is not a valid name.

O_CREAT was specified in oflag, the value of attr 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 name string exceeds PATH_MAX, or a pathname com-
ponent 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, name, 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.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
exec(2), exit(2), umask(2), mq_close(3R), mq_receive(3R), mq_send(3R), mq_setattr(3R),
mq_unlink(3R), sysconf(3C), attributes(5)

NOTES
In Solaris, message queues are based on shared memory. Although permissions to send
and receive messages are checked by the mq_receive() and mq_send() interfaces, any
application which can open the message queue can directly access the shared memory to
examine and manipulate messages in the queue. Thus message queues should not be
considered secure.

modified 30 Dec 1996

SunOS 5.6

3R-1017
NAME
mq_receive – receive a message from a message queue

SYNOPSIS
cc [ flag ... ] file ... -lposix4 [ library ... ]
#include <mqueue.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 */
    ...
};

DESCRIPTION
The mq_receive() function is used to receive the oldest of the highest priority message(s)
from the message queue specified by mqdes. If the size of the buffer in bytes, specified by
msg_len, is less than the mq_msgsize member of the message queue, the function fails
and returns an error. Otherwise, the selected message is removed from the queue and
copied to the buffer pointed to by msg_ptr.

If msg_prio is not NULL, the priority of the selected message is stored in the location refer-
enced by msg_prio.

If the specified message queue is empty and O_NONBLOCK is not set in the message
queue description associated with mqdes, (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 mqdes, no message is removed from the queue, and
mq_receive() returns an error.

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
The mq_receive() function will fail if:
EAGAIN O_NONBLOCK was set in the message description associated with
mqdes, and the specified message queue is empty.
EBADF The mqdes argument is not a valid message queue descriptor open for
reading.
EMGSIZE  The `msg_len` argument is less than the message size member of the message queue.
EINTR   The `mq_receive()` function operation was interrupted by a signal.
ENOSYS  The `mq_receive()` function is not supported by this implementation.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO `mq_open`(3R), `mq_send`(3R), `mq_setattr`(3R), `attributes`(5)
NAME
mq_send – send a message to a message queue

SYNOPSIS
cc [ flag ...] file ... -lpthread [ library ...]
#include <mqueue.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 */
    ...
};

DESCRIPTION
mq_send() adds the message pointed to by msg_ptr to the message queue specified by
mqdes. msg_len specifies the length of the message in bytes pointed to by msg_ptr. The
value of msg_len must be less than or equal to the mq_msgsize attribute of the message
queue, or mq_send() will fail.

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 msg_prio. A message with a
larger numeric value of msg_prio is inserted before messages with lower values of
msg_prio. A message is inserted after other messages in the queue, if any, with equal
msg_prio priority. The value of msg_prio must be greater than 0, and less than or equal to
MQ_PRIO_MAX.

If the specified message queue is full and if O_NONBLOCK is not set in the message
queue description associated with mqdes (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 wait-
ing 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 mqdes, the message is not queued, and mq_send() returns an
error.

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
EAGAIN The O_NONBLOCK flag is set in the message queue description associated
with mqdes, and the specified message queue is full.
EBADF mqdes is not a valid message queue descriptor open for writing.
EINTR A signal interrupted the call to mq_send()
EMSGSIZE The specified message length, msg_len, exceeds the message size attribute of
the message queue.

3R-1020 SunOS 5.6 modified 30 Dec 1996
ENOSYS  

`mq_send()` is not supported by this implementation.

**ATTRIBUTES**

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**SEE ALSO**  

`mq_open(3R), mq_receive(3R), mq_setattr(3R), sysconf(3C), attributes(5)`
NAME
mq_setattr, mq_getattr – set/get message queue attributes

SYNOPSIS
cc [ flag ...] file ... -lpthread4 [ library ...]
#include <mqueue.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 */
    ...
};

DESCRIPTION
mq_setattr() is used to set attributes associated with the message queue specified by
mqdes.

The message queue attributes corresponding to the following members defined in the
mq_attr structure are set to the specified values upon successful completion of
mq_setattr():

mq_flags The value of this member is either 0 or O_NONBLOCK.

The values of mq_maxmsg, mq_msgsize, and mq_curmsgs are ignored by mq_setattr().
If omqstat is non-NULL, mq_setattr() stores, in the location referenced by omqstat, the pre-
vious 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 mqdes.
Upon return, the mq_flags member of the mq_attr structure referenced by mqstat has the
value that was set when the message queue was created but also with modifications
made by subsequent mq_setattr() calls.

The following attributes were set at message queue creation:

mq_maxmsg
mq_msgsize

Upon return, the mq_curmsgs (the number of messages currently on the queue) member
of the mq_attr structure referenced by mqstat is set according to the current state of the
message queue.

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.
Realtime Library

**ERRORS**
- EBADF: *mqdes* is not a valid message queue descriptor.
- ENOSYS: *mq_setattr()* and *mq_getattr()* are not supported by this implementation.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**SEE ALSO**
- [mq_open(3R)](3R), [mq_receive(3R)](3R), [mq_send(3R)](3R), [attributes(5)](5)

modified 30 Dec 1996
NAME
mq_unlink – remove a message queue

SYNOPSIS
cc [ flag ... ] file ... -lposix4 [ library ... ]
#include <mqueue.h>
int mq_unlink(const char *name);

DESCRIPTION
mq_unlink() removes the message queue named by name. After a successful call to
mq_unlink() with name, a call to mq_open(3R) with the same name will fail if the flag
O_CREAT is not set in flags. If one or more processes have the message queue open when
mq_unlink() is called, destruction of the message queue is postponed until all references
to the message queue have been closed. Calls to mq_open(3R) to re-create the message
queue may fail until the message queue is actually removed. However, mq_unlink()
does not block (wait) until all references have been closed; it returns immediately.

RETURN VALUES
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.

ERRORS
EACCESS Permission is denied to unlink the named message queue.
ENAMETOOLONG The length of the name string exceeds PATH_MAX, or a pathname com-
ponent is longer than NAME_MAX while _POSIX_NO_TRUNC is in effect.
ENOENT The named message queue, name, does not exist.
ENOSYS mq_unlink() is not supported by this implementation.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
mq_close(3R), mq_open(3R), attributes(5)
NAME
msync — synchronize memory with physical storage

SYNOPSIS
#include <sys/mman.h>
int msync(void *addr, size_t len, int flags);

DESCRIPTION
The msync() function writes all modified copies of pages over the range [addr, addr + len) to
the underlying hardware, or invalidates any copies so that further references to the
pages will be obtained by the system from their permanent storage locations. The per-
manent storage for a modified MAP_SHARED mapping is the file the page is mapped to;
the permanent storage for a modified MAP_PRIVATE mapping is its swap area.
The flags argument is a bit pattern built from the following values:

- MS_ASYNC perform asynchronous writes
- MS_SYNC perform synchronous writes
- MS_INVALIDATE invalidate mappings

If flags is MS_ASYNC or MS_SYNC, the function synchronizes the file contents to match
the current contents of the memory region.

- All write references to the memory region made prior to the call are visible by subse-
  quent read operations on the file.
- All writes to the same portion of the file prior to the call may or may not be visible by
  read references to the memory region.
- Unmodified pages in the specified range are not written to the underlying hardware.

If flags is MS_ASYNC, the function may return immediately once all write operations are
scheduled; if flags is MS_SYNC, the function does not return until all write operations are
completed.

If flags is MS_INVALIDATE, the function synchronizes the contents of the memory region
to match the current file contents.

- All writes to the mapped portion of the file made prior to the call are visible by subse-
  quent read references to the mapped memory region.
- All write references prior to the call, by any process, to memory regions mapped to the
  same portion of the file using MAP_SHARED, are visible by read references to the
  region.

If msync() causes any write to the file, then the file’s st_ctime and st_mtime fields are
marked for update.

RETURN VALUES
Upon successful completion, msync() returns 0; otherwise, it returns −1 and sets errno to
indicate the error.

ERRORS
The msync() function will fail if:

- EBUSY Some or all of the addresses in the range [addr, addr + len) are locked and
  MC_SYNC with the MS_INVALIDATE option is specified.
- EINVAL The addr argument is not a multiple of the page size as returned by

modified 28 May 1997
SunOS 5.6
3C-1025
msync(3C).

sysconf(3C).
The flags argument is not some combination of MS_ASYNC and MS_INVALIDATE.

EIO
An I/O error occurred while reading from or writing to the file system.

ENOMEM
Addresses in the range [addr, addr + len) are outside the valid range for the address space of a process, or specify one or more pages that are not mapped.

EPERM
MS_INVALIDATE was specified and one or more of the pages is locked in memory.

USAGE
The msync() function should be used by programs that require a memory object to be in a known state, for example in building transaction facilities. Normal system activity can cause pages to be written to disk. Therefore, there are no guarantees that msync() is the only control over when pages are or are not written to disk.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
memcntl(2), mmap(2), sysconf(3C), attributes(5)
**NAME**
mutex, pthread_mutex_init, pthread_mutex_lock, pthread_mutex_trylock,
pthread_mutex_unlock, pthread_mutex_destroy, mutex_init, mutex_lock, mutex_trylock,
mutex_unlock, mutex_destroy – mutual exclusion locks

**SYNOPSIS**

**POSIX**
cc [ flag ... ] file ... -lpthread [ library ... ]
#include <pthread.h>
int pthread_mutex_init(pthread_mutex_t *mp, const pthread_mutexattr_t *attr);

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
int pthread_mutex_lock(pthread_mutex_t *mp);
int pthread_mutex_trylock(pthread_mutex_t *mp);
int pthread_mutex_unlock(pthread_mutex_t *mp);
int pthread_mutex_destroy(pthread_mutex_t *mp);

**Solaris**
cc [ flag ... ] file ... -lthread [ library ... ]
#include <thread.h>
#include <synch.h>
int mutex_init(mutex_t *mp, int type, void *arg);
int mutex_lock(mutex_t *mp);

int mutex_trylock(mutex_t *mp);
int mutex_unlock(mutex_t *mp);
int mutex_destroy(mutex_t *mp);

**DESCRIPTION**
Mutual exclusion locks (mutexes) prevent multiple threads from simultaneously executing critical sections of code which access shared data (that is, mutexes are used to serialize the execution of threads). All mutexes must be global. A successful call for a mutex lock via pthread_mutex_lock() or mutex_lock() will cause another thread that is also trying to lock the same mutex to block until the owner thread unlocks it via
pthread_mutex_unlock() or mutex_unlock(). Threads within the same process or within other processes can share mutexes.

Mutexes can synchronize threads within the same process or in other processes. Mutexes can be used to synchronize threads between processes if the mutexes are allocated in writable memory and shared among the cooperating processes (see mmap(2)), and have been initialized for this task.

**Initialize**
Mutexes are either intra-process or inter-process, depending upon the argument passed implicitly or explicitly to the initialization of that mutex. A statically allocated mutex does not need to be explicitly initialized; by default, a statically allocated mutex is initialized with all zeros and its scope is set to be within the calling process. For POSIX portability of statically allocated mutexes, use the pthread_mutex_initializer macro (see below).

For inter-process synchronization, a mutex needs to be allocated in memory shared between these processes. Since the memory for such a mutex must be allocated dynamically, the mutex needs to be explicitly initialized using mutex_init() or
POSIX Initialize

POSIX mutexes, threads, and condition variables use attributes objects in the same manner; they are initialized with the configuration of an attributes object (see `pthread_mutexattr_init(3T)`). The `pthread_mutex_init()` function initializes the mutex referenced by `mp` with attributes specified by `attr`. If `attr` is `NULL`, the default mutex attributes are used, which is the same as passing the address of a default mutex attributes object. Upon initialization, the state of the mutex is initialized and unlocked. If default mutex attributes are used, then only threads created within the same process can operate on the initialized mutex variable.

In POSIX, the attributes of a mutex may be specified via the attribute object created via `pthread_mutexattr_init()` and modified using the `pthread_mutexattr_*(3T)` functions. To explicitly specify whether a mutex is or is not shared between processes, it can be initialized with an attribute object modified via `pthread_mutexattr_setpshared(3T)`. The second argument to this function can be either of the following:

- `PTHREAD_PROCESS_PRIVATE`: The mutex can synchronize threads within this process. The `PTHREAD_PROCESS_PRIVATE` POSIX mutex type for process scope is equivalent to the `USYNC_THREAD` flag to `mutex_init()` in the Solaris API (see below).

- `PTHREAD_PROCESS_SHARED`: The mutex can synchronize threads in this process and other processes. Only one process should initialize the mutex. The `PTHREAD_PROCESS_SHARED` POSIX mutex type for system-wide scope is equivalent to the `USYNC_PROCESS` flag to `mutex_init()` in the Solaris API (see below). The object initialized with this attribute must be allocated in memory shared between processes, either in System V shared memory (see `shmop(2)`) or in memory mapped to a file (see `mmap(2)`). It is illegal to initialize the object this way and to not allocate it in such shared memory.

Initializing mutexes can also be accomplished by allocating in zeroed memory (default), in which case, `PTHREAD_PROCESS_PRIVATE` is assumed. The same mutex must not be simultaneously initialized by multiple threads, nor should a mutex lock be re-initialized while in use by other threads.

If default mutex attributes are used, statically allocated mutexes can be initialized by the macro `PTHREAD_MUTEX_INITIALIZER`. The effect is the same as a dynamic initialization by a call to `pthread_mutex_init()` with parameter `attr` specified as `NULL`, except error checks are not performed.

Default mutex initialization (intra-process):

```c
pthread_mutex_t mp;
pthread_mutexattr_t mattr;

pthread_mutex_init(&mp, NULL);
```
OR
pthread_mutexattr_init(&mattr);
pthread_mutex_init(&mp, &mattr);
OR
pthread_mutexattr_setpshared(&mattr, PTHREAD_PROCESS_PRIVATE);
pthread_mutex_init(&mp, &mattr);
OR
pthread_mutex_t mp = PTHREAD_MUTEX_INITIALIZER;
OR
pthread_mutex_t mp;
mp = calloc (1, sizeof (pthread_mutex_t));

Customized mutex initialization (inter-process):

pthread_mutexattr_init(&mattr);
pthread_mutexattr_setpshared(&mattr, PTHREAD_PROCESS_SHARED);
pthread_mutex_init(&mp, &mattr);

Solaris Initialize

The equivalent Solaris API used to initialize a mutex so that it has several different types
of behavior is the type argument passed to mutex_init(). No current type uses arg
although a future type may specify additional behavior parameters via arg. type may be
one of the following:

USYNC_THREAD The mutex can synchronize threads only in this process. arg is
ignored. The USYNC_THREAD Solaris mutex type for process
scope is equivalent to the POSIX mutex attribute setting
PTHREAD_PROCESS_PRIVATE.

USYNC_PROCESS The mutex can synchronize threads in this process and other
processes. Only one process should initialize the mutex. arg is
ignored. The USYNC_PROCESS Solaris mutex type for process
scope is equivalent to the POSIX mutex attribute setting
PTHREAD_PROCESS_SHARED. The object initialized with this
attribute must be allocated in memory shared between processes,
either in System V shared memory (see shmop(2)). or in memory
mapped to a file (see mmap(2)). It is illegal to initialize the object
this way and to not allocate it in such shared memory.

Initializing mutexes can also be accomplished by allocating in zeroed memory (default),
in which case, a type of USYNC_THREAD is assumed. The same mutex must not be
simultaneously initialized by multiple threads. A mutex lock must not be re-initialized
while in use by other threads.

If default mutex attributes are used, the macro DEFAULTMUTEX can be used to initialize
mutexes that are statically allocated.

Default mutex initialization (intra-process):

mutex_t mp;
mutex_init(&mp, NULL, NULL);

OR
mutex_init(&mp, USYNC_THREAD, NULL);

OR
mutex_t mp = DEFAULTMUTEX;

OR
mutex_t mp;

mp = calloc(1, sizeof(mutex_t));

OR
mutex_t mp;

mp = malloc(sizeof(mutex_t));

memset(mp, 0, sizeof(mutex_t));

Customized mutex initialization (inter-process):
mutex_init(&mp, USYNC_PROCESS, NULL);

Lock and Unlock

A critical section of code is enclosed by a the call to lock the mutex and the call to unlock the mutex to protect it from simultaneous access by multiple threads. Only one thread at a time may possess mutually exclusive access to the critical section of code that is enclosed by the mutex-locking call and the mutex-unlocking call, whether the mutex’s scope is intra-process or inter-process. A thread calling to lock the mutex either gets exclusive access to the code starting from the successful locking until its call to unlock the mutex, or it waits until the mutex is unlocked by the thread that locked it.

Mutexes have ownership, unlike semaphores. Although any thread, within the scope of a mutex, can get an unlocked mutex and lock access to the same critical section of code, only the thread that locked a mutex can unlock it.

If a thread waiting for a mutex receives a signal, upon return from the signal handler, the thread resumes waiting for the mutex as if there was no interrupt. A mutex protects code, not data; therefore, strongly bind a mutex with the data by putting both within the same structure, or at least within the same procedure.

POSIX/Solaris Locking

A call to pthread_mutex_lock() or mutex_lock() locks the mutex object referenced by mp. If the mutex is already locked, the calling thread blocks until the mutex is freed; this will return with the mutex object referenced by mp in the locked state with the calling thread as its owner. If the current owner of a mutex tries to relock the mutex, it will result in deadlock.

pthread_mutex_trylock() and mutex_trylock() is the same as pthread_mutex_lock() and mutex_lock(), respectively, except that if the mutex object referenced by mp is locked (by any thread, including the current thread), the call returns immediately with an error.
**Threads Library**

**Destroy**

Either `pthread_mutex_destroy()` or `mutex_destroy()` destroys the mutex object referenced by `mp`; the mutex object becomes uninitialized. The space used by the destroyed mutex variable is not freed. It needs to be explicitly reclaimed.

**RETURN VALUES**

If successful, all of these functions return 0; otherwise, an error number is returned.

`pthread_mutex_trylock()` or `mutex_trylock()` returns 0 if a lock on the mutex object referenced by `mp` is obtained; otherwise, an error number is returned.

**ERRORS**

These functions fail and return the corresponding value if any of the following conditions are detected:

- **EFAULT** `mp` or `attr` points to an illegal address.
- **EINVAL** The value specified by `mp` or `attr` is invalid.
- **EBUSY** The mutex pointed to by `mp` was already locked.

**EXAMPLES**

**Single Gate**

The following example uses one global mutex as a gate-keeper to permit each thread exclusive sequential access to the code within the user-defined function "change_global_data." This type of synchronization will protect the state of shared data, but it also prohibits parallelism.

```c
/* cc thisfile.c -lthread */
#define _REENTRANT
#include <stdio.h>
#include <thread.h>
#define NUM_THREADS 12

void *change_global_data(void *); /* for thr_create() */
main(int argc,char * argv[]) {
    int i=0;
    for (i=0; i< NUM_THREADS; i++) {
        thr_create(NULL, 0, change_global_data, NULL, 0, NULL);
    }
    while ((thr_join(NULL, NULL, NULL) == 0));
}
```

modified 8 May 1997

SunOS 5.6

3T-1031
The previous example, the mutex, the code it owns, and the data it protects was enclosed in one function. The next example uses C++ features to accommodate many functions that use just one mutex to protect one data:

```c++
/* CC thisfile.c -lthread use C++ to compile */
#define _REENTRANT
#include <stdlib.h>
#include <stdio.h>
#include <thread.h>
#include <errno.h>
#include <iostream.h>
#define NUM_THREADS 16

void *change_global_data(void *null) { /* for thr_create() */
    class Mutected {
        private:
            static mutex_t Global_mutex;
            static int Global_data;
        public:
            static int add_to_global_data(void);
            static int subtract_from_global_data(void);
    }
    int Mutected::Global_data = 0;
    mutex_t Mutected::Global_mutex;
    int Mutected::add_to_global_data() {
        mutex_lock(&Global_mutex);
        Global_data++;
        mutex_unlock(&Global_mutex);
        return Global_data;
    }
}
```

Multiple Instruction
Single Data

Single Instruction
Multiple Data

The previous example, the mutex, the code it owns, and the data it protects was enclosed in one function. The next example uses C++ features to accommodate many functions that use just one mutex to protect one data:

```c
/* CC thisfile.c -lthread use C++ to compile */
#include <stdlib.h>
#include <stdio.h>
#include <thread.h>
#include <errno.h>
#include <iostream.h>
#define _REENTRANT
#define NUM_THREADS 16

void *change_global_data(void *null) { /* for thr_create() */
    class Mutected {
        private:
            static mutex_t Global_mutex;
            static int Global_data;
        public:
            static int add_to_global_data(void);
            static int subtract_from_global_data(void);
    }
    int Mutected::Global_data = 0;
    mutex_t Mutected::Global_mutex;
    int Mutected::add_to_global_data() {
        mutex_lock(&Global_mutex);
        Global_data++;
        mutex_unlock(&Global_mutex);
        return Global_data;
    }
}
```
void subtract_from_global_data() {
    mutex_lock(&Global_mutex);
    Global_data--;
    mutex_unlock(&Global_mutex);
    return Global_data;
}

void main(int argc, char * argv[]) {
    int i=0;
    for (i=0; i<NUM_THREADS; i++) {
        thr_create(NULL, 0, change_global_data, NULL, 0, NULL);
    }
    while ((thr_join(NULL, NULL, NULL) == 0));
}

void * change_global_data(void *) {
    static int switcher = 0;
    if ((switcher++ % 3) == 0) /* one-in-three threads subtracts */
        cout << subtract_from_global_data() << endl;
    else
        cout << add_to_global_data() << endl;
    return NULL;
}

Interprocess Locking

A mutex can protect data that is shared among processes. The mutex would need to be initialized as either PTHREAD_PROCESS_SHARED for POSIX (see pthread_mutexattr_init(3T), or USYNC_PROCESS for Solaris threads. One process initializes the process-shared mutex and writes it to a file to be mapped into memory by all cooperating processes (see mmap(2)). Afterwards, other independent processes can run the same program (whether concurrently or not) and share mutex-protected data.

To execute, run the command line "a.out 0 & a.out 1" */

#define _REENTRANT
#include <sys/types.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <thread.h>
#define INTERPROCESS_FILE "ipc-sharedfile"
#define NUM_ADDTHREADS 12
#define NUM_SUBTRACTTHREADS 10

modified 8 May 1997
```c
#define INCREMENT '0'
#define DECREMENT '1'
typedef struct {
    mutex_t     Interprocess_mutex;
    int         Interprocess_data;
} buffer_t;
buffer_t *buffer;

void *add_interprocess_data(), *subtract_interprocess_data();
void create_shared_memory(), test_argv();
int zeroed[sizeof(buffer_t)];
int ipc_fd, i=0;

void
main(int argc,char * argv[])  
  {
    test_argv(argv[1]);

    switch (*argv[1])  
      {
    case INCREMENT:
      create_shared_memory();
      ipc_fd = open(INPROCESS_FILE, O_RDWR);
      buffer = (buffer_t *)mmap(NULL, sizeof(buffer_t),
                                 PROT_READ| PROT_WRITE, MAP_SHARED, ipc_fd, 0);
      buffer->Interprocess_data = 0;
      mutex_init(&buffer->Interprocess_mutex,USYNC_PROCESS,0);
      for (i=0; i< NUM_ADDTHREADS; i++)
        thr_create(NULL, 0, add_interprocess_data, argv[1],
                    0, NULL);
      break;

    case DECREMENT:
      while((ipc_fd = open(INPROCESS_FILE, O_RDWR)) == -1)
        sleep(1);
      buffer = (buffer_t *)mmap(NULL, sizeof(buffer_t),
                                 PROT_READ| PROT_WRITE, MAP_SHARED, ipc_fd, 0);
      for (i=0; i< NUM_SUBTRACTTHREADS; i++)
        thr_create(NULL, 0, subtract_interprocess_data, argv[1],
                   0, NULL);
      break;
      } /* end switch */

    while ((thr_join(NULL,NULL,NULL) == 0));
  } /* end main */
```
void *add_interprocess_data(char argv_1[])
{
    mutex_lock(&buffer->Interprocess_mutex);
    buffer->Interprocess_data++;
    sleep(2);
    printf("%d is add-interprocess data, and %c is argv1\n",
           buffer->Interprocess_data, argv_1[0]);
    mutex_unlock(&buffer->Interprocess_mutex);
    return NULL;
}

void *subtract_interprocess_data(char argv_1[])
{
    mutex_lock(&buffer->Interprocess_mutex);
    buffer->Interprocess_data--;
    sleep(2);
    printf("%d is subtract-interprocess data, and %c is argv1\n",
           buffer->Interprocess_data, argv_1[0]);
    mutex_unlock(&buffer->Interprocess_mutex);
    return NULL;
}

void create_shared_memory()
{
    int i;
    ipc_fd = creat(INTERPROCESS_FILE, O_CREAT|O_RDWR);
    for (i=0; i<sizeof(buffer_t); i++)
    {
        zeroed[i] = 0;
        write(ipc_fd, &zeroed[i],2);
    }
    close(ipc_fd);
    chmod(INTERPROCESS_FILE, S_IRWXU|S_IRWXG|S_IRWXO);
}

void test_argv(char argv1[])
{
    if (argv1 == NULL) {
        printf("use 0 as arg1 for initial process\n or use 1 as arg1 for the second process\n");
        exit(NULL);
    }
}

In this example, run the command line

    a.out 0 & a.out 1

Dynamically Allocated Mutexes

The following example allocates and frees memory in which a mutex is embedded.
struct record {
    int field1;
    int field2;
    mutex_t m;
} *r;

r = malloc(sizeof(struct record));
mutex_init(&r->m, USYNC_THREAD, NULL);
/*
 * The fields in this record are accessed concurrently
 * by acquiring the embedded lock.
 */

The thread execution in this example is as follows:

Thread 1 executes: Thread 2 executes:
...
mutex_lock(&r->m); mutex_lock(&r->m);
r->field1++; localvar = r->field1;
r->field2 += 2; r->field2 += 3;
mutex_unlock(&r->m); mutex_unlock(&r->m);
...

Later, when a thread decides to free the memory pointed to by r, the thread should call
mutex_destroy() on the mutexes in this memory.

In the following example, the main thread can do a thr_join() on both of the above
threads. If there are no other threads using the memory in r, the main thread can now
safely free r:

for (i = 0; i < 2; i++)
    thr_join(0, 0, 0);
mutex_destroy(&r->m); /* first destroy mutex */
free(r); /* Then free memory */

If the mutex is not destroyed, the program could have memory leaks.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
mmap(2), shmop(2), pthread_create(3T), pthread_mutexattr_init(3T), attributes(5), standards(5)

NOTES
Currently, the only supported policy is SCHED_OTHER. In Solaris, under the
SCHED_OTHER policy, there is no established order in which threads are unblocked.

In the current implementation of threads, pthread_mutex_lock(),
pthread_mutex_unlock(), mutex_lock(), mutex_unlock(), pthread_mutex_trylock(),
and mutex_trylock() do not validate the mutex type. Therefore, an uninitialized mutex or
a mutex with an invalid type does not return EINVAL. Interfaces for mutexes with an invalid type have unspecified behavior.

Uninitialized mutexes which are allocated locally may contain junk data. Such mutexes need to be initialized using pthread_mutex_init() or mutex_init().

By default, if multiple threads are waiting for a mutex, the order of acquisition is undefined.
<table>
<thead>
<tr>
<th>NAME</th>
<th>mvcur – move the cursor</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYNOPSIS</td>
<td><code>#include &lt;curses.h&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>int mvcur(int oldrow, int oldcol, int newrow, int newcol);</code></td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td><code>oldrow</code> Is the row from which cursor is to be moved.</td>
</tr>
<tr>
<td></td>
<td><code>oldcol</code> Is the column from which cursor is to be moved.</td>
</tr>
<tr>
<td></td>
<td><code>newrow</code> Is the row to which cursor is to be moved.</td>
</tr>
<tr>
<td></td>
<td><code>newcol</code> Is the column to which cursor is to be moved.</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>The <code>mvcur()</code> function is a low-level function used only outside of X/Open Curses when the program has to deal directly with the <code>terminfo</code> database to handle certain terminal capabilities. The use of appropriate X/Open Curses functions is recommended in all other situations, so that X/Open Curses can track the cursor.</td>
</tr>
<tr>
<td></td>
<td>The <code>mvcur()</code> function moves the cursor from the location specified by <code>oldrow</code> and <code>oldcol</code> to the location specified by <code>newrow</code> and <code>newcol</code>. A program using this function must keep track of the current cursor position.</td>
</tr>
<tr>
<td>RETURN VALUES</td>
<td>On success, the <code>mvcur()</code> function returns OK. Otherwise, it returns ERR.</td>
</tr>
<tr>
<td>ERRORS</td>
<td>None.</td>
</tr>
</tbody>
</table>
NAME  mvderwin – map area of parent window to subwindow

SYNOPSIS  
```
#include <curses.h>

int mvderwin(WINDOW *win, int par_y, int par_x);
```

ARGUMENTS  
- `win`: Is a pointer to the window to be mapped.
- `par_y`: Is the y (row) coordinate of the placement of the upper left corner of window relative to the parent window.
- `par_x`: Is the x (column) coordinate of the placement of the upper left corner of the window relative to the parent window.

DESCRIPTION  
The `mvderwin()` function defines a mapped area of `win`’s parent window that is the same size as `win` and has its upper left corner at position `par_y, par_x` of the parent window. Whenever `win` is refreshed, its contents are updated to match those of the mapped area and any reference to characters in `win` is treated as a reference to corresponding characters in the mapped area.

RETURN VALUES  
On success, the `mvderwin()` function returns `OK`. Otherwise, it returns `ERR`.

ERRORS  
None.

SEE ALSO  
`delwin(3XC), derwin(3XC)`
NAME  
mvprintw, mvwprintw, printw, vw_printw, wprintw – write formatted output to window

SYNOPSIS  
#include <curses.h>

int mvprintw(int y, int x, char *fmt [,arg...]);
int mvwprintw(WINDOW *win, int y, int x, char *fmt [,arg...])
int printw(char *fmt [,arg...]);
int vwprintw(WINDOW *win, char *fmt, void *arglist);
int vw_printw(WINDOW *win, char *fmt, void *arglist);
int wprintw(WINDOW *win, char *fmt [,arg...]);

ARGUMENTS  
y Is the y (row) coordinate position of the string’s placement in the window.
x Is the x (column) coordinate position of the string’s placement in the window.
fmt [,arg...] Is a printf() format string where arg is zero or more parameters used to satisfy the printf() string.
win Is a pointer to the window in which the string is to be written.
fmt, arglist Is a vprintf() format string where arglist is a pointer to a list of parameters. The vwprintw() function requires a variable parameter list as defined in <varargs.h>. The vw_printw() function requires a variable parameter list as defined in <stdarg.h>.

DESCRIPTION  
These functions are functionally equivalent to printf(3S). Their effect is similar to using sprintf(3S) to format the string and then using waddstr(3XC) to add that string to a window.

With printw() and wprintw(), the string is written to stdscr and win, respectively. The mvprintw() and mvwprintw() functions position the cursor as specified in stdscr or win, respectively, and then call printw().

The vwprintw() and vw_printw() functions are similar to wprintw() but use a pointer to a variable parameter list as defined by either <varargs.h> or <stdarg.h>. Each application must include the appropriate header.

RETURN VALUES  
On success, these functions return OK. Otherwise, they return ERR.

ERRORS  
None.

SEE ALSO  
addnstr(3XC), mvscanw(3XC), printf(3S), sprintf(3S)

3XC-1040  SunOS 5.6  modified 1 Jun 1996
NAME
  mvscanw, mvwscanw, scanw, vw_scanw, vwscanw, wscanw – read formatted input
  from window

SYNOPSIS
#include <curses.h>
int mvscanw(int y, int x, char *fmt[,arg...]);
int mvwscanw(WINDOW *win, int y, int x, char *fmt[,arg...])
int scanw(char *fmt[,arg...]);
int vwscanw(WINDOW *win, char *fmt, void *arglist);
int vw_scanw(WINDOW *win, char *fmt, void *arglist);
int wscanw(WINDOW *win, char *fmt[,arg...]);

ARGUMENTS
  y    Is the y (row) coordinate of the position of the character to be read.
  x    Is the x (column) coordinate of the position of the character to be read.
  fmt is a vwscanw() format string; arg is zero or more parameters used to satisfy
  the scanf() string.
  win  Is a pointer to the window in which the character is to be read.
  fmt, arglist
    fmt is a scanf() format string; arglist is a pointer to zero or more parameters used
    to satisfy the scanf() string. The vwprintw() function requires a variable
    parameter list as defined in <varargs.h>. The vw_printw() function requires a
    variable parameter list as defined in <stdarg.h>.

DESCRIPTION
These functions are functionally equivalent to scanf(3S). Characters are read from
the window using the getstr(3XC) set of functions. When a newline is received, the line is
processed by scanw() which places the result in the appropriate
args.

With scanw() and wscanw(), the characters are read from stdscr and win, respectively.
The mvscanw() and mvwscanw() functions position the cursor in the window and then
call scanw().

The vwscanw() and vw_scanw() functions are similar to wscanw() but use a pointer to a
variable parameter list as defined by either <varargs.h> or <stdarg.h>. Each application
must include the appropriate header.

RETURN VALUES
On success, these functions return OK. Otherwise, they return ERR.

ERRORS
None

SEE ALSO
getnstr(3XC), mvprintw(3XC), scanf(3S)
NAME  mvwin – move window

SYNOPSIS  
```c
#include <curses.h>
int mvwin(WINDOW *win, int y, int x);
```

ARGUMENTS  
- `win`  Is a pointer to the window to move.
- `y`  Is the y (row) coordinate of the upper left corner of the window.
- `x`  Is the x (column) coordinate of the upper left corner of the window.

DESCRIPTION  The `mvwin()` function moves the specified window (or subwindow), placing its upper left corner at the positions specified by `x` and `y`. The entire window must fit within the physical boundaries of the screen or an error results. In the case of a subwindow, the window must remain within the boundaries of the parent window.

RETURN VALUES  On success, the `mvwin()` function returns `OK`. Otherwise, it returns `ERR`.

ERRORS  None.

SEE ALSO  derwin(3XC)
NAME

nanosleep – high resolution sleep

SYNOPSIS

cc [ flag ...] file ... -lposix4 [ library ...]
#include <time.h>

int nanosleep(const struct timespec *rqtp, struct timespec *rmtgp);

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

DESCRIPTION

nanosleep() suspends the current thread from execution until either the time interval
specified by rqtp 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 multi-
ple 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 rqtp, as measured by the system clock, CLOCK_REALTIME.

nanosleep() will not block nor effect the action of any signal.

RETURN VALUES

If nanosleep() returns because the requested time has elapsed, it returns 0. Otherwise, if
it returns because it has been interrupted by a signal:

    It returns -1 and sets errno to indicate the interruption.

    If rmtgp is non-NULL, the timespec structure referenced by rmtgp will be updated to
    contain the remaining amount of time between rqtp and the time actually slept.

If any of the following error conditions occur, nanosleep() returns -1 and sets errno to
indicate the error condition.

ERRORS

EINTR    nanosleep() was interrupted by a signal.
EINVAL   rqtp specified a nanosecond value less than zero or greater than or equal to
          1,000,000,000.
ENOSYS   nanosleep() is not supported by this implementation.

ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO

sleep(3C), attributes(5)

modified 30 Dec 1996        SunOS 5.6        3R-1043
NAME
napms – sleep process for a specified length of time

SYNOPSIS
#include <curses.h>
int napms(int ms);

ARGUMENTS
ms Is the number of milliseconds to sleep.

DESCRIPTION
The napms() function sleeps for at least ms milliseconds.

RETURN VALUES
The napms() function always returns OK.

ERRORS
None.

SEE ALSO
delay_output(3XC)
NAME
netdir, netdir_getbyname, netdir_getbyaddr, netdir_free, netdir_options, taddr2uaddr,
uaddr2taddr, netdir_perror, netdir_sperror, netdir_mergeaddr – generic transport
name-to-address translation

SYNOPSIS
#include <netdir.h>

int netdir_getbyname(const struct netconfig *config,
    const struct nd_hostserv *service, struct nd_addrlst **addrs);

int netdir_getbyaddr(const struct netconfig *config,
    struct nd_hostservlist **service,
    const struct netbuf *netaddr);

void netdir_free(void *ptr, const int struct_type);

int netdir_options(const struct netconfig *config, const int option,
    const int fields, char *point_to_args);

char *taddr2uaddr(const struct netconfig *config, const struct netbuf *addr);

struct netbuf *uaddr2taddr(const struct netconfig *config, const char *uaddr);

void netdir_perror(char *s);

char *netdir_sperror(void);

DESCRIPTION
These routines provide a generic interface for name-to-address mapping that will work
with all transport protocols. This interface provides a generic way for programs to con-
vert transport specific addresses into common structures and back again. The
netconfig
structure, described on the netconfig(4) manual page, identifies the transport.

The netdir_getbyname() routine maps the machine name and service name in the
nd_hostserv structure to a collection of addresses of the type understood by the transport
identified in the netconfig structure. This routine returns all addresses that are valid for
that transport in the nd_addrlst structure. The nd_hostserv structure contains the fol-
lowing members:

    char *h_host; /* host name */
    char *h_serv; /* service name */

The nd_addrlst structure contains the following members:

    int n_cnt; /* number of addresses */
    struct netbuf *n_addrs;

netdir_getbyname() accepts some special-case host names. The host names are defined
in <netdir.h>. The currently defined host names are:

HOST_SELF Represents the address to which local programs will bind their
endpoints. HOST_SELF differs from the host name provided by
gethostname(3C), which represents the address to which remote
programs will bind their endpoints.
HOST_ANY Represents any host accessible by this transport provider. HOST_ANY allows applications to specify a required service without specifying a particular host name.

HOST_SELF_CONNECT Represents the host address that can be used to connect to the local host.

HOST_BROADCAST Represents the address for all hosts accessible by this transport provider. Network requests to this address will be received by all machines.

All fields of the nd_hostserv structure must be initialized.

To find the address of a given host and service on all available transports, call the netdir_getbyname() routine with each struct netconfig structure returned by getnetconfig(3N).

The netdir_getbyaddr() routine maps addresses to service names. This routine returns service, a list of host and service pairs that would yield this address. If more than one tuple of host and service name is returned, then the first tuple contains the preferred host and service names:

```
struct nd_hostservlist {
    int *h_cnt; /* number of hostservs found */
    struct hostserv *h_hostservs;
}
```

The netdir_free() structure is used to free the structures allocated by the name to address translation routines. ptr points to the structure that has to be freed. The struct_type identifies the structure:

```
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. fildes is the associated file descriptor. option, fildes, 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. fildes 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 fildes.

**ND_SET_RESERVEDPORT**
Allows the application to bind to a reserved port, if that concept exists for the transport provider. fildes is an unbound file descriptor into the transport. If pointer_to_args is NULL, fildes 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. fildes is not used. pointer_to_args is a pointer to a netbuf structure that contains the address. This option returns 0 only if the address specified in pointer_to_args is reserved.

**ND_MERGEADDR**
Used to take a “local address” (like the 0.0.0.0 address that TCP uses) and return a “real address” that client machines can connect to. fildes is not used. pointer_to_args is a pointer to a struct nd_mergearg, which has the following members:

- char s_uaddr; /* server’s universal address */
- char c_uaddr; /* client’s universal address */
- char m_uaddr; /* the result */

If s_uaddr is something like 0.0.0.0.1.12, and, if the call is successful, m_uaddr will be set to something like 192.11.109.89.1.12. For most transports, m_uaddr is exactly what s_uaddr is.

**RETURN VALUES**
The netdir_perror() routine prints an error message on the standard output stating why one of the name-to-address mapping routines failed. The error message is preceded by the string given as an argument.

The netdir_sperror() routine returns a string containing an error message stating why one of the name-to-address mapping routines failed.

netdir_sperror() returns a pointer to a buffer which contains the error message string. This buffer is overwritten on each call. In multithreaded applications, this buffer is implemented as thread-specific data.
ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO gethostname(3C), getnetconfig(3N), getnetpath(3N), netconfig(4), attributes(5)
### NAME

newpad, pnoutrefresh, prefresh, subpad – create or refresh a pad or subpad

### SYNOPSIS

```c
#include <curses.h>

WINDOW *newpad(int nlines, int ncols);
int pnoutrefresh(WINDOW *pad, int pminrow, int pmincol,
                 int sminrow, int smincol, int smaxrow, int smaxcol);
int prefresh(WINDOW *pad, int pminrow, int pmincol,
             int sminrow, int smincol, int smaxrow, int smaxcol);
WINDOW *subpad(WINDOW *orig, int nlines, int ncols);
```

### ARGUMENTS

- **nlines**: Is the number of lines in the pad to be created.
- **ncols**: Is the number of columns in the pad to be created.
- **pad**: Is a pointer to the pad to refresh.
- **pminrow**: Is the row coordinate of the upper left corner of the pad rectangle to be copied.
- **pmincol**: Is the column coordinate of the upper left corner of the pad rectangle to be copied.
- **sminrow**: Is the row coordinate of the upper left corner of the rectangle on the physical screen where pad is to be positioned.
- **smincol**: Is the column coordinate of the upper left corner of the rectangle on the physical screen where pad is to be positioned.
- **smaxrow**: Is the row coordinate of the lower right corner of the rectangle on the physical screen where pad is to be positioned.
- **smaxcol**: Is the column coordinate of the lower right corner of the rectangle on the physical screen where pad is to be positioned.
- **orig**: Is a pointer to the parent pad within which a sub-pad is created.

### DESCRIPTION

The `newpad()` function creates a new pad with the specified number of lines and columns. A pointer to the new pad structure is returned. A pad differs from a window in that it is not restricted to the size of the physical screen. It is useful when only part of a large window will be displayed at any one time.

Automatic refreshes by scrolling or echoing of input do not take place when pads are used. Pads have their own refresh commands, `prefresh()` and `pnoutrefresh()`.

The `prefresh()` function copies the specified portion of the logical pad to the terminal screen. The parameters `pmincol` and `pminrow` specify the upper left corner of the rectangular area of the pad to be displayed. The lower right coordinate of the rectangular area of the pad that is to be displayed is calculated from the screen parameters (`sminrow`, `smincol`, `smaxrow`, `smaxcol`).

This function calls the `pnoutrefresh()` function to copy the specified portion of `pad` to the terminal screen and the `doupdate(3XC)` function to do the actual update. The logical cursor is copied to the same location in the physical window unless `leaveok(3XC)` is enabled.

---

modified 1 Jun 1996  SunOS 5.6  3XC-1049
(in which case, the cursor is placed in a position that the program finds convenient).

When outputting several pads at once, it is often more efficient to call the `pnoutrefresh()` and `doupdate()` functions directly. A call to `pnoutrefresh()` for each pad first, followed by only one call to `doupdate()` to update the screen, results in one burst of output, fewer characters sent, and less CPU time used.

The `subpad()` function creates a sub-pad within the pad `orig` with the specified number of lines and columns. A pointer to the new pad structure is returned. The sub-pad is positioned in the middle of `orig`. Any changes made to one pad affect the other. `touchwin(3XC)` or `touchline(3XC)` will likely have to be called on pad `orig` to correctly update the window.

**RETURN VALUES**

On success, the `newpad()` and `subpad()` functions returns a pointer to the new pad data structure. Otherwise, they return a null pointer.

On success, the `pnoutrefresh()` and `prefresh()` functions return `OK`. Otherwise, they return `ERR`.

**SEE ALSO** `clearok(3XC)`, `doupdate(3XC)`, `is_linetouched(3XC)`, `pechochar(3XC)`
NAME
nextafter – next representable double-precision floating-point number

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

DESCRIPTION
The nextafter() function computes the next representable double-precision floating-point value following x in the direction of y. Thus, if y is less than x, nextafter() returns the largest representable floating-point number less than x.

RETURN VALUES
The nextafter() function returns the next representable double-precision floating-point value following x in the direction of y.
If x or y is NaN, then nextafter() returns NaN.
If x is finite and the correct function value would overflow, nextafter() returns ±HUGE_VAL (according to the sign of x) and sets errno to ERANGE.

ERRORS
The nextafter() function will fail if:
ERANGE  The correct value would overflow.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
attributes(5),

modified 29 Dec 1996
NAME
nice – change priority of a process

SYNOPSIS
/usr/ucb/cc [ flag ... ] file ...
int nice(incr)
int incr;

DESCRIPTION
The scheduling priority of the process is augmented by incr. 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.

Negative increments are illegal, except when specified by the privileged user. The prior-
ity 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.

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).

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 incr specified was negative, and the effective user ID is not the
privileged user.

SEE ALSO
nice(1), renice(1), fork(2), priocntl(2), getpriority(3C)

NOTES
Use of these interfaces should be restricted to only applications written on BSD plat-
forms. Use of these interfaces with any of the system libraries or in multi-thread applica-
tions is unsupported.

3B-1052 SunOS 5.6 modified 12 Feb 1993
Network Functions

NAME
nis_db, db_initialize, db_create_table, db_destroy_table, db_first_entry, db_next_entry, db_reset_next_entry, db_list_entries, db_remove_entry, db_add_entry, db_table_exists, db_unload_table, db_checkpoint, db_standby, db_free_result - NIS+ Database access functions

SYNOPSIS
cc [ flag ...] file... -lnisdb -lnsl [ library... ]
#include <rpcsvc/nis.h>
#include <rpcsvc/nis_db.h>
bool db_initialize(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 *);

DESCRIPTION
These functions describe the interface between the NIS+ server and the underlying database. They are defined in the shared library /usr/lib/libnisdb.so.

The interface is a simple subset of a complete relational database and provides just those items that are needed by the NIS+ server daemon. When you replace the database, your interface routines should match these exactly. Also note that the database is responsible for verifying that the objects passed do not exceed the internal limits of the database being used.

The database's performance will directly affect the performance of the server. The default information base that is provided with NIS+ is the Structured Storage Manager (SSM). This is a memory based database that has been tuned for NIS+.

modified 30 Dec 1996

SunOS 5.6

3N-1053
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 XDRRed 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:

```c
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:

```c
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.
**nis_db (3N)**

**Network Functions**

**db_checkpoint()** organizes the contents of the table in a more efficient manner. Checkpointing may mean different things to different types of databases. It does not affect the logical contents of the table — operations and queries should return the same result before and after a checkpoint. For example, in a log-based system, checkpointing may mean incorporating log entries of updates accumulated since the previous checkpoint into the table.

**db_free_result()** frees up the space allocated by various functions listed on this manual page that return a `db_result` structure.

**db_standby()** is an advisory call to the database manager. This call informs the database that activity has slowed down and it can free up unnecessary resources such as file descriptors.

**PROGRAMMING**

Most of the routines in this library use an NIS+ name to identify the object that the user desires. The name must be in canonical form before being passed to the database because one server may be serving several namespaces and discrimination of the requested objects is accomplished by comparing the domain names.

**DIAGNOSTICS**

**DB_SUCCESS** The query or operation completed successfully and returned status.

**DB_NOTFOUND** The name or entry that was named in the argument did not exist.

**DB_NOTUNIQUE** An attempt was made to remove an entry from a table that is not uniquely specified.

**DB_BADQUERY** The query that was submitted to the database was invalid (for example, it might name some nonexistent fields).

**DB_BADTABLE** The table was corrupted.

**DB_BADOBJECT** The fields of the object does not conform to the fields of the table to which it is being added.

**DB_MEMORY_LIMIT** There is insufficient memory to complete the operation requested.

**DB_STORAGE_LIMIT** There is insufficient file storage available to complete the operation requested.

**DB_INTERNAL_ERROR** An internal error was encountered during the execution of the operation requested (either a programming error or an unrecoverable exception).
### ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

### SEE ALSO

`rpc.nisd(1M), nis_objects(3N), nisfiles(4), attributes(5)`
NAME

nis_error, nis_sperrno, nis_perror, nis_lerror, nis_sperror, nis_sperror_r – display NIS+ error messages

SYNOPSIS

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_lerror(const nis_error status, const char *label);

char *nis_sperror_r(nis_error status, char *label, char *buf, int length);

char *nis_sperror(const nis_error status, const char *label);

DESCRIPTION

These functions convert NIS+ status values into text strings.

nis_sperrno() simply returns a pointer to a string constant which is the error string.

nis_perror() prints the error message corresponding to status as “label: error message” on standard error.

nis_lerror() sends the error text to syslog(3) at level LOG_ERR.

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). status and label are the same as for nis_perror(). The pointer returned by nis_sperror_r() is the same as buf, that is, the pointer returned by the function is a pointer to buf. length specifies the number of characters to copy from the error string to buf.

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.

ATTRIBUTES

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

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO

niserror(1), string(3C), syslog(3), attributes(5)

NOTES

When compiling multithreaded applications, see Intro(3), Notes On Multithread Applications, for information about the use of the _REENTRANT flag.
Network Functions

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

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);

DESCRIPTION

These functions manipulate NIS+ groups. They are used by NIS+ clients and servers, and are the interfaces to the group authorization object.
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.

There are three types of group members:

- An **explicit** member is just a NIS+ principal-name, for example "wickedwitch.west.oz."
- An **implicit** ("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 not included.
- A **recursive** ("group") member, written "@cowards.oz.", refers to another group; all principals that belong to that group are considered to belong here.

Any member may be made **negative** by prefixing it with a minus sign ('-'). A group may thus contain explicit, implicit, recursive, negative explicit, negative implicit, and negative recursive members.

A principal is considered to belong to a group if it belongs to at least one non-negative group member of the group and belongs to no negative group members.

The nis_ismember() function returns TRUE if it can establish that principal belongs to group; otherwise it returns FALSE.

The nis_addmember() and nis_removemember() functions add or remove a member. They do not check whether the member is valid. The user must have read and modify rights for the group in question.

modified 30 Dec 1996 SunOS 5.6
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

```c
bool_var = nis_ismember("lion.oz.", "sadsouls.oz.");
```

will return 1 (TRUE) and the function call

```c
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

```c
bool_var = nis_ismember("hogan.monkeys.west.oz.", "baddies.oz.");
```

will return 1 (TRUE) because any principal from the `monkeys.west.oz.` domain belongs to the implicit group `*.monkeys.west.oz.`., but the function call

```c
bool_var = nis_ismember("hogan.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

```c
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.

**ATRIBUTES**

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**SEE ALSO**

`nisgrpadm(1)`, `nis_objects(3N)`, `attributes(5)`

**NOTES**

These functions only accept fully-qualified NIS+ names.

A group is represented by a NIS+ object (see `nis_objects(3N)`) with a variant part that is defined in the `group_obj` structure. It contains the following fields:
```c
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.

There are currently no known methods for `nis_ismember()`, `nis_print_group_entry()`, and `nis_verifygroup()` to get their answers from only the master server.
NAME
nis_local_names, nis_local_directory, nis_local_host, nis_local_group, nis_local_principal
− NIS+ local names

SYNOPSIS
cc [ flag . . . ] file. . . −lnsl [ library. . . ]
#include <rpcsvc/nis.h>
nis_name nis_local_directory(void);
nis_name nis_local_host(void);
nis_name nis_local_group(void);
nis_name nis_local_principal(void);

DESCRIPTION
These functions return several default NIS+ names associated with the current process.
nis_local_directory() returns the name of the NIS+ domain for this machine. This is
currently the same as the Secure RPC domain returned by the sysinfo(2) system call.
nis_local_host() returns the NIS+ name of the current machine. This is the fully quali®ed
name for the host and is either the value returned by the gethostname(3C) function or, if
the host name is only partially quali®ed, 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 quali®ed.
nis_local_group() returns the name of the current NIS+ group name. This is currently set
by setting the environment variable NIS_GROUP to the groupname.
nis_local_principal() returns the NIS+ principal name for the user associated with the
effective UID of the calling process. This function maps the effective uid into a principal
name by looking for a LOCAL type credential (see nisaddcred(1M)) in the table named
cred.org_dir in the default domain.
Note: The result returned by these routines is a pointer to a data structure with the NIS+
library, and should be considered a “read-only” result and should not be modified.

ENVIRONMENT
NIS_GROUP
This variable contains the name of the local NIS+ group. If the name is
not fully quali®ed, the value returned by nis_local_directory() will be
concatenated to it.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
nisdefaults(1), nisaddcred(1M), sysinfo(2), gethostname(3C), nis_names(3N),
nis_objects(3N), attributes(5)
Network Functions

NAME

nis_names, nis_lookup, nis_add, nis_remove, nis_modify, nis_freeresult – NIS+ namespace functions

SYNOPSIS

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);

DESCRIPTION

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).

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.

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.

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 must 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)).

nis_lookup() takes two parameters, the name of the object to be resolved in name, and a flags parameter, flags, 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 only function from this group that can use a non-fully qualified name. If the parameter name is not a fully qualified name, then the flag EXPAND_NAME must be specified in the call. If this flag is not specified, the function will fail with the error NIS_BADNAME.

The flags parameter is constructed by logically ORing zero or more flags from the following list.

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 either a object is found that is not a LINK type object, or the library has followed 16 links.

modified 30 Dec 1996
SunOS 5.6
3N-1063
### 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:

```c
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_sperrno() (see nis_error(3N)).

The objects structure contains two members. objects_val is an array of nis_object structures; objects_len is the number of cells in the array. These objects will be freed by the call to nis_freeresult(). If you need to keep a copy of one or more objects, they can be copied with the function nis_clone_object() and freed with the function nis_destroy_object() (see nis_server(3N)). Refer to nis_objects(3N) for a description of the nis_object structure.

The various ticks contain details of where the time was taken during a request. They can be used to tune one’s data organization for faster access and to compare different database implementations (see nis_db(3N)).

zticks The time spent in the NIS+ service itself. This count starts when the server receives the request and stops when it sends the reply.

dticks The time spent in the database backend. This time is measured from the time a database call starts, until the result is returned. If the request results in multiple calls to the database, this is the sum of all the time spent in those calls.

aticks The time spent in any “accelerators” or caches. This includes the time required to locate the server needed to resolve the request.

cticks The total time spent in the request. This clock starts when you enter the client library and stops when a result is returned. By subtracting the sum of the other ticks values from this value, you can obtain the local overhead of generating a NIS+ request.
Subtracting the value in \textit{dticks} from the value in \textit{zticks} will yield the time spent in the service code itself. Subtracting the sum of the values in \textit{zticks} and \textit{aticks} from the value in \textit{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 \texttt{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 \textit{expired}. The time to live value has gone to zero and the object may have changed. If the flag \texttt{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 \texttt{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 \textit{add}, \textit{remove}, and \textit{modify} 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 \texttt{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 obj 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 LINK 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().

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO nis_error(3N), nis_objects(3N), nis_server(3N), nis_subr(3N), nis_tables(3N), attributes(5)

NOTES You cannot modify the name of an object if that modification would cause the object to reside in a different domain.
You cannot modify the schema of a table object.
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 \texttt{nis\_object} structure contains the following members:

\begin{verbatim}
typedef char *nis_name;
struct nis_object {  
nis_oid zo_oid;
nis_name zo_name;
nis_name zo_owner;
nis_name zo_group;
nis_name zo_domain;
u_long zo_access;
u_long zo_ttl;
objdata zo_data;
};
\end{verbatim}

In this structure, the first member \texttt{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.

The second member, \texttt{zo\_name}, contains the leaf name of the object. This name is never 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.

\texttt{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 \texttt{zo\_name} and \texttt{zo\_oid}, it uniquely identifies an object. This makes it possible to always reconstruct the name of an object by using the code fragment

\begin{verbatim}
sprintf(buf,"%s.%s", obj->zo_name, obj->zo_domain);
\end{verbatim}

The \texttt{zo\_owner} and \texttt{zo\_group} members contain the NIS+ names of the object’s principal owner and group owner, respectively. Both names must be NIS+ fully qualified names. However, neither name can be used directly to identify the object they represent. This stems from the condition that NIS+ uses itself to store information that it exports.
The **zo_owner** member contains a fully qualified NIS+ name of the form `principal.domain`. This name is called a NIS+ principal name and is used to identify authentication information in a credential table. When the server constructs a search query of the form

```
[cname=principal],cred.org_dir.domain.
```

The query will return to the server credential information about `principal` for all flavors of RPC authentication that are in use by that principal. When an RPC request is made to the server, the authentication flavor is extracted from the request and is used to find out the NIS+ principal name of the client. For example, if the client is using the `AUTH_DES` authentication flavor, it will include in the authentication credentials the network name or `netname` of the user making the request. This netname will be of the form `unix.UID@domain`.

The NIS+ server will then construct a query on the credential database of the form

```
[auth_name=netname,auth_type=AUTH_DES],cred.org_dir.domain.
```

This query will return an entry which contains a principal name in the first column. This NIS+ principal name is used to control access to NIS+ objects.

The group owner for the object is treated differently. The group owner member is optional (it should be the null string if not present) but must be fully qualified if present. A group name takes the form `group.domain` which the server then maps into a name of the form `group.groups_dir.domain`.

The purpose of this mapping is to prevent NIS+ group names from conflicting with user specified domain or table names. For example, if a domain was called `engineering.foo.com`, then without the mapping a NIS+ group of the same name to represent members of engineering would not be possible. The contents of groups are lists of NIS+ principal names which are used exactly like the `zo_owner` name in the object. See `nis_groups(3N)` for more details.

The **zo_access** member contains the bitmask of access rights assigned to this object. There are four access rights defined, and four are reserved for future use and must be zero. This group of 8 access rights can be granted to four categories of client. These categories are the object's owner, the object's group owner, all authenticated clients (world), and all unauthenticated clients (nobody). Note that access granted to “nobody” is really access granted to everyone, authenticated and unauthenticated clients.

The **zo_ttl** member contains the number of seconds that the object can “live” in a cache before it is expired. This value is called the time to live for this object. This number is particularly important on group and directory (domain) objects. When an object is cached, the current time is added to the value in **zo_ttl**. Then each time the cached object is used, the time in **zo_ttl** is compared with the current time. If the current time is later than the time in **zo_ttl** the object is said to have expired and the cached copy should not be used.
Setting the TTL is somewhat of an art. You can think of it as the “half life” of the object, or half the amount of time you believe will pass before the object changes. The benefit of setting the ttl to a large number is that the object will stay in a cache for long periods of time. The problem with setting it to a large value is that when the object changes it will take a long time for the caches to flush out old copies of that object. The problems and benefits are reversed for setting the value to a small number. 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:

```c
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.

```c
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.
The first type of object is the directory object. This object’s variant part is defined as follows:

```c
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 {
```
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 the 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 twotiered. 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.

```c
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.

```c
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
#define TA_MODIFIED 32

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 \texttt{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 \texttt{ta_cols} contains two members. \texttt{ta_cols_val} is an array of \texttt{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 \texttt{ta_cols_len}. \texttt{ta_maxcol} also contains the number of columns in the table and always has the same value as \texttt{ta_cols_len}. Once the table is created, this length field cannot be changed.

The \texttt{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 \texttt{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 \texttt{FOLLOW_PATH} with a \texttt{nis_list()} call. See \texttt{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.

The TA_MODIFIED flag is set only when the table column is modified. When TA_MODIFIED is set, and the object is modified again, the modified access rights for the table column must be copied, not the default access rights.

Entry Objects

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_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  cc [ flag . . . ] file. . . -lnsl [ library. . . ]
#include <rpcsvc/nis.h>
void nis_ping(const nis_name dirname, const u_long utime, const nis_object *dirobj);
nis_result *nis_checkpoint(const nis_name dirname);

DESCRIPTION  nis_ping() is called by the master server for a directory when a change has occurred within that directory. The parameter dirname identifies the directory with the change. If the parameter dirobj is NULL, this function looks up the directory object for dirname and uses the list of replicas it contains. The parameter utime 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.

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.

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.

nis_checkpoint() returns a pointer to a nis_result 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.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  nislog(1M), nis_names(3N), nis_tables(3N), nisfiles(4), attributes(5)
Network Functions

NAME

nis_server, nis_mkdir, nis_rmdir, nis_servstate, nis_stats, nis_getservlist, nis_freeservlist, nis_freetags − miscellaneous NIS+ functions

SYNOPSIS

cc [ flag . . . ] file. . . − lnsl [ library. . . ]
#include <rpcsvc/nis.h>
nis_error nis_mkdir(const nis_name dirname, const nis_server *machine);
nis_error nis_rmdir(const nis_name dirname, const nis_server *machine);
nis_error nis_servstate(const nis_server *machine, const nis_tag *tags, const int numtags, nis_tag **result);
nis_error nis_stats(const nis_server *machine, const nis_tag *tags, const int numtags, nis_tag **result);
void nis_freetags(nis_tag *tags, const int numtags);
nis_server **nis_getservlist(const nis_name dirname);
void nis_freeservlist(nis_server *machine);

DESCRIPTION

These functions provide a variety of services for NIS+ applications.
nis_mkdir() is used to create the necessary databases to support NIS+ service for a directory, dirname, on a server, machine. If this operation is successful, it means that the directory object describing dirname has been updated to reflect that server machine is serving the named directory. For a description of the nis_server structure, refer to nis_objects(3N).
nis_rmdir() is used to delete the directory, dirname, from the specified machine. The machine parameter cannot be NULL. For a description of the nis_server structure, refer to nis_objects(3N).
nis_servstate() is used to set and read the various state variables of the NIS+ servers. In particular the internal debugging state of the servers may be set and queried. The nis_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 nisstat(1M).
nis_servstate() and nis_stats() use the tag list. This tag list is a variable length array of nis_tag structures whose length is passed to the function in the numtags parameter. The set of legal tags are defined in the file <rpcsvc/nis_tags.h> which is included in <rpcsvc/nis.h>. Because these tags can and do vary between implementations of the NIS+ service, it is best to consult this file for the supported list. Passing unrecognized tags to a server will result in their tag_value member being set to the string “unknown.” Both of these functions return their results in malloced tag structure, *result. If there is an error, *result is set to NULL. The tag_value pointers points to allocated string memory which contains the results. Use nis_freetags() to free the tag structure.

modified 30 Dec 1996

SunOS 5.6

3N-1079
nis_getservlist() returns a null terminated list of nis_server structures that represent the list of servers that serve the domain named dirname. Servers from this list can be used when calling functions that require the name of a NIS+ server. For a description of the nis_server structure, refer to nis_objects(3N). nis_freeservlist() frees the list of servers returned by nis_getservlist(). Note that this is the only legal way to free that list.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO nisstat(1M), nis_names(3N), nis_objects(3N), nis_subr(3N), attributes(5)
NAME
nis_subr, nis_leaf_of, nis_name_of, nis_domain_of, nis_getnames, nis_freenames,
nis_dir_cmp, nis_clone_object, nis_destroy_object, nis_print_object – NIS+ subroutines

SYNOPSIS
cc [ flag . . . ] file . . . -lnsl [ library. . . ]
#include <rpcsvc/nis.h>
nis_name nis_leaf_of(const nis_name name);
nis_name nis_name_of(const nis_name name);
nis_name nis_domain_of(const nis_name name);
nis_name nis_getnames(const nis_name name);
void nis_freenames(nis_name *namelist);
namelist nis_dir_cmp(const nis_name n1, const nis_name n2);
nis_object nis_clone_object(const nis_object *src, nis_object *dest);
void nis_destroy_object(nis_object *obj);
void nis_print_object(const nis_object *obj);

DESCRIPTION
These subroutines are provided to assist in the development of NIS+ applications. They provide several useful operations on both NIS+ names and objects.

The first group, nis_leaf_of(), nis_domain_of(), and nis_name_of() provide the functions for parsing NIS+ names. nis_leaf_of() will return the first label in an NIS+ name. It takes into account the double quote character ‘”’ which can be used to protect embedded ‘.’ (dot) characters in object names. Note that the name returned will never have a trailing dot character. If passed the global root directory name ‘.”’, it will return the null string.

nis_domain_of() returns the name of the NIS+ domain in which an object resides. This name will always be a fully qualified NIS+ name and ends with a dot. By iteratively calling nis_leaf_of() and nis_domain_of() it is possible to break a NIS+ name into its individual components.

nis_name_of() is used to extract the unique part of a NIS+ name. This function removes from the tail portion of the name all labels that are in common with the local domain. Thus if a machine were in domain foo.bar.baz and nis_name_of() were passed a name bob.friends.foo.bar.baz, then nis_name_of() would return the unique part, bob.friends. If the name passed to this function is not in either the local domain or one of its children, this function will return null.

nis_getnames() will return a list of candidate names for the name passed in as name. If this name is not fully qualified, nis_getnames() will generate a list of names using the default NIS+ directory search path, or the environment variable NIS_PATH if it is set. The returned array of pointers is terminated by a NULL pointer, and the memory associated with this array should be freed by calling nis_freenames().

modified 30 Dec 1996

SunOS 5.6

3N-1081
Though \texttt{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 \texttt{name\_pos}. When the names passed to this function are identical, the function returns a value of \texttt{SAME\_NAME}. If the name \texttt{n1} is a direct ancestor of name \texttt{n2}, then this function returns the result \texttt{HIGHER\_NAME}. Similarly, if the name \texttt{n1} is a direct descendant of name \texttt{n2}, then this function returns the result \texttt{LOWER\_NAME}. When the name \texttt{n1} is neither a direct ancestor nor a direct descendant of \texttt{n2}, as it would be if the two names were siblings in separate portions of the namespace, then this function returns the result \texttt{NOT\_SEQUENTIAL}. Finally, if either name cannot be parsed as a legitimate name then this function returns the value \texttt{BAD\_NAME}.

The second set of functions, consisting of \texttt{nis\_clone\_object()} and \texttt{nis\_destroy\_object()}, are used for manipulating objects. \texttt{nis\_clone\_object()} creates an exact duplicate of the NIS+ object \texttt{src}. If the value of \texttt{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 \texttt{nis\_objects(3N)} for a description of the \texttt{nis\_object} structure.

\texttt{nis\_destroy\_object()} can be used to destroy an object created by \texttt{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 \texttt{dest} parameter to \texttt{nis\_clone\_object()}) then the object cannot be freed with this function. Instead, the function \texttt{xdr\_free(xdr\_nis\_object, dest)} must be used.

\texttt{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**

\texttt{NIS\_PATH}

This variable overrides the default NIS+ directory search path used by \texttt{nis\_getnames()}. It contains an ordered list of directories separated by ‘:’ (colon) characters. The ‘$’ (dollar sign) character is treated specially. Directory names that end in ‘$’ have the default domain appended to them, and a ‘$’ by itself is replaced by the list of directories between the default domain and the global root that are at least two levels deep. The default NIS+ directory search path is ‘$’.

**ATTRIBUTES**

See \texttt{attributes(5)} for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

**SEE ALSO**

\texttt{nis\_names(3N), nis\_objects(3N), nis\_tables(3N), attributes(5)}

**NOTES**

\texttt{nis\_leaf\_of()}, \texttt{nis\_name\_of()} and \texttt{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
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);

DESCRIPTION
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.

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:

[ colname=value, ... ],tablename

The list function, nis_list(), takes an indexed name as the value for the name parameter. Here, the tablename should be a fully qualified NIS+ name unless the EXPAND_NAME flag (described below) is set. The second parameter, flags, defines how the function will respond to various conditions. The value for this parameter is created by logically ORing together one or more flags from the following list.

FOLLOW_LINKS
If the table specified in name resolves to be a LINK type object (see nis_objects(3N)), this flag specifies that the client library follow that link and do the search at that object. If this flag is not set and the name resolves to a link, the error NIS_NOTSEARCHABLE will be returned.

FOLLOW_PATH This flag specifies that if the entry is not found within this table, the list operation should follow the path specified in the table object. When used in conjunction with the ALL_RESULTS flag below, it specifies that
the path should be followed regardless of the result of the search. When used in conjunction with the FOLLOW_LINKS flag above, named tables in the path that resolve to links will be followed until the table they point to is located. If a table in the path is not reachable because no server that serves it is available, the result of the operation will be either a “soft” success or a “soft” failure to indicate that not all tables in the path could be searched. If a name in the path names is either an invalid or non-existent object then it is silently ignored.

**HARD_LOOKUP**

This flag specifies that the operation should continue trying to contact a server of the named table until a definitive result is returned (such as NIS_NOTFOUND).

**ALL_RESULTS**

This flag can only be used in conjunction with FOLLOW_PATH and a callback function. When specified, it forces all of the tables in the path to be searched. If name does not specify a search criteria (imply that all entries are to be returned), then this flag will cause all of the entries in all of the tables in the path to be returned.

**NO_CACHE**

This flag specifies that the client library should bypass any client object caches and get its information directly from either the master server or a replica server for the named table.

**MASTERONLY**

This flag is even stronger than NO_CACHE in that it specifies that the client library should only get its information from the master server for a particular table. This guarantees that the information will be up to date. However, there may be severe performance penalties associated with contacting the master server directly on large networks. When used in conjunction with the HARD_LOOKUP flag, this will block the list operation until the master server is up and available.

**EXPAND_NAME**

When specified, the client library will attempt to expand a partially qualified name by calling nis_getnames() (see nis_local_names(3N)) which uses the environment variable NIS_PATH.

**RETURN_RESULT**

This flag is used to specify that a copy of the returning object be returned in the nis_result structure if the operation was successful.

The third parameter to nis_list(), callback, is an optional pointer to a function that will process the ENTRY type objects that are returned from the search. If this pointer is NULL, then all entries that match the search criteria are returned in the nis_result structure, otherwise this function will be called once for each entry returned. When called, this function should return 0 when additional objects are desired and 1 when it no longer wishes to see any more objects. The fourth parameter, userdata, is simply passed to callback function along with the returned entry object. The client can use this pointer to pass state information or other relevant data that the callback function might need to process the entries.
The nis_list() function is not MT-Safe with callbacks. See NOTES.

nis_add_entry() will add the NIS+ object to the NIS+ table_name. The flags parameter is used to specify the failure semantics for the add operation. The default (flags equal 0) is to fail if the entry being added already exists in the table. The ADD_OVERWRITE flag may be used to specify that existing object is to be overwritten if it exists, (a modify operation) or added if it does not exist. With the ADD_OVERWRITE flag, this function will fail with the error NIS_PERMISSION if the existing object does not allow modify privileges to the client.

If the flag RETURN_RESULT has been specified, the server will return a copy of the resulting object if the operation was successful.

nis_remove_entry() removes the identified entry from the table or a set of entries identified by table_name. If the parameter object is non-null, it is presumed to point to a cached copy of the entry. When the removal is attempted, and the object that would be removed is not the same as the cached object pointed to by object then the operation will fail with an NIS_NOTSAMEOBJ error. If an object is passed with this function, the search criteria in name is optional as it can be constructed from the values within the entry. However, if no object is present, the search criteria must be included in the name parameter. If the flags variable is null, and the search criteria does not uniquely identify an entry, the NIS_NOTUNIQUE error is returned and the operation is aborted. If the flag parameter REM_MULTIPLE is passed, and if remove permission is allowed for each of these objects, then all objects that match the search criteria will be removed. Note that a null search criteria and the REM_MULTIPLE flag will remove all entries in a table.

nis_modify_entry() modifies an object identified by name. The parameter object should point to an entry with the EN_MODIFIED flag set in each column that contains new information.

The owner, group, and access rights of an entry are modified by placing the modified information into the respective fields of the parameter, object: zo_owner, zo_group, and zo_access.

These columns will replace their counterparts in the entry that is stored in the table. The entry passed must have the same number of columns, same type, and valid data in the modified columns for this operation to succeed.

If the flags parameter contains the flag MOD_SAMEOBJ then the object pointed to by object is assumed to be a cached copy of the original object. If the OID of the object passed is different than the OID of the object the server fetches, then the operation fails with the NIS_NOTSAMEOBJ error. This can be used to implement a simple read-modify-write protocol which will fail if the object is modified before the client can write the object back.

If the flag RETURN_RESULT has been specified, the server will return a copy of the resulting object if the operation was successful.

nis_first_entry() fetches entries from a table one at a time. This mode of operation is extremely inefficient and callbacks should be used instead wherever possible. The table containing the entries of interest is identified by name. If a search criteria is present in name it is ignored. The value of cookie within the nis_result structure must be copied by the caller into local storage and passed as an argument to nis_next_entry().
nis_next_entry() retrieves the “next” entry from a table specified by table_name. The order in which entries are returned is not guaranteed. Further, should an update occur in the table between client calls to nis_next_entry() there is no guarantee that an entry that is added or modified will be seen by the client. Should an entry be removed from the table that would have been the “next” entry returned, the error NIS_CHAINBROKEN is returned instead.

RETURN VALUES
These functions return a pointer to a structure of type nis_result:

```
struct nis_result {
    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_sperrno() (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 \textit{dticks} from the value in \textit{zticks} will yield the time spent in the service code itself. Subtracting the sum of the values in \textit{zticks} and \textit{aticks} from the value in \textit{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 expired. 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 \texttt{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 \textit{object} 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 \texttt{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.
<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NIS_NAMEUNREACHABLE</td>
<td>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.</td>
</tr>
<tr>
<td>NIS_NOCALLBACK</td>
<td>The server was unable to contact the callback service on your machine. This results in no data being returned.</td>
</tr>
<tr>
<td>NIS_NOMEMORY</td>
<td>Generally a fatal result. It means that the service ran out of heap space.</td>
</tr>
<tr>
<td>NIS_NOSUCHNAME</td>
<td>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.</td>
</tr>
<tr>
<td>NIS_NOSUCHTABLE</td>
<td>The named table does not exist.</td>
</tr>
<tr>
<td>NIS_NOT_ME</td>
<td>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.</td>
</tr>
<tr>
<td>NIS_NOTFOUND</td>
<td>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.</td>
</tr>
<tr>
<td>NIS_NOTMASTER</td>
<td>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.</td>
</tr>
<tr>
<td>NIS_NOTSAMEOBJ</td>
<td>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.</td>
</tr>
<tr>
<td>NIS_NOTSEARCHABLE</td>
<td>The table name resolved to a NIS+ object that was not searchable.</td>
</tr>
<tr>
<td>NIS_PARTIAL</td>
<td>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.</td>
</tr>
<tr>
<td>NIS_RPCERROR</td>
<td>This fatal error indicates the RPC subsystem failed in some way. Generally there will be a syslog(3) message indicating why the error occurred.</td>
</tr>
</tbody>
</table>
Network Functions

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.

ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe with exceptions</td>
</tr>
</tbody>
</table>

SEE ALSO

niscat(1), niserror(1), nismatch(1), nis_cachemgr(1M), nis_clone_object(3N), nis_db(3N), nis_destroy_object(3N), nis_error(3N), nis_getnames(3N), nis_local_names(3N), nis_names(3N), nis_objects(3N), nis_server(3N), rpc_svc_calls(3N), syslog(3), attributes(5)

WARNINGS

Use the flag HARD_LOOKUP carefully since it can cause the application to block indefinitely during a network partition.

modified 30 Dec 1996  SunOS 5.6  3N-1089
NOTES

The path used when the flag `FOLLOW_PATH` is specified, is the one present in the first table searched. The path values in tables that are subsequently searched are ignored.

It is legal to call functions that would access the nameservice from within a list callback. However, calling a function that would itself use a callback, or calling `nis_list()` with a callback from within a list callback function is not currently supported.

There are currently no known methods for `nis_first_entry()` and `nis_next_entry()` to get their answers from only the master server.

The `nis_list()` function is not MT-Safe with callbacks. `nis_list()` callbacks are serialized. A call to `nis_list()` with a callback from within `nis_list()` will deadlock. `nis_list()` with a callback cannot be called from an rpc server. See `rpc_svc_calls(3N)`. Otherwise, this function is MT-Safe.
<table>
<thead>
<tr>
<th>NAME</th>
<th>nl, nonl – enable/disable newline control</th>
</tr>
</thead>
</table>
| SYNOPSIS | #include <curses.h>  

int nl(void);  
int nonl(void); |
| DESCRIPTION | The `nl()` function enables the handling of newlines. The `nl()` function converts newline into carriage return and line feed on output and converts carriage return into newline on input. The `nonl()` function disables the handling of newlines.  
The handling of newlines is initially enabled. Disabling the handling of newlines results in faster cursor motion since X/Open Curses can use the line-feed capability more efficiently. |
| RETURN VALUES | On success, these functions return **OK**. Otherwise, they return **ERR**. |
| ERRORS | None. |
NAME  nlist – get entries from symbol table

SYNOPSIS  
```
/usr/ucb/cc [ flag  ... ] file  ...
#include <nlist.h>
int nlist(filename, nl)
char *filename;
struct nlist *nl;
```

DESCRIPTION  nlist() examines the symbol table from the executable image whose name is pointed to by filename, and selectively extracts a list of values and puts them in the array of nlist structures pointed to by nl. The name list pointed to by nl 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 nl, 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 nl is set to zero.

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 nl to zero.

SEE ALSO  nlist(3E), a.out(4)

NOTES  Use of these interfaces should be restricted to only applications written on BSD platforms. Use of these interfaces with any of the system libraries or in multi-thread applications is unsupported.

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)).
NAME

nlist – get entries from name list

SYNOPSIS

cc [ flag ... ] file ... -lelf [ library ... ]
#include <nlist.h>
int nlist(const char *filename, struct nlist *nl);

DESCRIPTION

nlist() examines the name list in the executable file whose name is pointed to by filename, and selectively extracts a list of values and puts them in the array of nlist() structures pointed to by nl. The name list nl 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).

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.

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.

RETURN VALUES

All value entries are set to 0 if the file cannot be read or if it does not contain a valid name list.

nlist() returns 0 on success, -1 on error.

ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO

cc(1B), elf(3E), kvm_nlist(3K), kvm_open(3K), a.out(4), attributes(5), ksym(7D), mem(7D)
NAME nl_langinfo – language information

SYNOPSIS #include <langinfo.h>
    char *nl_langinfo(nl_item item);

DESCRIPTION nl_langinfo() returns a pointer to a null-terminated string containing information relevant to a particular language or cultural area defined in the program's locale. The manifest constant names and values of item are defined by <langinfo.h>. For example:

    nl_langinfo (ABDAY_1);

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.

RETURN VALUES If setlocale(3C) has not been called successfully, or if data for a supported language is either not available, or if item 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 item contains an invalid setting.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe with exceptions</td>
</tr>
<tr>
<td>CSI</td>
<td>CSI Enabled</td>
</tr>
</tbody>
</table>

SEE ALSO setlocale(3C), attributes(5), langinfo(5), nl_types(5)

WARNINGS The array pointed to by the return value should not be modified by the program. Subsequent calls to nl_langinfo() may overwrite the array.

NOTES nl_langinfo() can be used safely in a multi-thread application, as long as setlocale(3C) is not being called to change the locale.
NAME  
nlsgetcall – get client’s data passed via the listener

SYNOPSIS  
#include <sys/tiuser.h>
struct t_call *nlsgetcall(int fildes);

DESCRIPTION  
nlsgetcall() allows server processes started by the listener process to access the client’s
t_call structure, that is, the sndcall argument of t_connect(3N).
The t_call structure returned by nlsgetcall() can be released using t_free(3N).
nlsgetcall() returns the address of an allocated t_call structure or NULL if a t_call struc-
ture cannot be allocated. If the t_alloc() succeeds, undefined environment variables are
indicated by a negative len field in the appropriate netbuf structure. A len field of zero in
the netbuf structure is valid and means that the original buffer in the listener’s t_call
structure was NULL.

WARNING  
The len 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.
The listener process limits the amount of user data (udata) and options data (opt) to 128
bytes each. Address data addr is limited to 64 bytes. If the original data was longer, no
indication of overflow is given.

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 (len) in the appropriate netbuf structure.

FILES  
/usr/lib/libnsl_s.a
/usr/lib/libslan.a
/usr/lib/libnls.a

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO  
nlsadmin(1M), getenv(3C), t_alloc(3N), t_connect(3N), t_error(3N), t_free(3N),
t_sync(3N), attributes(5)

NOTES  
Server processes must call t_sync(3N) before calling this routine.
This interface is unsafe in multithreaded applications. Unsafe interfaces should be called
only from the main thread.

modified 30 Dec 1996  SunOS 5.6  3N-1095
NAME  nlsprovider – get name of transport provider

SYNOPSIS  char *nlsprovider(void);

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 pro-
cess.

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

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO  nlsadmin(1M), attributes(5)

NOTES  This interface is unsafe in multithreaded applications. Unsafe interfaces should be called
only from the main thread.
NAME
nlsrequest – format and send listener service request message

SYNOPSIS
#include <listen.h>

int nlsrequest(int fildes, char *service_code);
extern int _nlslog, t_errno;
extern char *nlsrmsg;

DESCRIPTION
Given a virtual circuit to a listener process (fildes) and a service code of a server process, nlsrequest() formats and sends a service request message to the remote listener process requesting that it start the given service. nlsrequest() waits for the remote listener process to return a service request response message, which is made available to the caller in the static, null terminated data buffer pointed to by _nlsrmsg. The service request response message includes a success or failure code and a text message. The entire message is printable.

RETURN VALUES
The success or failure code is the integer return code from nlsrequest(). Zero indicates success, other negative values indicate nlsrequest() failures as follows:
−1 Error encountered by nlsrequest(), see t_errno.
Positive values are error return codes from the listener process. Mnemonics for these codes are defined in <listen.h>.
2 Request message not interpretable.
3 Request service code unknown.
4 Service code known, but currently disabled.
If non-null, _nlsrmsg contains a pointer to a static, null terminated character buffer containing the service request response message. Note that both _nlsrmsg and the data buffer are overwritten by each call to nlsrequest().
If _nlslog is non-zero, nlsrequest() prints error messages on stderr. Initially, _nlslog is zero.

FILES
/usr/lib/libnls.a
/usr/lib/libslan.a
/usr/lib/libns1_s.a

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
nlsadmin(1M), t_error(3N), attributes(5)
WARNINGS  

\texttt{nlsrequest()} cannot always be certain that the remote server process has been successfully started. In this case, \texttt{nlsrequest()} returns with no indication of an error and the caller will receive notification of a disconnect event via a \texttt{T\_LOOK} error before or during the first \texttt{t\_snd()} or \texttt{t\_rcv()} call.

NOTES  

These interfaces are unsafe in multithreaded applications. Unsafe interfaces should be called only from the main thread.
NAME
nodelay – set blocking or non-blocking read

SYNOPSIS
#include <curses.h>
int nodelay(WINDOW *win, bool bf);

ARGUMENTS
win    Is a pointer to the window in which to enable non-blocking.
bf    Is a Boolean expression.

DESCRIPTION
If enabled, (bf is TRUE), the nodelay() function causes getch(3XC) to return ERR if no input is ready. When disabled, getch() blocks until a key is pressed.

RETURN VALUES
On success, the nodelay() function returns OK. Otherwise, it returns ERR.

ERRORS
None.

SEE ALSO
getch(3XC), halfdelay(3XC), notimeout(3XC)
noqiflush, qiflush – control flush of input and output on interrupt

#include <curses.h>
void noqiflush(void);
void qiflush(void);

The qiflush() function enables the flushing of input and output queues when an interrupt, quit, or suspend character is sent to the terminal. The noqiflush() function disables this flushing.

These functions do not return a value.

None

flushinp(3XC), intrflush(3XC)
NAME
NOTE, _NOTE – annotate source code with info for tools

SYNOPSIS
#include <note.h>
NOTE(NoteInfo)

or
#include <sys/note.h>
NOTE(NoteInfo)

DESCRIPTION
These macros are used to embed information for tools in program source. A use of one
of these macros is called an “annotation”. A tool may define a set of such annotations
which can then be used to provide the tool with information that would otherwise be
unavailable from the source code.

Annotations should, in general, provide documentation useful to the human reader. If
information is of no use to a human trying to understand the code but is necessary for
proper operation of a tool, use another mechanism for conveying that information to the
tool (one which does not involve adding to the source code), so as not to detract from the
readability of the source. The following is an example of an annotation which provides
information of use to a tool and to the human reader (in this case, which data are pro-
tected by a particular lock, an annotation defined by the static lock analysis tool
lock_lint).

NOTE(MUTEX_PROTECTS_DATA(foo_lock, foo_list Foo))

Such annotations do not represent executable code; they are neither statements nor
declarations. They should not be followed by a semicolon. If a compiler or tool that
analyzes C source does not understand this annotation scheme, then the tool will ignore
the annotations. (For such tools, NOTE(x) expands to nothing.)

Annotations may only be placed at particular places in the source. These places are
where the following C constructs would be allowed:

• a top-level declaration (that is, a declaration not within a function or other con-
  struct)
• a declaration or statement within a block (including the block which defines a
  function)
• a member of a struct or union.

Annotations are not allowed in any other place. For example, the following are illegal:

x = y + NOTE(...) z ;
typedef NOTE(...) unsigned int uint ;

While NOTE and _NOTE may be used in the places described above, a particular type of
annotation may only be allowed in a subset of those places. For example, a particular
annotation may not be allowed inside a struct or union definition.

NOTE vs _NOTE
Ordinarily, NOTE should be used rather than _NOTE, since use of _NOTE technically
makes a program non-portable. However, it may be inconvenient to use NOTE for this
purpose in existing code if NOTE is already heavily used for another purpose. In this

modified 31 Dec 1996
SunOS 5.6
3X-1101
NOTE (3X) Miscellaneous Library Functions

case one should use a different macro and write a header file similar to
/usr/include/note.h which maps that macro to _NOTE in the same manner. For example,
the following makes FOO such a macro:

```
#ifndef _FOO_H
#define _FOO_H
#define FOO _NOTE
#include <sys/note.h>
#endif
```

Public header files which span projects should use _NOTE rather than NOTE, since
NOTE may already be used by a program which needs to include such a header file.

**NoteInfo Argument**

The actual NoteInfo used in an annotation should be specified by a tool that deals with
program source (see the documentation for the tool to determine which annotations, if
any, it understands).

NoteInfo must have one of the following forms:

```
NoteName
NoteName(Args)
```

where NoteName is simply an identifier which indicates the type of annotation, and Args
is something defined by the tool that specifies the particular NoteName. The general restric-
tions on Args are that it be compatible with an ANSI C tokenizer and that unquoted
parentheses be balanced (so that the end of the annotation can be determined without
intimate knowledge of any particular annotation).

**ATTRIBUTES**

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

**SEE ALSO** note(4), attributes(5)
### NAME
notimeout, timeout, wtimeout – set timed blocking or non-blocking read

### SYNOPSIS
```c
#include <curses.h>

int notimeout(WINDOW *win, bool bf);
void timeout(int delay);
void wtimeout(WINDOW win, int delay);
```

### ARGUMENTS
- **win**: Is a pointer to the window in which to set the timed blocking.
- **bf**: Is a Boolean expression.
- **delay**: Is the number of milliseconds to block or wait for input.

### DESCRIPTION
If `bool` is **TRUE**, the `notimeout()` function disables a timer used by `getch(3XC)` when handling multibyte function key sequences.

When `bool` is **FALSE** and keypad handling is enabled, a timer is set by `getch()` to handle bytes received that could be the beginning of a function key (for example, ESC). If the remainder of the sequence is not received before the time expires, the first byte is returned; otherwise, the value of the function key is returned. Subsequent calls to the `getch()` function will return the other bytes received for the incomplete key sequence.

The `timeout()` and `wtimeout()` functions set the length of time `getch()` waits for input for windows `stdscr` and `win`, respectively. These functions are similar to `nodelay(3XC)` except the time to block or wait for input can be specified.

A negative `delay` causes the program to wait indefinitely for input; a `delay` of **0** returns `ERR` if no input is ready; and a positive `delay` blocks until input arrives or the time specified expires, (in which case, `ERR` is returned).

### RETURN VALUES
On success, the `notimeout()` function returns **OK**. Otherwise, it returns `ERR`.

The `timeout()` and `wtimeout()` functions do not return a value.

### ERRORS
None.

### SEE ALSO
`getch(3XC)`, `halfdelay(3XC)`, `nodelay(3XC)`
NAME    offsetof – offset of structure member

SYNOPSIS #include <stddef.h>
size_t offsetof(type, member-designator);

DESCRIPTION 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 member-designator), from the beginning of its structure (designated by type).

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO attributes(5)
NAME  opendir – open directory

SYNOPSIS  
```c
#include <sys/types.h>
#include <dirent.h>

DIR *opendir(const char *dirname);
```

DESCRIPTION  The opendir() function opens a directory stream corresponding to the directory named by the dirname argument. The directory stream is positioned at the first entry. If the type DIR, is implemented using a file descriptor, applications will only be able to open up to a total of [OPEN_MAX] files and directories. A successful call to any of the exec() functions will close any directory streams that are open in the calling process.

RETURN VALUES  Upon successful completion, opendir() returns a pointer to an object of type DIR. Otherwise, a null pointer is returned and errno is set to indicate the error.

ERRORS  The opendir() function will fail if:

- EACCES  Search permission is denied for the component of the path prefix of dirname or read permission is denied for dirname.
- ELOOP  Too many symbolic links were encountered in resolving path.
- ENAMETOOLONG  The length of the dirname argument exceeds [PATH_MAX], or a pathname component is longer than [NAME_MAX] while (_POSIX_NO_TRUNC) is in effect.
- ENOENT  A component of dirname does not name an existing directory or dirname is an empty string.
- ENOTDIR  A component of dirname is not a directory.

The opendir() function may fail if:

- EMFILE  [OPEN_MAX] file descriptors are currently open in the calling process.
- ENAMETOOLONG  Pathname resolution of a symbolic link produced an intermediate result whose length exceeds [PATH_MAX].
- ENFILE  Too many files are currently open in the system.

USAGE  The opendir() function should be used in conjunction with readdir(3C), closedir(3C) and rewinddir(3C) to examine the contents of the directory (see the EXAMPLES section in readdir(3C)). This method is recommended for portability.
ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO `lstat(2), symlink(2), closedir(3C), readdir(3C), rewinddir(3C), attributes(5)`.
NAME     overlay, overwrite – overlap or overwrite windows

SYNOPSIS  
#include <curses.h>

int overlay(WINDOW *const srcwin, WINDOW *dstwin);
int overwrite(WINDOW *const srcwin, WINDOW *dstwin);

ARGUMENTS  srcwin  Is a pointer to the source window to be copied.
dstwin   Is a pointer to the destination window to be overlayed or overwritten.

DESCRIPTION The overwrite() and overlay() functions copy the overlapping portion of srcwin to
dstwin. srcwin and dstwin do not have to be the same size.

The overwrite() function copies the characters from the overlapping portion to dstwin;
thus, destroying the previous contents of the window. The overlay() function only copies
non-blank characters, leaving blank characters intact. Thus, if the background character
of the original window was set to something other than a blank, this original background
could be preserved.

The following example shows how to use overwrite() to implement a pop-up dialog box.

#include <curses.h>

.getOrElse the .h

int popup(work, save, nrows, ncols, row, col)
WINDOW **work, **save;
int nrows, ncols, row, col;
{
  int mr, mc;

getmaxyx(curscr, mr, mc);
/* Windows are limited to the size of curscr. */
if (mr < nrows)
  nrows = mr;
if (mc < ncols)
  ncols = mc;
/* Center dimensions. */
if (row == -1)
  row = (mr-nrows)/2;
if (col == -1)
    col = (mc-ncols)/2;

/* The window must fit entirely in curscr. */
if (mr < row+nrows)
    row = 0;
if (mc < col+ncols)
    col = 0;

*work = newwin(nrows, ncols, row, col);
if (*work == NULL)
    return (-1);
if ((*save = dupwin(*work)) == NULL) {
    delwin(*work);
    return (-1);
}

overwrite(curscr, *save);

return (0);

/*
 * Restore the region covered by a pop-up window.
 * Delete the working window and the saved window.
 * This function is the complement to popup(). Return
 * 0 for success or -1 for an error.
 */
int popdown(work, save)
WINDOW *work, *save;
{
    (void) wnoutrefresh(save);
    (void) delwin(save);
    (void) delwin(work);
    return (0);
}

/*
 * Compute the size of a dialog box that would fit around
 * the string.
 */
void
dialsize(str, nrows, ncols)
char *str;
int *nrows, *ncols;
{
    int rows, cols, col;

    for (rows = 1, cols = col = 0; *str != '\0'; ++str) {
        if (*str == '\n') {
            if (cols < col)
                cols = col;
            col = 0;
            ++rows;
        } else {
            ++col;
        }
    }
    if (cols < col)
        cols = col;
    *nrows = rows;
    *ncols = cols;
}

/* Write a string into a dialog box. */
void
dialfill(w, s)
    WINDOW *w;
    char *s;
{
    int row;
    (void) wmove(w, 1, 1);
    for (row = 1; *s != '\0'; ++s) {
        (void) waddch(w, *((unsigned char *) s));
        if (*s == '\n')
            wmove(w, ++row, 1);
    }
    box(w, 0, 0);
}

void
dialog(str)
    char *str;
{
    WINDOW *work, *save;
    int nrows, ncols, row, col;
Figure out size of window.

dialsize(str, &nrows, &ncols);

Create a centered working window with extra room for a border.

(void) popup(&work, &save, nrows+2, ncols+2, -1, -1);

Write text into the working window.

dialfill(work, str);

Pause. Remember that wgetch() will do a wrefresh() for us.

(void) wgetch(work);

Restore curscr and free windows.

( void) popdown(work, save);

Redraw curscr to remove window from physical screen.

(void) doupdate();

RETURN VALUES
On success, these functions return OK. Otherwise, they return ERR.

ERRORS
None.

SEE ALSO
copywin(3XC)
NAME
p2open, p2close – open, close pipes to and from a command

SYNOPSIS
cc [ flag ... ] file ... -lgen [ library ... ]
#include <libgen.h>
int p2open(const char *cmd, FILE *fp[2]);
int p2close(FILE *fp[2]);

DESCRIPTION
p2open() forks and executes a shell running the command line pointed to by cmd. 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.
The function returns 0 if successful; otherwise, it returns –1.
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.

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
#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);
}
ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO    fclose(3S), popen(3S), setbuf(3S), attributes(5)

NOTES
Buffered writes on fp[0] can make it appear that the command is not listening. Judiciously placed fflush() calls or unbuffering fp[0] can be a big help; see fclose(3S).
Many commands use buffered output when connected to a pipe. That, too, can make it appear as if things are not working.
Usage is not the same as for popen(), although it is closely related.
**NAME**

pam – PAM (Pluggable Authentication Module)

**SYNOPSIS**

```c
#include <security/pam_appl.h>
cc [ flag ... ] file ... -lpam [ library ... ]
```

**DESCRIPTION**

The PAM framework, **libpam**, consists of an interface library and multiple authentication service modules. The PAM interface library is the layer implementing the Application Programming Interface (API). The authentication service modules are a set of dynamically loadable objects invoked by the PAM API to provide a particular type of user authentication. PAM gives system administrators the flexibility of choosing any authentication service available on the system to perform authentication. This framework also allows new authentication service modules to be plugged in and made available without modifying the applications.

**Interface Overview**

The PAM library interface consists of six categories of functions, the names for which all start with the prefix **pam_**.

The first category contains functions for establishing and terminating an authentication activity, which are **pam_start(3)** and **pam_end(3)**. The functions **pam_set_data(3)** and **pam_get_data(3)** maintain module specific data. The functions **pam_set_item(3)** and **pam_get_item(3)** maintain state information. **pam_strerror(3)** is the function that returns error status information.

The second category contains the functions that authenticate an individual user and set the credentials of the user, **pam_authenticate(3)** and **pam_setcred(3)**.

The third category of PAM interfaces is account management. The function **pam_acct_mgmt(3)** checks for password aging and access-hour restrictions.

Category four contains the functions that perform session management after access to the system has been granted. See **pam_open_session(3)** and **pam_close_session(3)**.

The fifth category consists of the function that changes authentication tokens, **pam_chauthtok(3)**. An authentication token is the object used to verify the identity of the user. In UNIX, an authentication token is a user’s password.

The sixth category of functions can be used to set values for PAM environment variables. See **pam_putenv(3)**, **pam_getenv(3)**, and **pam_getenvlist(3)**.

The **pam_***( ) interfaces are implemented through the library **libpam**. For each of the categories listed above, excluding categories one and six, dynamically loadable shared modules exist that provides the appropriate service layer functionality upon demand. The functional entry points in the service layer start with the **pam_sm_** prefix. The only difference between the **pam_sm_***( ) interfaces and their corresponding **pam_** interfaces is that all the **pam_sm_***( ) interfaces require extra parameters to pass service-specific options to the shared modules. Refer to **pam_sm(3)** for an overview of the PAM service module APIs.

**modified 26 Mar 1997**

SunOS 5.6

3-1113
Stateful Interface

A sequence of calls sharing a common set of state information is referred to as an authentication transaction. An authentication transaction begins with a call to `pam_start()`. `pam_start()` allocates space, performs various initialization activities, and assigns a PAM authentication handle to be used for subsequent calls to the library.

After initiating an authentication transaction, applications can invoke `pam_authenticate()` to authenticate a particular user, and `pam_acct_mgmt()` to perform system entry management. For example, the application may want to determine if the user’s password has expired.

If the user has been successfully authenticated, the application calls `pam_setcred()` to set any user credentials associated with the authentication service. Within one authentication transaction (between `pam_start()` and `pam_end()`), all calls to the PAM interface should be made with the same authentication handle returned by `pam_start()`. This is necessary because certain service modules may store module-specific data in a handle that is intended for use by other modules. For example, during the call to `pam_authenticate()`, service modules may store data in the handle that is intended for use by `pam_setcred()`.

To perform session management, applications call `pam_open_session()`. Specifically, the system may want to store the total time for the session. The function `pam_close_session()` closes the current session.

When necessary, applications can call `pam_get_item()` and `pam_set_item()` to access and to update specific authentication information. Such information may include the current username.

To terminate an authentication transaction, the application simply calls `pam_end()`, which frees previously allocated space used to store authentication information.

Application–Authentication Service Interactive Interface

The authentication service in PAM does not communicate directly with the user; instead it relies on the application to perform all such interactions. The application passes a pointer to the function, `conv()`, along with any associated application data pointers, through a `pam_conv` structure to the authentication service when it initiates an authentication transaction, via a call to `pam_start()`. The service will then use the function, `conv()`, to prompt the user for data, output error messages, and display text information. Refer to `pam_start(3)` for more information.

Stacking Multiple Schemes

The PAM architecture enables authentication by multiple authentication services through stacking. System entry applications, such as `login(1)`, stack multiple service modules to authenticate users with multiple authentication services. The order in which authentication service modules are stacked is specified in the configuration file, `pam.conf(4)`. A system administrator determines this ordering, and also determines whether the same password can be used for all authentication services.

Administrative Interface

The authentication library, `/usr/lib/libpam.so.1`, implements the framework interface. Various authentication services are implemented by their own loadable modules whose paths are specified through the `pam.conf(4)` file.
RETURN VALUES

The PAM functions may return one of the following generic values, or one of the values defined in the specific man pages:

- **PAM_SUCCESS**: The function returned successfully.
- **PAM_OPEN_ERR**: `dlopen()` failed when dynamically loading a service module.
- **PAM_SYMBOL_ERR**: Symbol not found.
- **PAM_SERVICE_ERR**: Error in service module.
- **PAM_SYSTEM_ERR**: System error.
- **PAM_BUF_ERR**: Memory buffer error.
- **PAM_CONV_ERR**: Conversation failure.
- **PAM_PERM_DENIED**: Permission denied.

ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT Level</td>
<td>MT-Safe with exceptions</td>
</tr>
</tbody>
</table>

SEE ALSO

`login(1)`, `pam_authenticate(3)`, `pam_chauthtok(3)`, `pam_open_session(3)`, `pam_set_item(3)`, `pam_setcred(3)`, `pam_sm(3)`, `pam_start(3)`, `pam_strerror(3)`

WARNING

Please note that all the PAM APIs and their data structures are subject to change without notice.

NOTES

The interfaces in `libpam()` are MT-safe only if each thread within the multi-threaded application uses its own PAM handle.
NAME
pam_acct_mgmt – perform PAM account validation procedures

SYNOPSIS
cc [ flag ...] file ... -lpam [ library ...]
#include <security/pam_appl.h>
int pam_acct_mgmt(pam_handle_t *pamh, int flags);

DESCRIPTION
The function pam_acct_mgmt() is called to determine if the current user's account is
valid. It checks for password and account expiration, and verifies access hour restrictions. This function is typically called after the user has been authenticated with
pam_authenticate(3).

The pamh argument is an authentication handle obtained by a prior call to pam_start().
The following flags may be set in the flags field:

PAM_SILENT
The account management service should not generate any
messages.

PAM_DISALLOW_NULL_AUTHTOK
The account management service should return
PAM_NEW_AUTHTOK_REQD if the user has a null authentication token.

RETURN VALUES
Upon successful completion, PAM_SUCCESS is returned. In addition to the error return
values described in pam(3), the following values may be returned:

PAM_USER_UNKNOWN
User not known to underlying account management module.

PAM_AUTH_ERR
Authentication failure.

PAM_NEW_AUTHTOK_REQD
New authentication token required. This is normally returned
if the machine security policies require that the password
should be changed because the password is NULL or has aged.

PAM_ACCT_EXPIRED
User account has expired.

ATTRIBUTES
See attributes(5) for description of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT Level</td>
<td>MT-Safe with exceptions</td>
</tr>
</tbody>
</table>

SEE ALSO
pam(3), pam_authenticate(3), pam_start(3), libpam(4), attributes(5)

NOTES
The interfaces in libpam() are MT-safe only if each thread within the multi-threaded
application uses its own PAM handle.
NAME  pam_authenticate – perform authentication within the PAM framework

SYNOPSIS  cc [ flag ...]file ... -lpam [ library ...]
            #include <security/pam_appl.h>
            int pam_authenticate(pam_handle_t *pamh, int flags);

DESCRIPTION  pam_authenticate() is called to authenticate the current user. The user is usually required to enter a password or similar authentication token depending upon the authentication service configured within the system. The user in question should have been specified by a prior call to pam_start() or pam_set_item().

The following flags may be set in the flags field:

PAM_SILENT  Authentication service should not generate any messages.

PAM_DISALLOW_NULL_AUTHTOK  The authentication service should return PAM_AUTH_ERROR if the user has a null authentication token.

RETURN VALUES  Upon successful completion, PAM_SUCCESS is returned. In addition to the error return values described in pam(3), the following values may be returned:

PAM_AUTH_ERR  Authentication failure.

PAM_CRED_INSUFFICIENT  Cannot access authentication data due to insufficient credentials.

PAM_AUTHINFO_UNAVAIL  Underlying authentication service cannot retrieve authentication information.

PAM_USER_UNKNOWN  User not known to the underlying authentication module.

PAM_MAXTRIES  An authentication service has maintained a retry count which has been reached. No further retries should be attempted.

ATTRIBUTES  See attributes(5) for description of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT Level</td>
<td>MT-Safe with exceptions</td>
</tr>
</tbody>
</table>

SEE ALSO  pam(3), pam_open_session(3), pam_set_item(3), pam_setcred(3), pam_start(3), libpam(4), attributes(5)

NOTES  In the case of authentication failures due to an incorrect username or password, it is the responsibility of the application to retry pam_authenticate() and to maintain the retry count. An authentication service module may implement an internal retry count and return an error PAM_MAXTRIES if the module does not want the application to retry.

modified 11 Apr 1997  SunOS 5.6  3-1117
If the PAM framework cannot load the authentication module, then it will return `PAM_ABORT`. This indicates a serious failure, and the application should not attempt to retry the authentication.

For security reasons, the location of authentication failures is hidden from the user. Thus, if several authentication services are stacked and a single service fails, `pam_authenticate()` requires that the user re-authenticate each of the services.

A null authentication token in the authentication database will result in successful authentication unless `PAM_DISALLOW_NULL_AUTHTOK` was specified. In such cases, there will be no prompt to the user to enter an authentication token.

The interfaces in `libpam()` are MT-Safe only if each thread within the multi-threaded application uses its own PAM handle.
NAME
pam_chauthtok – perform password related functions within the PAM framework

SYNOPSIS
cc [ flag ...] file ... -lpam [ library ...]
#include <security/pam_appl.h>
int pam_chauthtok(pam_handle_t *pamh, const int flags);

DESCRIPTION
pam_chauthtok() is called to change the authentication token associated with a particular user referenced by the authentication handle, pamh.
The following flag may be passed in to pam_chauthtok():
PAM_SILENT The password service should not generate any messages.
PAM_CHANGE_EXPIRED_AUTHTOK
The password service should only update those passwords that have aged. If this flag is not passed, all password services should update their passwords.

Upon successful completion of the call, the authentication token of the user will be changed in accordance with the password service configured in the system through pam.conf(4).

RETURN VALUES
Upon successful completion, PAM_SUCCESS is returned. In addition to the error return values described in pam(3), the following values may be returned:
PAM_PERM_DENIED No permission.
PAM_AUTHTOK_ERR Authentication token manipulation error.
PAM_AUTHTOK_RECOVERY_ERR Authentication information cannot be recovered.
PAM_AUTHTOK_LOCK_BUSY Authentication token lock busy.
PAM_AUTHTOK_DISABLE_AGING Authentication token aging disabled.
PAM_USER_UNKNOWN User unknown to password service.
PAM_TRY_AGAIN Preliminary check by password service failed.

ATTRIBUTES
See attributes(5) for description of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT Level</td>
<td>MT-Safe with exceptions</td>
</tr>
</tbody>
</table>

SEE ALSO
login(1), passwd(1), pam(3), pam_authenticate(3), pam_start(3), attributes

NOTES
The flag PAM_CHANGE_EXPIRED_AUTHTOK is typically used by a login application which has determined that the user’s password has aged or expired. Before allowing the user to login, the login application may invoke pam_chauthtok() with this flag to allow
modified 28 Oct 1996 SunOS 5.6 3-1119
the user to update the password. Typically applications such as passwd(1) should not use this flag.

`pam_chauthtok()` performs a preliminary check before attempting to update passwords. This check is performed for each password module in the stack as listed in `pam.conf(4)`. The check may include pinging remote name services to determine if they are available. If `pam_chauthtok()` returns `PAM_TRY_AGAIN`, then the check has failed, and passwords are not updated.

The interfaces in `libpam()` are MT-Safe only if each thread within the multi-threaded application uses its own PAM handle.
NAME
pam_getenv – returns the value for a PAM environment name

SYNOPSIS
cc [ flag ...] file ... -lpam [ library ...]
#include <security/pam_appl.h>
char *pam_getenv(pam_handle_t *pamh, const char *name);

DESCRIPTION
pam_getenv() searches the PAM handle, pamh, for a value associated with name. If a
value is present, pam_getenv() makes a copy of the value and returns a pointer to the
copy back to the calling application. If no such entry exists, pam_getenv() returns NULL.
It is the responsibility of the calling application to free the memory returned by
pam_getenv().

RETURN VALUES
If successful, pam_getenv() returns a copy of the value associated with name in the PAM
handle; otherwise, it returns a NULL pointer.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT Level</td>
<td>MT-Safe with exceptions</td>
</tr>
</tbody>
</table>

SEE ALSO
pam(3), pam_getenvlist(3), pam_putenv(3), libpam(4), attributes(5)

NOTES
The interfaces in libpam() are MT-Safe only if each thread within the multi-threaded application uses its own PAM handle.
pam_getenvlist (3)  C Library Functions

NAME
pam_getenvlist – returns a list of all the PAM environment variables

SYNOPSIS
cc [ flag ... ] file ... -lpam [ library ... ]
#include <security/pam_appl.h>
char **pam_getenvlist(pam_handle_t *pamh);

DESCRIPTION
pam_getenvlist() returns a list of all the PAM environment variables stored in the PAM
handle, pamh. The list is returned as a null-terminated array of pointers to strings. Each
string contains a single PAM environment variable of the form name=value. The list
returned is a duplicate copy of all the environment variables stored in pamh. It is the
responsibility of the calling application to free the memory returned by
pam_getenvlist().

RETURN VALUES
If successful pam_getenvlist() returns, in a null-terminated array, a copy of all the PAM
environment variables stored in pamh. Upon error, pam_getenvlist() returns a null
pointer.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT Level</td>
<td>MT-Safe with exceptions</td>
</tr>
</tbody>
</table>

SEE ALSO
pam(3), pam_getenv(3), pam_putenv(3), libpam(4), attributes(5)

NOTES
The interfaces in libpam() are MT-Safe only if each thread within the multi-threaded
application uses its own PAM handle.
**NAME**

pam_get_user – PAM routine to retrieve user name

**SYNOPSIS**

```
c [ flag ... ] file ... -lpam [ library ... ]
#include <security/pam_appl.h>

int pam_get_user(pam_handle_t *pamh, char **user, const char *prompt);
```

**DESCRIPTION**

`pam_get_user()` is used by PAM service modules to retrieve the current user name from the PAM handle. If the user name has not been set via `pam_start()` or `pam_set_item()`, then the PAM conversation function will be used to prompt the user for the user name with the string "prompt". If `prompt` is NULL, then `pam_get_item()` is called and the value of `PAM_USER_PROMPT` is used for prompting. If the value of `PAM_USER_PROMPT` is NULL, the following default prompt is used:

```
Please enter user name:
```

After the user name is gathered by the conversation function, `pam_set_item()` is called to set the value of `PAM_USER`. By convention, applications that need to prompt for a user name should call `pam_set_item()` and set the value of `PAM_USER_PROMPT` before calling `pam_authenticate()`. The service module’s `pam_sm_authenticate()` function will then call `pam_get_user()` to prompt for the user name. Note that certain PAM service modules, such as a smart card module, may override the value of `PAM_USER_PROMPT` and pass in their own prompt. Applications that call `pam_authenticate()` multiple times should set the value of `PAM_USER` to NULL with `pam_set_item()` before calling `pam_authenticate()`, if they want the user to be prompted for a new user name each time. The value of `user` retrieved by `pam_get_user()` should not be modified or freed. The item will be released by `pam_end()`.

**RETURN VALUES**

Upon success `pam_get_user()` returns PAM_SUCCESS; otherwise it returns an error code. Refer to `pam(3)` for information on error related return values.

**ATTRIBUTES**

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT Level</td>
<td>MT-Safe with exceptions</td>
</tr>
</tbody>
</table>

**SEE ALSO**

`pam(3), pam_authenticate(3), pam_end(3), pam_get_item(3), pam_set_item(3), pam_sm(3), pam_sm_authenticate(3), pam_start(3), attributes(5)`

**NOTES**

The interfaces in `libpam()` are MT-Safe only if each thread within the multi-threaded application uses its own PAM handle.

modified 28 Oct 1996
NAME  pam_open_session, pam_close_session – perform PAM session creation and termination operations

SYNOPSIS  cc [ flag ... ] file ... -lpam [ library ... ]
#include <security/pam_appl.h>

int pam_open_session(pam_handle_t *pamh, int flags);
int pam_close_session(pam_handle_t *pamh, int flags);

DESCRIPTION  pam_open_session() is called after a user has been successfully authenticated. See pam_authenticate(3) and pam_acct_mgmt(3). It is used to notify the session modules that a new session has been initiated. All programs that use the pam(3) library should invoke pam_open_session() when beginning a new session. Upon termination of this activity, pam_close_session() should be invoked to inform pam(3) that the session has terminated.

The pamh argument is an authentication handle obtained by a prior call to pam_start(). The following flag may be set in the flags field for pam_open_session() and pam_close_session():

PAM_SILENT    The session service should not generate any messages.

RETURN VALUES  Upon successful completion, PAM_SUCCESS is returned. In addition to the return values defined in pam(3), the following value may be returned on error:

PAM_SESSION_ERR    Cannot make or remove an entry for the specified session.

ATTRIBUTES  See attributes(5) for description of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT Level</td>
<td>MT-Safe with exceptions</td>
</tr>
</tbody>
</table>

SEE ALSO  getutxent(3C), pam(3), pam_acct_mgmt(3), pam_authenticate(3), pam_start(3), attributes(5)

NOTES  In many instances, the pam_open_session() and pam_close_session() calls may be made by different processes. For example, in UNIX the login process opens a session, while the init process closes the session. In this case, UTMP/WTMP entries may be used to link the call to pam_close_session() with an earlier call to pam_open_session(). This is possible because UTMP/WTMP entries are uniquely identified by a combination of attributes, including the user login name and device name, which are accessible through the PAM handle, pamh. The call to pam_open_session() should precede UTMP/WTMP entry management, and the call to pam_close_session() should follow UTMP/WTMP exit management.

The interfaces in libpam() are MT-Safe only if each thread within the multi-threaded application uses its own PAM handle.
NAME  pam_putenv – change or add a value to the PAM environment

SYNOPSIS  cc [ flag ...] file ... -lpam [ library ...]
#include <security/pam_appl.h>
int pam_putenv(pam_handle_t *pamh, const char *name_value);

DESCRIPTION  The pam_putenv() routine sets the value of the PAM environment variable name equal to value either by altering an existing PAM variable or by creating a new one. The variable name_value points to a string of the form name=value. A call to pam_putenv() does not immediately change the environment. All name_value pairs are stored in the PAM handle, pamh. An application such as login(1) may make a call to pam_getenv(3) or pam_getenvlist(3) to retrieve the PAM environment variables saved in the PAM handle and set them in the environment if appropriate. login will not set PAM environment values which overwrite the values for SHELL, HOME, LOGNAME, MAIL, CDPATH, IFS, and PATH. Nor will login set PAM environment values which overwrite any value that begins with LD_. If name_value equals NAME=, then the value associated with NAME in the PAM handle will be set to an empty value. If name_value equals NAME, then the environment variable NAME will be removed from the PAM handle.

RETURN VALUES  The function pam_putenv() may return one of the following values:
PAM_SUCCESS  The function returned successfully.
PAM_OPEN_ERR  dlopen() failed when dynamically loading a service module.
PAM_SYMBOL_ERR  Symbol not found.
PAM_SERVICE_ERR  Error in service module.
PAM_SYSTEM_ERR  System error.
PAM_BUF_ERR  Memory buffer error.
PAM_CONV_ERR  Conversation failure.
PAM_PERM_DENIED  Permission denied.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT Level</td>
<td>MT-Safe with exceptions</td>
</tr>
</tbody>
</table>

SEE ALSO  dlopen(3X), pam(3), pam_getenv(3), pam_getenvlist(3), libpam(4), attributes(5)

NOTES  The interfaces in libpam() are MT-Safe only if each thread within the multi-threaded application uses its own PAM handle.

modified 26 Mar 1997  SunOS 5.6  3-1125
NAME  pam_setcred – modify/delete user credentials for an authentication service

SYNOPSIS  cc [ flag ... ] file ... -lpam [ library ... ]
#include <security/pam_appl.h>
int pam_setcred(pam_handle_t * pamh, int flags);

DESCRIPTION  pam_setcred() is used to establish, modify, or delete user credentials. pam_setcred() is
typically called after the user has been authenticated and after a session has been opened.
See pam_authenticate(3), pam_acct_mgmt(3), and pam_open_session(3).
The user is specified by a prior call to pam_start() or pam_set_item(), and is referenced
by the authentication handle, pamh. The following flags may be set in the flags field.
Note that the first four flags are mutually exclusive:
PAM_ESTABLISH_CRED  Set user credentials for an authentication service.
PAM_DELETE_CRED  Delete user credentials associated with an authentication service.
PAM_REINITIALIZE_CRED  Reinitialize user credentials.
PAM_REFRESH_CRED  Extend lifetime of user credentials.
PAM_SILENT  Authentication service should not generate any messages.
If no flag is set, PAM_ESTABLISH_CRED is used as the default.

RETURN VALUES  Upon success, pam_setcred() returns PAM_SUCCESS. In addition to the error return
values described in pam(3), the following values may be returned upon error:
PAM_CRED_UNAVAIL  Underlying authentication service can not retrieve user
credentials unavailable.
PAM_CRED_EXPIRED  User credentials expired.
PAM_USER_UNKNOWN  User unknown to underlying authentication service.
PAM_CRED_ERR  Failure setting user credentials.

ATTRIBUTES  See attributes(5) for description of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT Level</td>
<td>MT-Safe with exceptions</td>
</tr>
</tbody>
</table>

SEE ALSO  pam(3), pam_acct_mgmt(3), pam_authenticate(3), pam_open_session(3),
pam_set_item(3), pam_start(3), libpam(4), attributes(5)

NOTES  The interfaces in libpam() are MT-Safe only if each thread within the multi-threaded
application uses its own PAM handle.
NAME
pam_set_data, pam_get_data – PAM routines to maintain module specific state

SYNOPSIS
cc [ flag ...] file ... -lpam [ library ...]
#include <security/pam_appl.h>

int pam_set_data(pam_handle_t *pamh,
        const char *module_data_name, void *data, void (*cleanup)(pam_handle_t *pamh, void *data,
        int pam_end_status));

int pam_get_data(const pam_handle_t *pamh,
        const char *module_data_name,
        const void ***data);

DESCRIPTION
pam_set_data() and pam_get_data() allow PAM service modules to access and update
module specific information as needed. These functions should not be used by applica-
tions.
pam_set_data() stores module specific data within the PAM handle, pamh. The
module_data_name argument uniquely identifies the data, and the data argument
represents the actual data. module_data_name should be unique across all services (UNIX,
etc).
The cleanup function frees up any memory used by the data after it is no longer needed,
and is invoked by pam_end(). The cleanup function takes as its arguments a pointer to
the PAM handle, pamh, a pointer to the actual data, data, and a status code,
pam_end_status. The status code determines exactly what state information needs to be
purged.
If pam_set_data() is called and module data already exists from a prior call to
pam_set_data() under the same module_data_name, then the existing data is replaced by
the new data, and the existing cleanup function is replaced by the new cleanup function.
pam_get_data() retrieves module-specific data stored in the PAM handle, pamh,
identified by the unique name, module_data_name. The data argument is assigned the
address of the requested data. The data retrieved by pam_get_data() should not be
modified or freed. The data will be released by pam_end().

RETURN VALUES
In addition to the return values listed in pam(3), the following value may also be
returned:

PAM_NO_MODULE_DATA No module specific data is present.

ATTRIBUTES
See attributes(5) for description of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT Level</td>
<td>MT-Safe with exceptions</td>
</tr>
</tbody>
</table>

modified 28 Oct 1996 SunOS 5.6 3-1127
### SEE ALSO

```
pam(3), pam_end(3), libpam(4), attributes(5)
```

### NOTES

The interfaces in `libpam()` are MT-Safe only if each thread within the multi-threaded application uses its own PAM handle.
NAME

pam_set_item, pam_get_item – authentication information routines for PAM

SYNOPSIS

cc [ flag ...] file ... -lpam [ library ...]
#include <security/pam_appl.h>

int pam_set_item(pam_handle_t *pamh, int item_type, const void *item);
int pam_get_item(const pam_handle_t *pamh, int item_type, void **item);

DESCRIPTION

pam_get_item() and pam_set_item() allow applications and PAM service modules to
access and to update PAM information as needed. The information is specified by
item_type, and can be one of the following:

PAM_SERVICE The service name.
PAM_USER The user name.
PAM_AUTHTOK The user authentication token.
PAM_OLEDAUTHTOK The old user authentication token.
PAM_TTY The tty name.
PAM_RHOST The remote host name.
PAM_RUSER The remote user name.
PAM_CONV The pam_conv structure.
PAM_USER_PROMPT The default prompt used by pam_get_user().

For security reasons, the item_type PAM_AUTHTOK and PAM_OLEDAUTHTOK are avail-
able only to the module providers. The authentication module, account module, and ses-
session management module should treat PAM_AUTHTOK as the current authentication
token and ignore PAM_OLEDAUTHTOK. The password management module should treat
PAM_OLEDAUTHTOK as the current authentication token and PAM_AUTHTOK as the new
authentication token.

pam_set_item() is passed the authentication handle, pamh, returned by pam_start(), a
pointer to the object, item, and its type, item_type. If successful, pam_set_item() copies
the item to an internal storage area allocated by the authentication module and returns
PAM_SUCCESS. An item that had been previously set will be overwritten by the new
value.

pam_get_item() is passed the authentication handle, pamh, returned by pam_start(), an
item_type, and the address of the pointer, item, which is assigned the address of the
requested object. The object data is valid until modified by a subsequent call to
pam_set_item() for the same item_type, or unless it is modified by any of the underlying
service modules. If the item has not been previously set, pam_get_item() returns a null
pointer. An item retrieved by pam_get_item() should not be modified or freed. The item
will be released by pam_end().

modified 28 Oct 1996
SunOS 5.6
3-1129
Upon success `pam_get_item()` returns `PAM_SUCCESS`; otherwise it returns an error code. Refer to `pam(3)` for information on error related return values.

**ATTRIBUTES**

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT Level</td>
<td>MT-Safe with exceptions</td>
</tr>
</tbody>
</table>

**SEE ALSO**

`pam(3), pam_acct_mgmt(3), pam_authenticate(3), pam_chauthtok(3), pam_get_user(3), pam_open_session(3), pam_setcred(3), pam_start(3), attributes(5)`

**NOTES**

The interfaces in `libpam()` are MT-Safe only if each thread within the multi-threaded application uses its own PAM handle.
NAME    pam_sm – PAM Service Module APIs

SYNOPSIS  

```
#include <security/pam_appl.h>
#include <security/pam_modules.h>
cc [ flag ... ] file ... -lpam [ library ... ]
```

DESCRIPTION  

PAM gives system administrators the flexibility of choosing any authentication service available on the system to perform authentication. The framework also allows new authentication service modules to be plugged in and made available without modifying the applications.

The PAM framework, libpam, consists of an interface library and multiple authentication service modules. The PAM interface library is the layer implementing the Application Programming Interface (API). The authentication service modules are a set of dynamically loadable objects invoked by the PAM API to provide a particular type of user authentication.

This manual page gives an overview of the PAM APIs for the service modules.

Interface Overview  

The PAM service module interface consists of functions which can be grouped into four categories. The names for all the authentication library functions start with `pam_sm`. The only difference between the `pam_`(*) interfaces and their corresponding `pam_sm_`(*) interfaces is that all the `pam_sm_`(*) interfaces require extra parameters to pass service-specific options to the shared modules. They are otherwise identical.

The first category contains functions to authenticate an individual user, `pam_sm_authenticate(3)`, and to set the credentials of the user, `pam_sm_setcred(3)`. These back-end functions implement the functionality of `pam_authenticate(3)` and `pam_setcred(3)` respectively.

The second category contains the function to do account management: `pam_sm_acct_mgmt(3)`. This includes checking for password aging and access-hour restrictions. This back-end function implements the functionality of `pam_acct_mgmt(3)`.

The third category contains the functions `pam_sm_open_session(3)` and `pam_sm_close_session(3)` to perform session management after access to the system has been granted. These back-end functions implement the functionality of `pam_open_session(3)` and `pam_close_session(3)`, respectively.

The fourth category consists a function to change authentication tokens `pam_sm_chauthtok(3)`. This back-end function implements the functionality of `pam_chauthtok(3)`.

Stateful Interface  

A sequence of calls sharing a common set of state information is referred to as an authentication transaction. An authentication transaction begins with a call to `pam_start()`. `pam_start()` allocates space, performs various initialization activities, and assigns an authentication handle to be used for subsequent calls to the library. Note that the service modules do not get called or initialized when `pam_start()` is called. The modules are loaded and the symbols resolved upon first use of that function.

modified 28 Oct 1996

SunOS 5.6

3-1131
The PAM handle keeps certain information about the transaction that can be accessed through the `pam_get_item()` API. Though the modules can also use `pam_set_item()` to change any of the item information, it is recommended that nothing be changed except `PAM_AUTHTOK` and `PAM_OLEDAUTHTOK`.

If the modules want to store any module specific state information then they can use the `pam_set_data()` function to store that information with the PAM handle. The data should be stored with a name which is unique across all modules and module types. For example, `SUNW_PAM_UNIX_AUTH_userid` can be used as a name by the UNIX module to store information about the state of user’s authentication. Some modules use this technique to share data across two different module types.

Also, during the call to `pam_authenticate()`, the UNIX module may store the authentication status (success or reason for failure) in the handle, using a unique name such as `SUNW_SECURE_RPC_DATA`. This information is intended for use by `pam_setcred()`.

During the call to `pam_acct_mgmt()`, the account modules may store data in the handle to indicate which passwords have aged. This information is intended for use by `pam_chauthtok()`.

The module can also store a cleanup function associated with the data. The PAM framework calls this cleanup function, when the application calls `pam_end()` to close the transaction.

**Interaction with the User**

The PAM service modules do not communicate directly with the user; instead they rely on the application to perform all such interactions. The application passes a pointer to the function, `conv()`, along with any associated application data pointers, through the `pam_conv` structure when it initiates an authentication transaction (via a call to `pam_start()`). The service module will then use the function, `conv()`, to prompt the user for data, output error messages, and display text information. Refer to `pam_start(3)` for more information. The modules are responsible for the localization of all messages to the user.

**CONVENTIONS**

By convention, applications that need to prompt for a user name should call `pam_set_item()` and set the value of `PAM_USER_PROMPT` before calling `pam_authenticate()`. The service module’s `pam_sm_authenticate()` function will then call `pam_get_user()` to prompt for the user name. Note that certain PAM service modules (such as a smart card module) may override the value of `PAM_USER_PROMPT` and pass in their own prompt.

Though the PAM framework enforces no rules about the module’s names, location, options and such, there are certain conventions that all module providers are expected to follow.

By convention, the modules should be located in the `/usr/lib/security` directory. Additional modules may be located in `/opt/<pkg>/lib`.

By convention, the modules are named `pam_<service_name>_<module_type>.so.1`. If the given module implements more than one module type (for example, `pam_unix.so.1` module), then the module_type suffix should be dropped.
For every such module, there should be a corresponding manual page in section 5 which should describe the *module_type* it supports, the functionality of the module, along with the options it supports. The dependencies should be clearly identified to the system administrator. For example, it should be made clear whether this module is a stand-alone module or depends upon the presence of some other module. One should also specify whether this module should come before or after some other module in the stack.

By convention, the modules should support the following options:

**debug**  
Syslog debugging information at **LOG_DEBUG** level. Be careful as to not log any sensitive information such as passwords.

**nowarn**  
Turn off warning messages such as "password is about to expire."

In addition, it is recommended that the auth and the password module support the following options:

**use_first_pass**  
Instead of prompting the user for the password, use the user’s initial password (entered when the user was authenticated to the first authentication module in the stack) for authentication. If the passwords do not match, or if no password has been entered, return failure and do not prompt the user for a password. Support for this scheme allows the user to type only one password for multiple schemes.

**try_first_pass**  
Instead of prompting the user for the password, use the user’s initial password (entered when the user was authenticated to the first authentication module in the stack) for authentication. If the passwords do not match, or if no password has been entered, prompt the user for a password after identifying which type of password (ie. UNIX, etc.) is being requested. Support for this scheme allows the user to try to use only one password for multiple schemes, and type multiple passwords only if necessary.

If an unsupported option is passed to the modules, it should syslog the error at **LOG_ERR** level.

The permission bits on the service module should be set such that it is not writable by either "group" or "other." The PAM framework will not load the module if the above permission rules are not followed.

**ERROR LOGGING**  
If there are any errors, the modules should log them using **syslog(3)** at the **LOG_ERR** level.

**RETURN VALUES**  
The PAM service module functions may return any of the PAM error numbers specified in the specific man pages. It can also return a **PAM_IGNORE** error number to mean that the PAM framework should ignore this module regardless of whether it is required, optional or sufficient. This error number is normally returned when the module does not want to deal with the given user at all.
ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT Level</td>
<td>MT-Safe with exceptions</td>
</tr>
</tbody>
</table>

SEE ALSO

pam(3), pam_authenticate(3), pam_chauthtok(3), pam_get_user(3),
pam_open_session(3), pam_setcred(3), pam_set_item(3), pam_sm_authenticate(3),
pam_sm_chauthtok(3), pam_sm_open_session(3), pam_sm_setcred(3), pam_start(3),
pam_strerror(3), syslog(3), pam.conf(4), attributes(5)

NOTES

The interfaces in libpam() are MT-Safe only if each thread within the multi-threaded application uses its own PAM handle.
NAME pam_sm_acct_mgmt – service provider implementation for pam_acct_mgmt

SYNOPSIS

```
cc [ flag ...] file ... -lpam [ library ...]
#include <security/pam_appl.h>
#include <security/pam_modules.h>
int pam_sm_acct_mgmt(pam_handle_t *pamh, int flags, int argc, const char **argv);
```

DESCRIPTION

In response to a call to `pam_acct_mgmt(3)`, the PAM framework calls `pam_sm_acct_mgmt()` from the modules listed in the `pam.conf(4)` file. The account management provider supplies the back-end functionality for this interface function. Applications should not call this API directly.

The function `pam_sm_acct_mgmt()` determines whether or not the current user’s account and password are valid. This includes checking for password and account expiration, and valid login times. The user in question is specified by a prior call to `pam_start()`, and is referenced by the authentication handle, `pamh`, which is passed as the first argument to `pam_sm_acct_mgmt()`. The following flags may be set in the `flags` field:

- **PAM_SILENT**: The account management service should not generate any messages.
- **PAM_DISALLOW_NULL_AUTHTOK**: The account management service should return `PAM_NEW_AUTHTOK_REQD` if the user has a null authentication token.

The `argc` argument represents the number of module options passed in from the configuration file `pam.conf(4)`. `argv` specifies the module options, which are interpreted and processed by the account management service. Please refer to the specific module man pages for the various available options. If an unknown option is passed to the module, an error should be logged through `syslog(3)` and the option ignored.

If an account management module determines that the user password has aged or expired, it should save this information as state in the authentication handle, `pamh`, using `pam_set_data()`. `pam_chauthok()` uses this information to determine which passwords have expired.

RETURN VALUES

If there are no restrictions to logging in, `PAM_SUCCESS` is returned. The following error values may also be returned upon error:

- **PAM_USER_UNKNOWN**: User not known to underlying authentication module.
- **PAM_NEW_AUTHTOK_REQD**: New authentication token required.
- **PAM_ACCT_EXPIRED**: User account has expired.

modified 9 Jan 1996

SunOS 5.6

3-1135
PAM_PERM_DENIED  User denied access to account at this time.
PAM_IGNORE  Ignore underlying account module regardless of whether the control flag is required, optional or sufficient.

ATTRIBUTES  See attributes(5) for description of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT Level</td>
<td>MT-Safe with exceptions</td>
</tr>
</tbody>
</table>

SEE ALSO  pam(3), pam_acct_mgmt(3), pam_set_data(3), pam_start(3), syslog(3), libpam(4), pam.conf(4), attributes(5)

NOTES  The interfaces in libpam() are MT-Safe only if each thread within the multi-threaded application uses its own PAM handle.
NAME
pam_sm_authenticate – service provider implementation for pam_authenticate

SYNOPSIS
cc [flag ... ] file ... -lpam [library ...]
#include <security/pam_appl.h> #include <security/pam_modules.h>
int pam_sm_authenticate(pam_handle_t *pamh, int flags, int argc, const char **argv);

DESCRIPTION
In response to a call to pam_authenticate(3), the PAM framework calls
pam_sm_authenticate() from the modules listed in the pam.conf(4) file. The authentication
provider supplies the back-end functionality for this interface function.
The function, pam_smAuthenticate(), is called to verify the identity of the current user.
The user is usually required to enter a password or similar authentication token depend-
ing upon the authentication scheme configured within the system. The user in question is
specified by a prior call to pam_start(), and is referenced by the authentication handle,
pamh.

If the user is unknown to the authentication service, the service module should mask this
error and continue to prompt the user for a password. It should then return the error,
PAM_USER_UNKNOWN.
The following flag may be passed in to pam_sm_authenticate():
PAM_SILENT The authentication service should not generate any mes-
sages.
PAM_DISALLOW_NULL_AUTHTOK The authentication service should return
PAM_AUTH_ERROR The user has a null authentication token.
The argc argument represents the number of module options passed in from the
configuration file pam.conf(4). argv specifies the module options, which are interpreted
and processed by the authentication service. Please refer to the specific module man
pages for the various available options. If any unknown option is passed in, the module
should log the error and ignore the option.

Before returning, pam_sm_authenticate() should call pam_get_item() and retrieve
PAM_AUTHTOK. If it has not been set before and the value is NULL,
pam_sm_authenticate() should set it to the password entered by the user using
pam_set_item().

An authentication module may save the authentication status (success or reason for
failure) as state in the authentication handle using pam_set_data(3). This information is
intended for use by pam_setcred().

RETURN VALUES
Upon successful completion, PAM_SUCCESS must be returned. In addition, the follow-
ing values may be returned:
PAM_MAXTRIES Maximum number of authentication attempts exceeded.
PAM_AUTH_ERR Authentication failure.
PAM_CRED_INSUFFICIENT Cannot access authentication data due to insufficient
PAM_AUTHINFO_UNAVAIL  Underlying authentication service can not retrieve authentication information.
PAM_USER_UNKNOWN  User not known to underlying authentication module.
PAM_IGNORE  Ignore underlying authentication module regardless of whether the control flag is required, optional, or sufficient.

ATTRIBUTES  See attributes(5) for description of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT Level</td>
<td>MT-Safe with exceptions</td>
</tr>
</tbody>
</table>

SEE ALSO  pam(3), pam_authenticate(3), pam_get_item(3), pam_set_data(3), pam_set_item(3), pam_setcred(3), pam_start(3), libpam(4), pam.conf(4), attributes(5)

NOTES  Modules should not retry the authentication in the event of a failure. Applications handle authentication retries and maintain the retry count. To limit the number of retries, the module can return a PAM_MAXTRIES error.

The interfaces in libpam() are MT-Safe only if each thread within the multi-threaded application uses its own PAM handle.
NAME    pam_sm_chauthtok – service provider implementation for pam_chauthtok

SYNOPSIS    cc [ flag ... ] file ... -lpam [ library ... ]
#include <security/pam_appl.h> #include <security/pam_modules.h>
int pam_sm_chauthtok(pam_handle_t *pamh, const int flags);

DESCRIPTION    In response to a call to pam_chauthtok() the PAM framework calls
pam_sm_chauthtok(3) from the modules listed in the pam.conf(4) file. The password
management provider supplies the back-end functionality for this interface function.
pam_sm_chauthtok() changes the authentication token associated with a particular user
referenced by the authentication handle, pamh.

The following flag may be passed in to pam_chauthtok():

PAM_SILENT    The password service should not generate any messages.
PAM_CHANGE_EXPIRED_AUTHTOK    The password service should only update those passwords
that have aged. If this flag is not passed, the password service
should update all passwords.
PAM_PRELIM_CHECK    The password service should only perform preliminary
checks. No passwords should be updated.
PAM_UPDATE_AUTHTOK    The password service should update passwords.

Note that PAM_PRELIM_CHECK and PAM_UPDATE_AUTHTOK cannot be set at the same
time.

Upon successful completion of the call, the authentication token of the user will be ready
for change or will be changed, depending upon the flag, in accordance with the authenti-
cation scheme configured within the system.

The argc argument represents the number of module options passed in from the
configuration file pam.conf(4). argv specifies the module options, which are interpreted
and processed by the password management service. Please refer to the specific module
man pages for the various available options.

It is the responsibility of pam_sm_chauthtok() to determine if the new password meets
certain strength requirements. pam_sm_chauthtok() may continue to re-prompt the user
(for a limited number of times) for a new password until the password entered meets the
strength requirements.

Before returning, pam_sm_chauthtok() should call pam_get_item() and retrieve both
PAM_AUTHTOK and PAM_OLD_AUTHTOK. If both are NULL, pam_sm_chauthtok()
should set them to the new and old passwords as entered by the user.

RETURN VALUES    Upon successful completion, PAM_SUCCESS must be returned. The following values
may also be returned:

PAM_PERM_DENIED    No permission.
### ATTRIBUTES
See attributes(5) for description of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT Level</td>
<td>MT-Safe with exceptions</td>
</tr>
</tbody>
</table>

### SEE ALSO
ping(1M), pam(3), pam_chauthtok(3), pam_get_data(3), pam_get_item(3), pam_set_data(3), libpam(4), pam.conf(4), attributes(5)

### NOTES
The PAM framework invokes the password services twice. The first time the modules are invoked with the flag, PAM_PRELIM_CHECK. During this stage, the password modules should only perform preliminary checks. For example, they may ping remote name services to see if they are ready for updates. If a password module detects a transient error such as a remote name service temporarily down, it should return PAM_TRY_AGAIN to the PAM framework, which will immediately return the error back to the application. If all password modules pass the preliminary check, the PAM framework invokes the password services again with the flag, PAM_UPDATE_AUTHTOK. During this stage, each password module should proceed to update the appropriate password. Any error will again be reported back to application.

If a service module receives the flag PAM_CHANGE_EXPIRED_AUTHTOK, it should check whether the password has aged or expired. If the password has aged or expired, then the service module should proceed to update the password. If the status indicates that the password has not yet aged or expired, then the password module should return PAM_IGNORE.

If a user’s password has aged or expired, a PAM account module could save this information as state in the authentication handle, pamh, using pam_set_data(). The related password management module could retrieve this information using pam_get_data() to determine whether or not it should prompt the user to update the password for this particular module.

The interfaces in libpam() are MT-Safe only if each thread within the multi-threaded application uses its own PAM handle.
NAME
pam_sm_open_session, pam_sm_close_session – service provider implementation for pam_open_session and pam_close_session

SYNOPSIS
cc [flag ...] file ... -lpam [library ...]
#include <security/pam_appl.h>
#include <security/pam_modules.h>

int pam_sm_open_session(pam_handle_t *pamh, int flags, int argc, const char **argv);
int pam_sm_close_session(pam_handle_t *pamh, int flags, int argc, const char **argv);

DESCRIPTION
In response to a call to pam_open_session(3) and pam_close_session(3), the PAM framework calls pam_sm_open_session() and pam_sm_close_session(), respectively from the modules listed in the pam.conf(4) file. The session management provider supplies the back-end functionality for this interface function.

pam_sm_open_session() is called to initiate session management.
pam_sm_close_session() is invoked when a session has terminated. The argument pamh is an authentication handle. The following flag may be set in the flags field:

PAM_SILENT Session service should not generate any messages.

The argc argument represents the number of module options passed in from the configuration file pam.conf(4). argv specifies the module options, which are interpreted and processed by the session management service. If an unknown option is passed in, an error should be logged through syslog(3) and the option ignored.

RETURN VALUES
Upon successful completion, PAM_SUCCESS should be returned. The following values may also be returned upon error:

PAM_SESSION_ERR Cannot make or remove an entry for the specified session.
PAM_IGNORE Ignore underlying session module regardless of whether the control flag is required, optional or sufficient.

ATTRIBUTES
See attributes(5) for description of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT Level</td>
<td>MT-Safe with exceptions</td>
</tr>
</tbody>
</table>

SEE ALSO
pam(3), pam_open_session(3), syslog(3), libpam(4), pam.conf(4), attributes(5)

NOTES
The interfaces in libpam() are MT-Safe only if each thread within the multi-threaded application uses its own PAM handle.

modified 28 Oct 1996

SunOS 5.6
3-1141
NAME
pam_sm_setcred – service provider implementation for pam_setcred

SYNOPSIS
cc [flag ...] file ... -lpam [library ...]
#include <security/pam_appl.h>
#include <security/pam_modules.h>
int pam_sm_setcred(pam_handle_t *pamh, int flags, int argc, const char **argv);

DESCRIPTION
In response to a call to pam_setcred(3), the PAM framework calls pam_sm_setcred() from the modules listed in the pam.conf(4) file. The authentication provider supplies the back-end functionality for this interface function.
pam_sm_setcred() is called to set the credentials of the current user associated with the authentication handle, pamh. The following flags may be set in the flags field. Note that the first four flags are mutually exclusive:
PAM_ESTABLISH_CRED Set user credentials for the authentication service.
PAM_DELETE_CRED Delete user credentials associated with the authentication service.
PAM_REINITIALIZE_CRED Reinitialize user credentials.
PAM_REFRESH_CRED Extend lifetime of user credentials.
PAM_SILENT Authentication service should not generate messages

If no flag is set, PAM_ESTABLISH_CRED is used as the default.
The argc argument represents the number of module options passed in from the configuration file pam.conf(4). argv specifies the module options, which are interpreted and processed by the authentication service. If an unknown option is passed to the module, an error should be logged and the option ignored.

If the PAM_SILENT flag is not set, then pam_sm_setcred() should print any failure status from the corresponding pam_sm_authenticate() function using the conversation function.
The authentication status (success or reason for failure) is saved as module-specific state in the authentication handle by the authentication module. The status should be retrieved using pam_get_data(), and used to determine if user credentials should be set.

RETURN VALUES
Upon successful completion, PAM_SUCCESS should be returned. The following values may also be returned upon error:
PAM_CRED_UNAVAIL Underlying authentication service can not retrieve user credentials.
PAM_CRED_EXPIRED User credentials have expired.
PAM_USER_UNKNOWN User unknown to the authentication service.
PAM_CRED_ERR  Failure in setting user credentials.
PAM_IGNORE    Ignore underlying authentication module regardless of whether the control flag is required, optional, or sufficient.

ATTRIBUTES See attributes(5) for description of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT Level</td>
<td>MT-Safe with exceptions</td>
</tr>
</tbody>
</table>

SEE ALSO pam(3), pam_authenticate(3), pam_get_data(3), pam_setcred(3), pam_sm_authenticate(3), libpam(4), pam.conf(4), attributes(5)

NOTES pam_sm_setcred() is passed the same module options that are used by pam_sm_authenticate().

The interfaces in libpam() are MT-Safe only if each thread within the multi-threaded application uses its own PAM handle.
NAME
pam_start, pam_end – authentication transaction routines for PAM

SYNOPSIS
cc [ flag ... ] file ... -lpam [ library ... ]
#include <security/pam_appl.h>

int pam_start(const char *service, const char *user,
               const struct pam_conv *pam_conv, pam_handle_t **pamh);
int pam_end(pam_handle_t *pamh, int status);

DESCRIPTION
pam_start() is called to initiate an authentication transaction. pam_start() takes as arguments the name of the current service, service, the name of the user to be authenticated, user, the address of the conversation structure, pam_conv, and the address of a variable to be assigned the authentication handle pamh. Upon successful completion, pamh will refer to a PAM handle for use with subsequent calls to the authentication library.

The structure pam_conv contains the address of the conversation function provided by the application. The underlying PAM service module invokes this function to output information to and retrieve input from the user. The pam_conv structure has the following entries:

```
struct pam_conv {
    int (*conv)();  /* Conversation function */
    void *appdata_ptr;  /* Application data */
};
```

where

```
int conv(int num_msg,
         const struct pam_message **msg,
         struct pam_response **resp,
         void *appdata_ptr);
```

The function conv() is called by a service module to hold a PAM conversation with the application or user. For window applications, the application can create a new pop-up window to be used by the interaction.

The parameter num_msg is the number of messages associated with the call. The parameter msg is a pointer to an array of length num_msg of the pam_message structure.

The structure pam_message is used to pass prompt, error message, or any text information from the authentication service to the application or user. It is the responsibility of the PAM service modules to localize the messages. The memory used by pam_message has to be allocated and freed by the PAM modules. The pam_message structure has the following entries:

```
struct pam_message{
    int msg_style;
    char *msg;
};
```
The message style, `msg_style`, can be set to one of the following values:

- **PAM_PROMPT_ECHO_OFF** Prompt user, disabling echoing of response.
- **PAM_PROMPT_ECHO_ON** Prompt user, enabling echoing of response.
- **PAM_ERROR_MSG** Print error message.
- **PAM_TEXT_INFO** Print general text information.

The maximum size of the message and the response string is `PAM_MAX_MSG_SIZE` as defined in `<security/pam.appl.h>`.

The structure `pam_response` is used by the authentication service to get the user’s response back from the application or user. The storage used by `pam_response` has to be allocated by the application and freed by the PAM modules. The `pam_response` structure has the following entries:

```c
struct pam_response{
    char *resp; /* currently not used, */
    int resp_retcode; /* should be set to 0 */
};
```

It is the responsibility of the conversation function to strip off NEWLINE characters for `PAM_PROMPT_ECHO_OFF` and `PAM_PROMPT_ECHO_ON` message styles, and to add NEWLINE characters (if appropriate) for `PAM_ERROR_MSG` and `PAM_TEXT_INFO` message styles.

`appdata_ptr` is an application data pointer which is passed by the application to the PAM service modules. Since the PAM modules pass it back through the conversation function, the applications can use this pointer to point to any application-specific data.

`pam_end()` is called to terminate the authentication transaction identified by `pamh` and to free any storage area allocated by the authentication module. The argument, `status`, is passed to the `cleanup(|)` function stored within the `pam` handle, and is used to determine what module-specific state must be purged. A cleanup function is attached to the handle by the underlying PAM modules through a call to `pam_set_item(3)` to free module specific data.

**RETURN VALUES** Refer to `pam(3)` for information on error related return values.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT Level</td>
<td>MT-Safe with exceptions</td>
</tr>
</tbody>
</table>

**SEE ALSO** `pam(3), pam_acct_mgmt(3), pam_authorize(3), pam_chauthtok(3), pam_open_session(3), pam_setcred(3), pam_set_item(3), pam_strerror(3), attributes(5)`

modified 28 Oct 1996             SunOS 5.6             3-1145
NOTES

The interfaces in `libpam()` are MT-Safe only if each thread within the multi-threaded application uses its own PAM handle.
NAME  pam_strerror – get PAM error message string

SYNOPSIS  cc [ flag ... ] file ... -lpam [ library ... ]
#include <security/pam_appl.h>
const char *pam_strerror(pam_handle_t*pamh, int errnum);

DESCRIPTION  pam_strerror() maps the PAM error number in errnum to a PAM error message string,
and returns a pointer to that string. The application should not free or modify the string
returned.

The pamh argument is the PAM handle obtained by a prior call to pam_start(). If
pam_start() returns an error, a null PAM handle should be passed.

ERRORS  pam_strerror() returns NULL if errnum is out-of-range.

ATTRIBUTES  See attributes(5) for description of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT Level</td>
<td>MT-Safe with exceptions</td>
</tr>
</tbody>
</table>

SEE ALSO  pam(3), pam_start(3), attributes(5)

NOTES  The interfaces in libpam() are MT-Safe only if each thread within the multi-threaded
application uses its own PAM handle.
NAME
panels – character based panels package

SYNOPSIS
#include <panel.h>

DESCRIPTION
The panel library is built using the curses library, and any program using panels routines must call one of the curses initialization routines such as initscr. A program using these routines must be compiled with -lp panel 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.

<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Manual Page Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>bottom_panel</td>
<td>panel_top(3X)</td>
</tr>
<tr>
<td>del_panel</td>
<td>panel_new(3X)</td>
</tr>
<tr>
<td>hide_panel</td>
<td>panel_show(3X)</td>
</tr>
<tr>
<td>move_panel</td>
<td>panel_move(3X)</td>
</tr>
<tr>
<td>new_panel</td>
<td>panel_new(3X)</td>
</tr>
<tr>
<td>panel_above</td>
<td>panel_above(3X)</td>
</tr>
<tr>
<td>panel_below</td>
<td>panel_above(3X)</td>
</tr>
<tr>
<td>panel_hidden</td>
<td>panel_show(3X)</td>
</tr>
<tr>
<td>panel_userptr</td>
<td>panel_userptr(3X)</td>
</tr>
<tr>
<td>panel_window</td>
<td>panel_window(3X)</td>
</tr>
<tr>
<td>replace_panel</td>
<td>panel_window(3X)</td>
</tr>
<tr>
<td>set_panel_userptr</td>
<td>panel_userptr(3X)</td>
</tr>
<tr>
<td>show_panel</td>
<td>panel_show(3X)</td>
</tr>
<tr>
<td>top_panel</td>
<td>panel_top(3X)</td>
</tr>
<tr>
<td>update_panels</td>
<td>panel_update(3X)</td>
</tr>
</tbody>
</table>

RETURN VALUES
Each panels routine that returns a pointer to an object returns NULL if an error occurs. Each panel routine that returns an integer, returns OK if it executes successfully and ERR if it does not.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
<tr>
<td>SEE ALSO</td>
<td>curses(3X), attributes(5) and 3X pages whose names begin “panel_” for detailed routine descriptions.</td>
</tr>
<tr>
<td>NOTES</td>
<td>The header <code>&lt;panel.h&gt;</code> automatically includes the header <code>&lt;curses.h&gt;</code>.</td>
</tr>
</tbody>
</table>
NAME     panel_above, panel_below – panels deck traversal primitives

SYNOPSIS cc [ flag ... ] file ... -lpanel  -lcurses [ library ... ]
          #include <panel.h>
          PANEL *panel_above(PANEL *panel);
          PANEL *panel_below(PANEL *panel);

DESCRIPTION panel_above() returns a pointer to the panel just above panel, or NULL if panel is the top panel. panel_below() returns a pointer to the panel just below panel, or NULL if panel is the bottom panel.

If NULL is passed for panel, 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.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO curses(3X), panels(3X), attributes(5)

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
cc [ flag ... ] file ... -lpanel -lcurses [ library ... ]
#include <panel.h>
int move_panel(PANEL *panel, int starty, int startx);

DESCRIPTION
move_panel() moves the curses window associated with panel so that its upper left-hand corner is at starty, startx. See usage note, below.

RETURN VALUES
OK is returned if the routine completes successfully, otherwise ERR is returned.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
curses(3X), panel_update(3X), panels(3X), attributes(5)

NOTES
For panels windows, use move_panel() instead of the mvwin() curses routine. Otherwise, update_panels() will not properly update the virtual screen.
The header <panel.h> automatically includes the header <curses.h>.
panel_new (3X)

NAME        panel_new, new_panel, del_panel – create and destroy panels

SYNOPSIS    cc [ flag ... ] file ... -lpanel -lcurses [ library ... ]
             #include <panel.h>
             PANEL *new_panel(WINDOW *win);
             int del_panel(PANEL * panel);

DESCRIPTION  new_panel() creates a new panel associated with win and returns the panel pointer. The
             new panel is placed on top of the panel deck.
             del_panel() destroys panel, but not its associated window.

RETURN VALUES  new_panel() returns NULL if an error occurs.
             del_win() returns OK if successful, ERR otherwise.

ATTRIBUTES  See attributes(5) for descriptions of the following attributes:
             
             | ATTRIBUTE TYPE | ATTRIBUTE VALUE |
             |----------------|-----------------|
             | MT-Level       | Unsafe          |

SEE ALSO      curses(3X), panel_update(3X), panels(3X), attributes(5)

NOTES         The header <panel.h> automatically includes the header <curses.h>.
NAME  panel_show, show_panel, hide_panel, panel_hidden – panels deck manipulation routines

SYNOPSIS  cc [ flag . . . ] file . . . -lpanel -lcurses [ library . . ]
#include <panel.h>
int show_panel(PANEL *panel);
int hide_panel(PANEL *panel);
int panel_hidden(PANEL *panel);

DESCRIPTION  show_panel() makes panel, previously hidden, visible and places it on top of the deck of panels.
hide_panel() removes panel from the panel deck and, thus, hides it from view. The internal data structure of the panel is retained.
panel_hidden() returns TRUE (1) or FALSE (0) indicating whether or not panel is in the deck of panels.

RETURN VALUES  show_panel() and hide_panel() return the integer OK upon successful completion or ERR upon error.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO  curses(3X), panel_update(3X), panels(3X), attributes(5)

NOTES  The header <panel.h> automatically includes the header <curses.h>.

modified 31 Dec 1996  SunOS 5.6  3X-1153
NAME
panel_top, top_panel, bottom_panel – panels deck manipulation routines

SYNOPSIS
cc [ flag ...] file ... -Ipanel -lcurses [ library ... ]
#include <panel.h>
int top_panel(PANEL *panel);
int bottom_panel(PANEL *panel);

DESCRIPTION
top_panel() pulls panel to the top of the desk of panels. It leaves the size, location, and contents of its associated window unchanged.
bottom_panel() puts panel at the bottom of the desk of panels. It leaves the size, location, and contents of its associated window unchanged.

RETURN VALUES
All of these routines return the integer OK upon successful completion or ERR upon error.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
curses(3X), panel_update(3X), panels(3X), attributes(5)

NOTES
The header <panel.h> automatically includes the header <curses.h>.
NAME  panel_update, update_panels – panels virtual screen refresh routine

SYNOPSIS  cc [ flag ... ] file ... -lpanel -lcurses [ library ... ]

           #include <panel.h>
           void update_panels(void);

DESCRIPTION  update_panels() refreshes the virtual screen to reflect the depth relationships between
              the panels in the deck. The user must use the curses library call
doupdate() (see
curs_refresh(3X)) to refresh the physical screen.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO  curs_refresh(3X), curses(3X), panels(3X), attributes(5)

NOTES  The header <panel.h> automatically includes the header <curses.h>.
NAME  panel_userptr, set_panel_userptr – associate application data with a panels panel

SYNOPSIS  cc [ flag ... ] file ... -lpanel -lcurses [ library ... ]
   #include <panel.h>
   int set_panel_userptr(PANEL *panel, char *ptr);
   char *panel_userptr(PANEL *panel);

DESCRIPTION  Each panel has a user pointer available for maintaining relevant information.
   set_panel_userptr() sets the user pointer of panel to ptr.
   panel_userptr() returns the user pointer of panel.

RETURN VALUES  set_panel_userptr returns OK if successful, ERR otherwise.
   panel_userptr returns NULL if there is no user pointer assigned to panel.

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

  ATTRIBUTE TYPE | ATTRIBUTE VALUE
    MT-Level       | Unsafe

SEE ALSO  curses(3X), panels(3X), attributes(5)

NOTES  The header <panel.h> automatically includes the header <curses.h>.
NAME  panel_window, replace_panel – get or set the current window of a panels panel

SYNOPSIS  cc [ flag ... ] file ... -lpanel -lcurses [ library ... ]
#include <panel.h>
WINDOW *panel_window(PANEL *panel);
int replace_panel(PANEL *panel, WINDOW *win);

DESCRIPTION  panel_window() returns a pointer to the window of panel.
replace_panel() replaces the current window of panel with win.

RETURN VALUES  panel_window() returns NULL on failure.
replace_panel() returns OK on successful completion, ERR otherwise.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO  curses(3X), panels(3X), attributes(5)

NOTES  The header <panel.h> automatically includes the header <curses.h>.

modified 31 Dec 1996  SunOS 5.6  3X-1157
NAME  pathfind – search for named file in named directories

SYNOPSIS  cc [ flag . . . ] file . . . -lgen [ library . . . ]
  #include <libgen.h>
  char *pathfind(const char *path, const char *name, const char *mode);

DESCRIPTION  pathfind() searches the directories named in path for the file name. The directories named
  in path are separated by semicolons. mode is a string of option letters chosen from the set
  [rwxfbcdpugks]:

<table>
<thead>
<tr>
<th>Letter</th>
<th>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>

Options read, write, and execute are checked relative to the real (not the effective) user ID
and group ID of the current process.

If the file name, with all the characteristics specified by mode, is found in any of the direc-
tories specified by path, then pathfind() returns a pointer to a string containing the
member of path, followed by a slash character (/), followed by name.

If name begins with a slash, it is treated as an absolute path name, and path is ignored.
An empty path member is treated as the current directory. / is not prepended at the
occurrence of the first match; rather, the unadorned name is returned.

EXAMPLES  To find the ls command using the PATH environment variable:

  pathfind (getenv ("PATH"), "ls", "rx")
RETURN VALUES
If no match is found, `pathname` returns a null pointer, `((char *) 0)`.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
`sh(1), test(1), access(2), mknod(2), stat(2), getenv(3C), attributes(5)`

NOTES
The string pointed to by the returned pointer is stored in an area that is reused on subsequent calls to `pathfind()`. The string should not be deallocated by the caller.
When compiling multi-thread applications, the `_REENTRANT` flag must be defined on the compile line. This flag should only be used in multi-thread applications.
NAME  pechochar, pecho_wchar – add character and refresh window

SYNOPSIS  
```
#include <curses.h>
int pechochar(WINDOW *pad, chtype ch);
int pecho_wchar(WINDOW *pad, const chtype *wch);
```

ARGUMENTS  
- `pad`  Is a pointer to the pad in which the character is to be added.
- `ch`  Is a pointer to the character to be written to the pad.
- `wch`  Is a pointer to the complex character to be written to the pad.

DESCRIPTION  The `pechochar()` function is equivalent to calling `waddch(3XC)` followed by a call to `prefresh(3XC)`. The `pecho_wchar()` function is equivalent to calling `wadd_wch(3XC)` followed by a call to `prefresh()`. `prefresh()` reuses the last position of the pad on the screen for its parameters.

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

ERRORS  None.

SEE ALSO  `add_wch(3XC)`, `addch(3XC)`, `newpad(3XC)`

3XC-1160  SunOS 5.6  modified 1 Jun 1996
NAME perror, errno – print system error messages

SYNOPSIS
#include <stdio.h>
void perror(const char *s);
#include <errno.h>
int errno;

DESCRIPTION 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 s is printed first, then a colon and a blank, then the message and a newline. (However, if s 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.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO intro(2), fmtmsg(3C), gettext(3C), setlocale(3C), strerror(3C), attributes(5)

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 */);

DESCRIPTION
pfmt() retrieves a format string from a locale-specific message database (unless MM_NOGET is specified) and uses it for printf() style formatting of args. The output is displayed on stream.
pfmt() encapsulates the output in the standard error message format (unless MM_NOSTD is specified, in which case the output is simply printf() like).
If the printf() format string is to be retrieved from a message database, the format argument must have the following structure:
<catalog>:<msgnum>:<defmsg>.

If MM_NOGET is specified, only the <defmsg> part must be specified.
<catalog> is used to indicate the message database that contains the localized version of the format string. <catalog> 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).
<msgnum> is a positive number that indicates the index of the string into the message database.
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 <defmsg> part of the format argument.
If <catalog> is omitted, pfmt() will attempt to retrieve the string from the default catalog specified by the last call to setcat(). In this case, the format argument has the following structure:
<msgnum>:<defmsg>.
pfmt() will output Message not found!!\n as format string if <catalog> is not a valid catalog name, if no catalog is specified (either explicitey or via setcat()), if <msgnum> is not a valid number, or if no message could be retrieved from the message databases, and <defmsg> was omitted.
The flags determine the type of output (i.e. whether the format should be interpreted as is or encapsulated in the standard message format), and the access to message catalogs to retrieve a localized version of format.
The flags 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 |

3C-1162 SunOS 5.6 modified 29 Dec 1996
### Catalog access control

- **MM_STD**: Output using the standard message format (default, value 0).
- **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`.

`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:

\[-1 \quad \text{write error to } \text{stream.}\]

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 ...");
```

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.

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

```
<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-safe</td>
</tr>
</tbody>
</table>
```

SEE ALSO
`addsev(3C)`, `gettext(3C)`, `lfmt(3C)`, `printf(3S)`, `setcat(3C)`, `setLabel(3C)`, `setlocale(3C)`, `attributes(5)`, `environ(5)`
NAME       plock – lock or unlock into memory process, text, or data

SYNOPSIS   #include <sys/lock.h>
            int plock(int op);

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 userID of the calling process must be super-user to use this call. plock() performs the function specified by op:

            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:
            EAGAIN     Not enough memory.
            EINVAL    op is equal to PROCLOCK and a process lock, a text lock, or a data lock already exists on the calling process.
            EINVAL    op is equal to TXTLOCK and a text lock, or a process lock already exists on the calling process.
            EINVAL    op is equal to DATLOCK and a data lock, or a process lock already exists on the calling process.
            EINVAL    op 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.

modified 25 Jan 1993
NAME  plot, arc, box, circle, closepl, closevt, cont, erase, label, line, linmod, move, openpl, openvt, point, space – graphics interface

SYNOPSIS  void arc(short x0, short y0, short x1, short y1, short x2, short y2);
void box(short x0, short y0, short x1, short y1);
void circle(short x, short y, short r);
void closepl();
void closevt();
void cont(short x, short y);
void erase();
void label(char *s);
void line(short x0, short y0, short x1, short y1);
void linmod(char *s);
void move(short x, short y);
void openpl();
void openvt();
void point(short x, short y);
void space(short x0, short y0, short x1, short y1);

DESCRIPTION  These routines generate graphics output for a set of output devices. The format of the output is dependent upon which link editor option is used when the program is compiled and linked (see Link Editor).

The term “current point” refers to the current setting for the x and y coordinates.

arc() specifies a circular arc. The coordinates (x0, y0) specify the center of the arc. The coordinates (x1, y1) specify the starting point of the arc. The coordinates (x2, y2) specify the end point of the circular arc.

box() specifies a rectangle with coordinates (x0, y0), (x0, y1), (x1, y0), and (x1, y1). The current point is set to (x1, y1).

circle() specifies a circle with a center at the coordinates (x, y) and a radius of r.

closevt() and closepl() flush the output.

cont() specifies a line beginning at the current point and ending at the coordinates (x, y). The current point is set to (x, y).

erase() starts another frame of output.

label() places the null terminated string s so that the first character falls on the current point. The string is then terminated by a NEWLINE character.

line() draws a line starting at the coordinates (x0, y0) and ending at the coordinates (x1, y1). The current point is set to (x1, y1).
linmod() specifies the style for drawing future lines. `s` may contain one of the following:
- `dotted`
- `solid`
- `longdashed`
- `shortdashed`
- `dotdashed`

move() sets the current point to the coordinates `(x, y)`.

openpl() or openvt() must be called to open the device before any other plot routines are called.

point() plots the point given by the coordinates `(x, y)`. The current point is set to `(x, y)`.

space() specifies the size of the plotting area. The plot will be reduced or enlarged as necessary to fit the area specified. The coordinates `(x0, y0)` specify the lower left hand corner of the plotting area. The coordinates `(x1, y1)` specify the upper right hand corner of the plotting area.

### Link Editor
Various flavors of these routines exist for different output devices. They are obtained by using the following `ld(1)` options:

- `-lplot` device-independent graphics stream on standard output in the format described in `plot(4B)`
- `-l300` GSI 300 terminal
- `-l300s` GSI 300S terminal
- `-l4014` Tektronix 4014 terminal
- `-l450` GSI 450 terminal
- `-lvto`

### FILES
- `/usr/lib/libplot.a`
- `/usr/lib/lib300.a`
- `/usr/lib/lib300s.a`
- `/usr/lib/lib4014.a`
- `/usr/lib/lib450.a`
- `/usr/lib/libvt0.a`

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

### SEE ALSO
- `graph(1)`, `ld(1)`, `plot(4B)`, `attributes(5)`

modified 29 Dec 1996

SunOS 5.6
NAME
popen, pclose – initiate pipe to/from a process

SYNOPSIS
#include <stdio.h>

FILE ∗popen(const char ∗command, const char ∗type);

int pclose (FILE ∗stream);

DESCRIPTION
The `popen()` function creates a pipe between the calling program and the command to be executed. The arguments to `popen()` are pointers to null-terminated strings. `command` consists of a shell command line. `type` 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 `stream` (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 `stream`. Because open files are shared, a type `r` command may be used as an input filter and a type `w` as an output filter.

The environment of the executed command will be as if a child process were created within the `popen()` call using `fork(2)`. If the application is standard-conforming (see `standards(5)`), the child is invoked with the call:

```
exec("/usr/bin/ksh", "ksh", "−c", command, (char ∗)0);
```

otherwise, the child is invoked with the call:

```
exec("/usr/bin/sh", "sh", "−c", command, (char ∗)0);
```

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.

RETURN VALUES
The `popen()` function returns a null pointer if files or processes cannot be created.

The `pclose()` function returns the termination status of the command. The `pclose()` function returns −1 if `stream` is not associated with a `popen()` command and sets `errno` to indicate the error.

EXAMPLES
The following is an example of a typical call:

```c
#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);
```

SunOS 5.6 modified 26 Feb 1997
This program will print on the standard output (see `stdio(3S)`) all the file names in the current directory that have a `.c` suffix.

**ATTRIBUTES**

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

**SEE ALSO**

`ksh(1), pipe(2), wait(2), waitpid(2), fclose(3S), fopen(3S), stdio(3S), system(3S), attributes(5), wstat(5), standards(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 (" \").
NAME  pow – power function

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

DESCRIPTION  The pow() function computes the value of x raised to the power y, \( x^y \). If x is negative, y must be an integer value.

RETURN VALUES  Upon successful completion, pow() returns the value of x raised to the power y.
If x is 0 and y is 0, 1.0 is returned.
If y is NaN, or y is non-zero and x is NaN, NaN is returned. If y is 0.0 and x is NaN, NaN is returned.
If x is 0.0 and y is negative, \( -HUGE\_VAL \) is returned and errno may be set to EDOM or ERANGE.
If the correct value would cause overflow, \( +HUGE\_VAL \) is returned, and errno is set to ERANGE.
If the correct value would cause underflow to 0, 0 is returned and errno may be set to ERANGE.
For exceptional cases, matherr(3M) tabulates the values to be returned as dictated by Standards other than XPG4.

ERRORS  The pow() function will fail if:
EDOM   The value of x is negative and y is non-integral.
ERANGE The value to be returned would have caused overflow.
The pow() function may fail if:
EDOM   The value of x is 0.0 and y is negative.
ERANGE The correct value would cause underflow.

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

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  exp(3M), isnan(3M), matherr(3M), attributes(5), standards(5)
NAME
printf, fprintf, sprintf, vprintf, vfprintf, vsprintf – formatted output conversion

SYNOPSIS
/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 vfprintf( stream, format, ap )
FILE *stream;
char *format;
va_list ap;

char *vsprintf( s, format, ap )
char *s, *format;
va_list ap;

DESCRIPTION
printf() places output on the standard output stream stdout. fprintf() places output on
the named output stream. sprintf() places “output,” followed by the NULL character
(NULL), in consecutive bytes starting at *s; it is the user’s responsibility to ensure that
enough storage is available.

vprintf(), vfprintf(), and vsprintf() are the same as printf(), fprintf(), and sprintf()
respectively, except that instead of being called with a variable number of arguments,
they are called with an argument list as defined by varargs(5).

Each of these functions converts, formats, and prints its args under control of the format.
The format is a character string which contains two types of objects: plain characters,
which are simply copied to the output stream, and conversion specifications, each of
which causes conversion and printing of zero or more 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 simply ignored.

Each conversion specification is introduced by the character %. After the %, the follow-
ing appear in sequence:
Zero or more flags, which modify the meaning of the conversion specification. An optional decimal digit string specifying a minimum field width. If the converted value has fewer characters than the field width, it will be padded on the left (or right, if the left-adjustment flag ‘−’, described below, has been given) to the field width. The padding is with blanks unless the field width digit string starts with a zero, in which case the padding is with zeros.

A precision that gives the minimum number of digits to appear for the d, i, o, u, x, or X conversions, the number of digits to appear after the decimal point for the e, E, and f conversions, the maximum number of significant digits for the G and g conversion, or the maximum number of characters to be printed from a string in s conversion. The precision takes the form of a period (.) followed by a decimal digit string; a NULL digit string is treated as zero. Padding specified by the precision overrides the padding specified by the field width.

An optional l (ell) specifying that a following d, i, o, u, x, or X conversion character applies to a long integer arg. An l before any other conversion character is ignored.

A character that indicates the type of conversion to be applied. A field width or precision or both may be indicated by an asterisk (∗) instead of a digit string. In this case, an integer arg supplies the field width or precision. The arg that is actually converted is not fetched until the conversion letter is seen, so the 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 abcdedef are used for x conversion and the letters ABCDEF for X conversion. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it will
be expanded with leading zeroes. (For compatibility with older versions, padding with leading zeroes may alternatively be specified by prepending a zero to the field width. This does not imply an octal value for the field width.) The default precision is 1. The result of converting a zero value with a precision of zero is a NULL string.

f The float or double arg is converted to decimal notation in the style \([-]ddd.ddd\) where the number of digits after the decimal point is equal to the precision specification. If the precision is missing, 6 digits are given; if the precision is explicitly 0, no digits and no decimal point are printed.

e,E The float or double arg is converted in the style \([-]d.ddd[e\pm]ddd\), 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.

RETURN VALUES Upon success, printf() and fprintf() return the number of characters transmitted, excluding the null character. vprintf() and vfprintf() return the number of characters transmitted. sprintf() and vsprintf() always return s. If an output error is encountered, printf(), fprintf(), vprintf(), and vfprintf() return EOF.

EXAMPLES To print a date and time in the form “Sunday, July 3, 10:02,” where weekday and month are pointers to NULL-terminated strings:

modified 18 Feb 1993  SunOS 5.6  3B-1173
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, snprintf – print formatted output

SYNOPSIS

#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 */);
int snprintf(char *s, size_t n, const char *format, /* args */);

DESCRIPTION

The printf() function places output on the standard output stream stdout.
The fprintf() function places output on strm.
The sprintf() function 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.
The snprintf() function is identical to sprintf() with the addition of the argument n, which specifies the size of the buffer referred to by s.

Each of these functions converts, formats, and prints its args under control of the format.
The format is a character string that contains three types of objects defined below:
1. plain characters that are simply copied to the output stream;
2. escape sequences that represent non-graphic characters;
3. conversion specifications.

The following escape sequences produce the associated action on display devices capable of the action:
\a Alert. Ring the bell.
\b Backspace. Move the printing position to one character before the current position, unless the current position is the start of a line.
\f Form feed. Move the printing position to the initial printing position of the next logical page.
\n Newline. Move the printing position to the start of the next line.
\r Carriage return. Move the printing position to the start of the current line.
\t Horizontal tab. Move the printing position to the next implementation-defined horizontal tab position on the current line.
\v Vertical tab. Move the printing position to the start of the next implementation-defined vertical tab position.

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 (.)

modified 27 Feb 1997

SunOS 5.6

3S-1175
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 conversion character is s, a standard-conforming application (see standards(5)) interprets the field width as the minimum number of bytes to be printed; an application that is not standard-conforming interprets the field width as the minimum number of columns of screen display. For an application that is not standard-conforming, %10s means if the converted value has a screen width of 7 columns, 3 spaces would be padded on the right.

If the format is %ws, then the field width should be interpreted as the minimum number of columns of screen display.

An optional precision that gives the minimum number of digits to appear for the d, i, o, u, x, or X conversions (the field is padded with leading zeros), the number of digits to appear after the decimal-point character for the e, E, and f conversions, the maximum number of significant digits for the g and G conversions. If the conversion character is s, a standard-conforming application (see standards(5)) interprets the precision as the maximum number of bytes to be written; an application that is not standard-conforming interprets the precision as the maximum number of columns of screen display. For an application that is not standard-conforming, %.5s would print only the portion of the string that would display in 5 screen columns. Only complete characters are written.

For %ws, the precision should be interpreted as the maximum number of columns of screen display. The precision takes the form of a period (.) followed by a decimal digit string; a null digit string is treated as zero. Padding specified by the precision overrides the padding specified by the field width.

An optional h specifies that a following d, i, o, u, x, or X conversion 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 a long int argument. An optional ll (ell ell) specifies that a following d, i, o, u, x, or X conversion specifier applies to a long long or unsigned long long argument; an optional ll (ell ell) specifies that a following n conversion specifier applies to a pointer to a long long 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 Nth 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 `−[ld]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 `args` is converted to the style `−[le]ddd.e±ddd`, 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 `args` is printed in style `f` or `e` (or in style `E` in the case of a `G` conversion character), with the precision specifying the number of significant digits. If the precision is zero, it is taken as one. The style used depends on the value converted: style `e` (or `E`) will be used only if the exponent resulting from the conversion is less than −4 or greater than or equal to the precision. Trailing zeros are removed from the fractional part of the result. A decimal-point character appears only if it is followed by a digit.

- **c**
  - The `int args` is converted to an `unsigned char`, and the resulting character is printed.

- **C**
  - The `wchar_t` argument is converted to an array of bytes representing a character, and the resulting character is written. The conversion is the same as that expected from `wctomb(3C)`.

- **wc**
  - The `int args` is converted to a wide character (`wchar_t`), and the resulting wide character is printed.

- **s**
  - The `args` is taken to be a string (character pointer) and characters from the string are written up to (but not including) a terminating null character. If a precision is specified, a standard-conforming application (see
printf (3S) will write only the number of bytes specified by precision; an application that is not standard-conforming will write only the portion of the string that will display in the number of columns of screen display specified by precision.

If the precision is not specified, it is taken to be infinite, so all characters up to the first null character are printed. A null value for args will yield undefined results.

S
The argument must be a pointer to an array of type wchar_t. Wide-character codes from the array, up to but not including any terminating null wide-character code, are converted to a sequence of bytes, and the resulting bytes are written. If the precision is specified, no more than that many bytes are written, and only complete characters are written. If the precision is not specified, or is greater than the size of the array of converted bytes, the array of wide characters must be terminated by a null wide character. The conversion is the same as that expected from wcstombs(3C).

ws
The args is taken to be a wide character string (wide character pointer) and wide characters from the string are written up to (but not including) a terminating null character; if the precision is specified, only the portion of the wide character string that will display in the number of columns of screen display specified by precision will be written. If the precision is not specified, it is taken to be infinite, so all wide characters up to the first null character are printed. A null value for args will yield undefined results.

p
The args should be a pointer to void. The value of the pointer is converted to 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 %adigits$ 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 [±]Infinity, where Infinity is either Infinity or Inf, depending on the desired output string length. Printing of the sign follows the rules described above.

If a floating-point value is the internal representation for “not-a-number,” the output is [±]NaN. Printing of the sign follows the rules described above.
In no case does a non-existent or small field width cause truncation of a field; if the result of a conversion is wider than the field width, the field is simply expanded to contain the conversion result. Characters generated by `printf()` and `fprintf()` are printed as if the `putc()` routine had been called.

RETURN VALUES The `printf()`, `fprintf()`, and `sprintf()` functions return the number of bytes transmitted (not including the `\0` in the case of `sprintf()`). The `snprintf()` function returns the number of characters formatted, that is, the number of characters that would have been written to the buffer if it were large enough. Each function returns a negative value if an output error was encountered.

ERRORS The `printf()` and `fprintf()` functions will fail if either the `stream` is unbuffered or the `stream`'s buffer needed to be flushed and:

- **EFBIG** The file is a regular file and an attempt was made to write at or beyond the offset maximum.
- **EILSEQ** An invalid character has been detected.

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:

```c
printf("%s, %s %i, %d:%.2d", weekday, month, day, hour, min);
```

To print π to 5 decimal places:

```c
printf("pi = %.5f", 4 * atan(1.0));
```

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe with exceptions</td>
</tr>
<tr>
<td>CSI</td>
<td>Enabled</td>
</tr>
</tbody>
</table>

SEE ALSO `exit(2)`, `lseek(2)`, `write(2)`, `abort(3C)`, `ecvt(3C)`, `putc(3S)`, `scanf(3S)`, `setlocale(3C)`, `stdio(3S)`, `wcstombs(3C)`, `wctomb(3C)`, `attributes(5)`, `environ(5)`, `standards(5)`

NOTES The `sprintf()` function is MT-Safe in multi-thread applications. The `printf` and `fprintf` functions can be used safely in a multi-thread application, as long as `setlocale(3C)` is not being called to change the locale.
NAME  proc_service — process service interfaces

SYNOPSIS  
#include <proc_service.h>

ps_err_e ps_kill(const struct ps_prochandle *ph, int signum)
ps_err_e ps_lcontinue(const struct ps_prochandle *ph, lwpid_t lwpid);

ps_err_e ps_lgetfregs(const struct ps_prochandle *ph,
                      lwpid_t lwpid, prfregset_t *fregset);

ps_err_e ps_lgetregs(const struct ps_prochandle *ph, lwpid_t lwpid,
                      prgregset_t gregset);

ps_err_e ps_lrolltoaddr(const struct ps_prochandle *ph,
                        lwpid_t lwpid,
                        ps_err_e ps_lsetfregs(const struct ps_prochandle *ph,
                                              lwpid_t lwpid, const prfregset_t *fregset);

psaddr_t psaddr_t go_addr, psaddr_t stop_addr)

ps_err_e ps_lgetxregs(const struct ps_prochandle *ph,
                       lwpid_t lwpid, prxregset_t *xregset);

ps_err_e ps_lgetxregsize(const struct ps_prochandle *ph,
                          lwpid_t lwpid, int *xregsize);

sp_err_e ps_lsetfregs(const struct ps_prochandle *ph,
                      lwpid_t lwpid, const prfregset_t *fregset);

ps_err_e ps_lsetxregs(const struct ps_prochandle *ph,
                       lwpid_t lwpid, prxregset_t *xregset);

ps_err_e ps_pcontinue(const struct ps_prochandle *ph);

ps_err_e ps PDTREAD(const struct ps_prochandle *ph, psaddr_t addr,
                    char *buf, int size);

ps_err_e ps PDTWRITE(const struct ps_prochandle *ph, psaddr_t addr,
                     char *buf, int size);

ps_err_e ps_PGLOBALLookup(const struct ps_prochandle *ph,
                           const char *ld_object_name, const char *ld_symbol_name,
                           psaddr_t *ld_symbol_addr);

ps_err_e ps_PSTOP(const struct ps_prochandle *ph);

ps_err_e ps_PTRDATE(const struct ps_prochandle *ph, psaddr_t addr,
                    char *buf, int size);

ps_err_e ps_PTWRITE(const struct ps_prochandle *ph, psaddr_t addr,
                    char *buf, int size);

SPARC Platform
Only

modified 4 Mar 1997 SunOS 5.6 3T-1181
Every program that links `libthread_db` or `librtld_db` must provide a set of process control primitives that will allow `libthread_db` and `librtld_db` to access memory and registers in the target process, to start and to stop the target process, and to look up symbols in the target process. See `libthread_db`(3T). For information on `librtld_db`, refer to the `Linker and Libraries Guide`.

Refer to the individual reference manual pages that describe these routines for a functional specification that clients of `libthread_db` and `librtld_db` can use to implement this required interface. `<proc_service.h>` lists the C and C++ declarations of these routines.

**FUNCTIONS**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ps_pstop()</code></td>
<td>Stops the target process.</td>
</tr>
<tr>
<td><code>ps_pcontinue()</code></td>
<td>Resumes target process.</td>
</tr>
<tr>
<td><code>ps_lstop()</code></td>
<td>Stops a single lightweight process (LWP) within the target process.</td>
</tr>
<tr>
<td><code>ps_lcontinue()</code></td>
<td>Resumes a single LWP within the target process.</td>
</tr>
<tr>
<td><code>ps_pglobal_lookup()</code></td>
<td>Looks up the symbol in the symbol table of the load object in the target process.</td>
</tr>
<tr>
<td><code>ps_pdread()</code></td>
<td>Copies <code>size</code> bytes from the data segment of the target process to the controlling process.</td>
</tr>
<tr>
<td><code>ps_pdwrite()</code></td>
<td>Copies <code>size</code> bytes from the data segment of the controlling process to the target process.</td>
</tr>
<tr>
<td><code>ps_ptread()</code></td>
<td>Copies <code>size</code> bytes from the text segment of the target process to the controlling process.</td>
</tr>
<tr>
<td><code>ps_ptwrite()</code></td>
<td>Copies <code>size</code> bytes from the text segment of the controlling process to the target process.</td>
</tr>
<tr>
<td><code>ps_lgetregs()</code></td>
<td>Gets the general registers of the LWP.</td>
</tr>
<tr>
<td><code>ps_lsetregs()</code></td>
<td>Sets the general registers of the LWP.</td>
</tr>
<tr>
<td><code>ps_plog()</code></td>
<td>Logs a message.</td>
</tr>
<tr>
<td><code>ps_lrolltoaddr()</code></td>
<td>Rolls the LWP out of a critical section when the process is stopped.</td>
</tr>
<tr>
<td><code>ps_kill()</code></td>
<td>Sends signal to target process.</td>
</tr>
<tr>
<td><code>ps_lgetfpregs()</code></td>
<td>Gets the LWP’s floating point register set.</td>
</tr>
<tr>
<td><code>ps_lsetfpregs()</code></td>
<td>Sets the LWP’s floating point register set.</td>
</tr>
</tbody>
</table>
Threads Library

**proc_service (3T)**

**SPARC Platform Only**
- `ps_lgetxregs()` Gets the extra state registers from a LWP.
- `ps_lsetxregs()` Sets the extra state registers from a LWP.
- `ps_lgetxregsize()` Returns the size of the architecture-dependent extra state registers.

**x86 Platform Only**
- `ps_lgetLDT()` Reads the local descriptor table of a LWP.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

**SEE ALSO**
- `libthread_db(3T), attributes(5)`
- *Linker and Libraries Guide*

modified 4 Mar 1997 SunOS 5.6 3T-1183
NAME       psignal, sys_siglist – system signal messages

SYNOPSIS   /usr/ucb/cc [ flag ... ] file ...
void psignal ( sig, s)
unsigned sig;
char *s;
char *sys_siglist[];

DESCRIPTION psignal() produces a short message on the standard error file describing the indicated signal. First the argument string s 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>.

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.

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, psiginfo – system signal messages

SYNOPSIS
#include <siginfo.h>
void psignal(int sig, const char *s);
void psiginfo(siginfo_t *pinfo, char *s);

DESCRIPTION
psignal() and psiginfo() produce messages on the standard error output describing a signal. sig is a signal that may have been passed as the first argument to a signal handler. pinfo 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 s is printed first, then a colon and a blank, then the message and a newline.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
sigaction(2), gettext(3C), perror(3C), setlocale(3C), attributes(5), 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  ps_lgetregs, ps_lsetregs, ps_lgetfregs, ps_lsetfregs, ps_lgetxregsize, ps_lgetxregs, ps_lsetxregs — routines that access the target process register in libthread_db

SYNOPSIS  #include <proc_service.h>
            ps_err_e ps_lgetregs(const struct ps_prochandle *ph, lwpid_t lid, prgregset_t gregset)
            ps_err_e ps_lsetregs(const struct ps_prochandle *ph, lwpid_t lid, prgregset_t gregset)
            ps_err_e ps_lgetfregs(const struct ps_prochandle *ph, lwpid_t lid, prgregset_t gregset)
            ps_err_e ps_lsetfregs(const struct ps_prochandle *ph, lwpid_t lid, prgregset_t gregset)
            ps_err_e ps_lgetxregsize(const struct ps_prochandle *ph, lwpid_t lid, int *xregsize)
            ps_err_e ps_lgetxregs(const struct ps_prochandle *ph, lwpid_t lid, prxregset_t xregset)
            ps_err_e ps_lsetxregs(const struct ps_prochandle *ph, lwpid_t lid, prxregset_t xregset)

DESCRIPTION  ps_lgetregs, ps_lsetregs, ps_lgetfregs, ps_lsetfregs, ps_lgetxregsize, ps_lgetxregs, ps_lsetxregs read and write register sets from lightweight processes (LWP’s) within the target process identified by ph. ps_lgetregs gets the general registers of the LWP identified by lid, and ps_lsetregs sets them. ps_lgetfregs gets the LWP’s floating point register set, while ps_lsetfregs sets it. ps_getxregsize, ps_getxregs, and ps_setxregs are SPARC-specific. They do not need to be defined by a controlling process on non-SPARC architecture. ps_getxregsize returns in *xregsize the size of the architecture-dependent extra state registers. ps_getxregs gets the extra state registers, and ps_setxregs sets them.

RETURN VALUES  PS_OK The call returned successfully.
PS_NOFPREGS Floating point registers are neither available for this architecture nor for this process.
PS_NOXREGS Extra state registers are not available on this architecture.
PS_ERR The function did not return successfully.

ATTRIBUTES  See attributes(5) for description of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  libthread(3T), libthread_db(3T), proc_service(3T), libthread_db(4), attributes(5)
NAME
ps_pread, ps_ptread, ps_pdwrite, ps_ptwrite – interfaces in libthread_db that target process memory access

SYNOPSIS
#include <proc_service.h>
ps_err_e ps_pread(const struct ps_prochandle *ph, psaddr_t addr, char *buf, int size)
ps_err_e ps_ptread(const struct ps_prochandle *ph, psaddr_t addr, char *buf, int size)
ps_err_e ps_pdwrite(const struct ps_prochandle *ph, psaddr_t addr, char *buf, int size)
ps_err_e ps_ptwrite(const struct ps_prochandle *ph, psaddr_t addr, char *buf, int size)

DESCRIPTION
These routines copy data between the target process’s address space and the controlling process. ps_pread and ps_ptread copy size bytes from address addr in the target process into buf in the controlling process. In ps_pread, addr refers to an address in the target process’s data segment. In ps_ptread, addr refers to an address in the target process’s text segment. On architectures where text and data share an address space, it is permissible for ps_pread and ps_ptread to be identical. ps_pdwrite and ps_ptwrite are just like ps_pread and ps_ptread, respectively, except that the direction of the copy is reversed. Data is copied from the controlling process to the target process.

RETURN VALUES
PS_OK The call returned successfully. size bytes were copied.
PS_BADADDR The address range from addr through addr+size-1, is not part of the target process’s address space.
PS_ERR The function did not return successfully.

ATTRIBUTES
See attributes(5) for description of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO libthread(3T), libthread_db(3T), proc_service(3T), libthread_db(4), attributes(5)
NAME
ps_pglobal_lookup – looks up the symbol in the symbol table of the load object in the target process

SYNOPSIS
#include <proc_service.h>

ps_err_e ps_pglobal_lookup(const struct ps_prochandle *ph, const char *ld_object_name, const char *ld_symbol_name, psaddr_t *ld_symbol_addr);

DESCRIPTION
ps_pglobal_lookup() looks up the symbol sym_name in the symbol table of the load object obj_name in the target process identified by ph. It returns its value in *sym_addr. The symbol must be global.

RETURN VALUES
PS_OK The call completed successfully. *sym_addr contains the value of the specified symbol.
PS_NOSYM The specified symbol was not found.
PS_ERR The function did not return successfully.

ATTRIBUTES
See attributes(5) for description of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
kill(2), libthread(3T), libthread_db(3T), proc_service(3T), libthread_db(4), attributes(5)
NAME
ps_pstop, ps_pcontinue, ps_lstop, ps_lcontinue, ps_lrolltoaddr, ps_kill – process and LWP control in libthread_db

SYNOPSIS
#include <proc_service.h>
ps_err_e ps_pstop(const struct ps_prochandle *ph)
ps_err_e ps_pcontinue(const struct ps_prochandle *ph)
ps_err_e ps_lstop(const struct ps_prochandle *ph, lwpid_t lwpid)
ps_err_e ps_lcontinue(const struct ps_prochandle *ph, lwpid_t lwpid)
ps_err_e ps_lrolltoaddr(const struct ps_prochandle *ph, lwpid_t lwpid, psaddr_t go_addr, psaddr_t stop_addr)
ps_err_e ps_kill(const struct ps_prochandle *ph, int signum)

DESCRIPTION
ps_pstop() stops the target process identified by ph, while ps_pcontinue() allows it to resume.
liththread_db uses ps_pstop() to freeze the target process while it is under inspection. Within the scope of any single call from outside liththread_db to a liththread_db routine, liththread_db will call ps_pstop(), at most once. If it does, it will call ps_pcontinue() within the scope of the same routine.

The controlling process may already have stopped the target process when it calls liththread_db. In that case, it is not obligated to resume the target process when liththread_db calls ps_pcontinue(). In other words, ps_pstop() is mandatory, while ps_pcontinue() is advisory. After ps_pstop(), the target process must be stopped; after ps_pcontinue(), the target process may be running.

ps_lstop() and ps_lcontinue() stop and resume a single lightweight process (LWP) within the target process ph. They are not currently used by liththread_db.

ps_lrolltoaddr() is used to roll an LWP forward out of a critical section when the process is stopped. It is also used to run the liththread_db agent thread on behalf of liththread.

ps_lrolltoaddr() is always called with the target process stopped, that is, there has been a preceding call to ps_pstop(). The specified LWP must be continued at the address go_addr or at its current address if go_addr is NULL. It should then be stopped when its execution reaches stop_addr. This routine does not return until the LWP has stopped at stop_addr.

ps_kill() directs the signal signum to the target process for which the handle is ph.
ps_kill() has the same semantics as kill(2).

RETURN VALUES
PS_OK The call completed successfully. In the case of ps_pstop, the target process is stopped.
PS_BADLID For ps_lstop and ps_lcontinue only: there is no LWP with id lwpid in the target process.
PS_ERR The function did not return successfully.

modified 4 Feb 1997
SunOS 5.6
3T-1189
ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO

`kill(2), libthread(3T), libthread_db(3T), proc_service(3T), libthread_db(4), attributes(5)`
**NAME**

pthread_atfork – register fork handlers

**SYNOPSIS**

```c
#include <sys/types.h>

int pthread_atfork (void (*prepare)(void), void (*parent)(void), void (*child)(void));
```

**DESCRIPTION**

pthread_atfork() declares fork handlers to be called prior to and following fork(), within
the thread that called fork(). The order of calls to pthread_atfork() is important.

Before fork() processing begins, the prepare fork handler is called. The prepare handler is not called if its address is NULL.

The parent fork handler is called after fork() processing finishes in the parent process, and the child fork handler is called after fork() processing finishes in the child process. If the address of parent or child is NULL, then its handler is not called.

The prepare fork handler is called in LIFO (last-in first-out) order, whereas the parent and child fork handlers are called in FIFO (first-in first-out) order. This calling order allows applications to preserve locking order.

**RETURN VALUES**

Upon successful completion, pthread_atfork() returns 0; otherwise, an error number is returned.

**ERRORS**

ENOMEM Insufficient table space exists to record the fork handler addresses.

**EXAMPLES**

All multi-threaded applications that call fork() in a POSIX threads program, or call
fork1() in a Solaris threads program, and which do more than simply call exec() in the
child of the fork, should ensure that the child is protected from deadlock.

The deadlock scenario: since the "fork-one" model results in cloning only the thread that
called fork, it is possible that, at the time of the call, another thread in the parent owns a
lock. In the child, this thread is not cloned, and so no thread will unlock this lock in the
child. Now, if the single thread in the child needs this lock, there is a deadlock.

The problem is more serious with locks in libraries. Since a library writer does not know
if the application that is using the library calls fork() or not, the library has to protect
itself, for complete correctness, from such a deadlock scenario. If the application that
links with this library calls fork() and does not call exec() in the child, and needs a library
lock that may be held by some other thread in the parent which is inside the library at the
time of the fork, then the application deadlocks inside the library. The problem may be
solved by using pthread_atfork().

The following is a brief and simple description of how to make a library safe with respect
to fork1() by using pthread_atfork().

- Identify all the locks used by the library. Let’s say this list is \{L1, ..., Ln\}. Also identify
the locking order for these locks. Let’s say that this order is also L1, ..., Ln.
- Add a call to pthread_atfork(f1, f2, f3) in the library’s .init section. f1, f2, f3 are
defined as follows:

```c
pthread_atfork(f1, f2, f3);
```
f1()
{
    pthread_mutex_lock(L1);
    pthread_mutex_lock(...);
    pthread_mutex_lock(Ln);
    \|-- ordered in lock order
    \}
}

f2()
{
    pthread_mutex_unlock(L1);
    pthread_mutex_unlock(...);
    pthread_mutex_unlock(Ln);
}

f3()
{
    pthread_mutex_unlock(L1);
    pthread_mutex_unlock(...);
    pthread_mutex_unlock(Ln);
}

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
fork(2), atexit(3C), attributes(5), standards(5)

NOTES
Solaris threads do not offer this functionality, although a call to this interface may be used by a Solaris thread program since the two thread APIs are interoperable.
NAME

pthread_attr_init, pthread_attr_destory, pthread_attr_setscope, pthread_attr_getscope,
pthread_attr_setdetachstate, pthread_attr_getdetachstate, pthread_attr_setstacksize,
pthread_attr_getstacksize, pthread_attr_setstackaddr, pthread_attr_getstackaddr,
pthread_attr_setschedparam, pthread_attr_getschedparam, pthread_attr_setschedpolicy,
pthread_attr_getschedpolicy, pthread_attr_setinheritsched, pthread_attr_getinheritsched
− thread creation attributes

SYNOPSIS

#include <pthread.h>

int pthread_attr_init(pthread_attr_t *attr);
int pthread_attr_destroy(pthread_attr_t *attr);
int pthread_attr_setscope(pthread_attr_t *attr, int contentscope);
int pthread_attr_getscope(const pthread_attr_t *attr, int *contentscope);
int pthread_attr_setdetachstate(pthread_attr_t *attr, int detachstate);
int pthread_attr_getdetachstate(const pthread_attr_t *attr, int *detachstate);
int pthread_attr_setstacksize(pthread_attr_t *attr, size_t stacksize);
int pthread_attr_getstacksize(const pthread_attr_t *attr, size_t *stacksize);
int pthread_attr_setstackaddr(pthread_attr_t *attr, void *stackaddr);
int pthread_attr_getstackaddr(const pthread_attr_t *attr, void **stackaddr);
int pthread_attr_setschedparam(pthread_attr_t *attr, const struct sched_param *param);
int pthread_attr_getschedparam(const pthread_attr_t *attr, struct sched_param *param);
int pthread_attr_setschedpolicy(pthread_attr_t *attr, int policy);
int pthread_attr_getschedpolicy(const pthread_attr_t *attr, int *policy);
int pthread_attr_setinheritsched(pthread_attr_t *attr, int inheritsched);
int pthread_attr_getinheritsched(const pthread_attr_t *attr, int *inheritsched);

DESCRIPTION

The pthread approach to setting attributes for threads is to request the initialization of an
attribute object, attr, and pass the initialized attribute object to pthread_create(3T). The
convention in Solaris is to pass these attributes as flags to thr_create(3T).

All attributes in attr are independent of one another and may be singularly modified or
retrieved. attr, itself, is independent of any thread and can be modified or used to create
new threads. However, any change to attr after a thread is created will not affect that

init

The pthread_attr_init() function initializes a thread attributes object (attr) with the
default value for each attribute as follows:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Default</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>contentscope</td>
<td>PTHREAD_SCOPE_PROCESS</td>
<td>resource competition within process</td>
</tr>
<tr>
<td>detachstate</td>
<td>PTHREAD_CREATE_JOINABLE</td>
<td>joinable by other threads</td>
</tr>
<tr>
<td>stacksize</td>
<td>NULL</td>
<td>stack allocated by system</td>
</tr>
<tr>
<td>stackaddr</td>
<td>NULL</td>
<td>1 megabyte</td>
</tr>
<tr>
<td>priority</td>
<td>--</td>
<td>priority of parent (calling) thread</td>
</tr>
<tr>
<td>policy</td>
<td>SCHED_OTHER</td>
<td>determined by system</td>
</tr>
</tbody>
</table>

modified 8 May 1997

SunOS 5.6

3T-1193
Thread attribute objects should be destroyed before an initialized attribute object is re-initialized.

Destroy

**pthread_attr_destroy()** destroys a thread attributes object (attr), which cannot be reused until it is reinitialized.

Resource

The **pthread_attr_setscope()** and **pthread_attr_getscope()** functions set and get the **contentionscope** thread attribute in the attr object. The **contentionscope** value may be set to the following:

- **PTHREAD_SCOPE_SYSTEM**
  - Indicates system scheduling contention scope. This thread is permanently "bound" to an LWP, and is also called a bound thread. This value is equivalent to **THR_BOUND** in Solaris threads (see **thr_create(3T)**).

- **PTHREAD_SCOPE_PROCESS**
  - Indicates process scheduling contention scope. This thread is not "bound" to an LWP, and is also called an unbound thread. **PTHREAD_SCOPE_PROCESS**, or unbound, is the default.

Detachstate

The **pthread_attr_setdetachstate()** and **pthread_attr_getdetachstate()** functions set and get the **detachstate** attribute in the attr object. The **detachstate** attribute determines whether the thread is created in a detached state or not. The **detachstate** may be set to the following values:

- **PTHREAD_CREATE_DETACHED**
  - Creates a new detached thread. A detached thread disappears without leaving a trace. The thread ID and any of its resources are freed and ready for reuse. **pthread_join(3T)** and **thr_join(3T)** cannot wait for a detached thread.

- **PTHREAD_CREATE_JOINABLE**
  - Creates a new non-detached thread. The thread ID and its user-defined stack, if specified at thread creation time, is not freed until **pthread_join(3T)** or **thr_join(3T)** are called. **pthread_join(3T)** or **thr_join(3T)** must be called to release any resources associated with the terminated thread.

Stacksize and Stackaddr

The **pthread_attr_setstacksize()** and **pthread_attr_getstacksize()** functions set and get the **stacksize** thread attribute in the attr object. The **stacksize** default argument is NULL, and a thread default stack size is 1 megabyte.

The **pthread_attr_setstackaddr()** and **pthread_attr_getstackaddr()** functions set and get the **stackaddr** thread attribute in the attr object. The **stackaddr** default is NULL. (See **pthread_create(3T)**.)
The `pthread_attr_setschedparam()` and `pthread_attr_getschedparam()` functions set and get the scheduling parameter thread attributes in the `attr` argument, determined by the scheduling policy set in the `param` object. The only required member of the `param` structure for the SCHED_OTHER, SCHED_FIFO, and SCHED_RR policies is `sched_priority` (see NOTES section below). You can use these functions to get and set the priority of the thread to be created. The `sched_priority` of the `param` structure is NULL, by default, which means the newly created thread inherits the priority of its parent thread.

Values for the `policy` attribute are SCHED_FIFO, SCHED_RR, or the default value SCHED_OTHER (see NOTES section below).

Upon successful completion, the following functions return 0; otherwise, an error number is returned to indicate the error:

- `pthread_attr_init()`
- `pthread_attr_destroy()`
- `pthread_attr_setstacksize()`
- `pthread_attr_getstacksize()`
- `pthread_attr_setstackaddr()`
- `pthread_attr_getstackaddr()`
- `pthread_attr_setdetachstate()`
- `pthread_attr_getdetachstate()`
- `pthread_attr_setscope()`
- `pthread_attr_getscope()`
- `pthread_attr_setinheritsched()`
- `pthread_attr_getinheritsched()`
- `pthread_attr_setschedpolicy()`
- `pthread_attr_getschedpolicy()`

If any of the following conditions occur, `pthread_attr_init()` returns the corresponding error number:

- **ENOMEM** Insufficient memory exists to create the thread attributes object.

If any of the following conditions occur, `pthread_attr_setstacksize()` returns the corresponding error number:

- **EINVAL** The value of `stacksize` is less than PTHREAD_STACK_MIN or exceeds a system-imposed limit.

If any of the following conditions occur, `pthread_attr_destroy()`, `pthread_attr_setstacksize()`, `pthread_attr_getstacksize()`, `pthread_attr_setstackaddr()`, `pthread_attr_getstackaddr()`, `pthread_attr_setdetachstate()`, `pthread_attr_getdetachstate()`, `pthread_attr_setscope()`, `pthread_attr_getscope()`, `pthread_attr_setschedparam()`, `pthread_attr_getschedparam()`, `pthread_attr_setinheritsched()`, `pthread_attr_getinheritsched()`, `pthread_attr_setschedpolicy()`, and `pthread_attr_getschedpolicy()` return the corresponding error number:

- **EINVAL** The value of `attr` is not valid.

If any of the following conditions occur, `pthread_attr_setstacksize()` returns the corresponding error number:

- **EINVAL** The value of `stacksize` is less than PTHREAD_STACK_MIN.

If any of the following conditions occur, `pthread_attr_setdetachstate()` returns the corresponding error number:

- **EINVAL** The value of `detachstate` is not valid.
If any of the following conditions occur, `pthread_attr_setscope()` returns the corresponding error number:

**EINVAL** The value of `contentionscope` is not valid.

If any of the following conditions occur, `pthread_attr_setschedparam()` returns the corresponding error number:

**EINVAL** The value of the `sched_priority` member of the `param` structure is less than or equal to 0.

If any of the following conditions occur, `pthread_attr_getstacksize()` returns the corresponding error number:

**EINVAL** The value of `stacksize` is NULL.

If any of the following conditions occur, `pthread_attr_getstackaddr()` returns the corresponding error number:

**EINVAL** The value of `stackaddr` is NULL.

If any of the following conditions occur, `pthread_attr_getdetachstate()` returns the corresponding error number:

**EINVAL** The value of `detachstate` is NULL.

If any of the following conditions occur, `pthread_attr_getscope()` returns the corresponding error number:

**EINVAL** The value of `contentionscope` is NULL.

If any of the following conditions occur, either `pthread_attr_setschedparam()` and `pthread_attr_getschedparam()` returns the corresponding error number:

**EINVAL** The value of `param` is NULL.

For each of the following conditions, if the condition is detected, `pthread_attr_setinheritsched()` and `pthread_attr_setschedpolicy()` return the corresponding error number:

**ENOTSUP** An attempt was made to set the attribute to an unsupported `policy` or `inheritsched`.

For each of the following conditions, if the condition is detected, `pthread_attr_getinheritsched()` and `pthread_attr_getschedpolicy()` return the corresponding error number:

**EINVAL** `policy` or `inheritsched` is NULL.

### ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### SEE ALSO

`pthread_create(3T)`, `pthread_join(3T)`, `thr_create(3T)`, attributes(5), standards(5)
NOTES

Currently, the only policy supported is \texttt{SCHED\_OTHER}. Attempting to set policy as \texttt{SCHED\_FIFO} or \texttt{SCHED\_RR} will result in the error \texttt{ENOSUP}.

The attribute object is part of the POSIX threads interface. There is no Solaris threads counterpart to the POSIX threads attribute object.

modified 8 May 1997

SunOS 5.6

3T-1197
NAME
pthread_cancel – cancel execution of a thread

SYNOPSIS

#include <pthread.h>

int pthread_cancel(pthread_t target_thread);

DESCRIPTION

pthread_cancel() requests that target_thread be canceled. By default, cancellation is deferred until target_thread reaches a cancellation point (see cancellation(3T) for the definition of a cancellation point). Cancellation cleanup handlers for target_thread are called when the cancellation is acted on. Upon return of the last cancellation cleanup handler, the thread-specific data destructor functions are called for target_thread. target_thread is terminated when the last destructor function returns.

RETURN VALUES

When successful, pthread_cancel() returns 0; otherwise, an error number is returned.

ERRORS

For the following condition, pthread_cancel() returns the corresponding error when the condition occurs:

ESRCH No thread was found with an ID corresponding to that of the specified target_thread.

ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO

cancellation(3T), condition(3T), pthread_cleanup_pop(3T), pthread_cleanup_push(3T), pthread_exit(3T), pthread_join(3T), pthread_setcancelstate(3T), pthread_setcanceltype(3T), pthread_testcancel(3T), setjmp(3C), attributes(5)

NOTES

See cancellation(3T) for a discussion of cancellation concepts.
NAME 

pthread_cleanup_pop – pop a thread cancellation cleanup handler

SYNOPSIS

```
#include <pthread.h>

void pthread_cleanup_pop(int execute);
```

MT-LEVEL

MT-Safe

DESCRIPTION

pthread_cleanup_pop() removes the cleanup handler routine at the top of the cancellation cleanup stack of the calling thread and executes it if execute is non-zero.

When a thread exits or is canceled and its cancellation cleanup stack is not empty, the cleanup handlers are invoked with the pthread_cleanup_push(3T) argument arg in LIFO order from the cancellation cleanup stack.

When the thread calls pthread_cleanup_pop() with a non-zero execute argument, the argument at the top of the stack is popped and executed. An argument of zero pops the handler without executing it.

The Solaris system generates a compile time error if pthread_cleanup_push() does not have a matching pthread-cleanup_pop().

Be aware that using longjmp() or siglongjmp() to jump into or out of a push/pop pair can lead to trouble, as either the matching push or the matching pop statement might not get executed.

RETURN VALUES

pthread_cleanup_pop() is a statement and does not return anything.

ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO

cancellation(3T), condition(3T), pthread_cancel(3T), pthread_cleanup_push(3T), pthread_exit(3T), pthread_join(3T), pthread_setcancelstate(3T), pthread_setcanceltype(3T), pthread_testcancel(3T), setjmp(3C), attributes(5)

NOTES

See cancellation(3T) for a discussion of cancellation concepts.
NAME
pthread_cleanup_push – push a thread cancellation cleanup handler

SYNOPSIS
#include <pthread.h>

void pthread_cleanup_push(void (∗handler)(void ∗arg));

MT-LEVEL
MT-Safe

DESCRIPTION
pthread_cleanup_push(3T) pushes the specified cancellation cleanup handler routine, handler, onto the cancellation cleanup stack of the calling thread.

When a thread exits or is canceled and its cancellation cleanup stack is not empty, the cleanup handlers are invoked with the argument arg in LIFO order from the cancellation cleanup stack.

When the thread calls pthread_cleanup_pop(3T) with a non-zero execute argument, the argument at the top of the stack is popped and executed. When the thread calls pthread_cleanup_pop(3T) with a zero execute argument, the argument at the top of the stack is popped but not executed.

The Solaris system generates a compile time error if pthread_cleanup_push(3T) does not have a matching pthread_cleanup_pop(3T).

Be aware that using longjmp(3C) or siglongjmp() to jump into or out of a push/pop pair can lead to trouble, as either the matching push or the matching pop statement might not get executed.

RETURN VALUES
pthread_cleanup_push(3T) is a statement and does not return anything.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
cancellation(3T), condition(3T), longjmp(3C), pthread_cancel(3T),
pthread_cleanup_pop(3T), pthread_exit(3T), pthread_join(3T),
pthread_setcancelstate(3T), pthread_setcanceltype(3T), pthread_testcancel(3T), attributes(5)

NOTES
See the cancellation(3T) page for a discussion of cancellation concepts.
NAME

 pthread_condattr_init, pthread_condattr_setpshared, pthread_condattr_getpshared,
 pthread_condattr_destroy – condition variable initialization attributes

SYNOPSIS

#include <pthread.h>

int pthread_condattr_init(pthread_condattr_t *attr);
int pthread_condattr_setpshared(pthread_condattr_t *attr, int process-shared);
int pthread_condattr_getpshared(const pthread_condattr_t *attr, int *process-shared);
int pthread_condattr_destroy(pthread_condattr_t *attr);

DESCRIPTION

Initialize

The function pthread_condattr_init() initializes a condition variable attributes object attr
with the default value for all the attributes.

At present, the only attribute available is the scope of condition variables, specified by
process-shared.

The default value of the process-shared attribute is PTHREAD_PROCESS_PRIVATE, which
only allows the condition variable to be operated upon by threads created within the
same process as the thread that initialized the condition variable. If threads from other
processes try to operate on this condition variable, the behavior is undefined.

The process-shared attribute may be set to PTHREAD_PROCESS_SHARED, which allows a
condition variable to be operated upon by any thread with access to the memory allo-
cated to the condition variable, even if the condition variable is allocated in memory that
is shared by multiple processes.

Attempts to initialize previously initialized condition variable attributes object will leave
the storage allocated by the previous initialization unallocated.

Once a condition variable attributes object initializes one or more condition variables, any
function affecting the attributes object (including destruction) will not effect any previ-
ously initialized condition variables.

Set/Get Scope

pthread_condattr_setpshared() sets the process-shared attribute in an initialized attributes
object referenced by attr. pthread_condattr_getpshared() obtains the value of the
process-shared attribute from the attributes object referenced by attr.

Destroy

pthread_condattr_destroy() destroys a condition variable attributes object; the object
becomes uninitialized. A destroyed condition variable attributes object can be reinitia-
lized with pthread_condattr_init(); however, the results of referencing the object after it
has been destroyed are undefined.

RETURN VALUES

pthread_condattr_init(), pthread_condattr_destroy(), and
pthread_condattr_setpshared() return 0 upon a successful return; otherwise, an error
number is returned.

pthread_condattr_getpshared() returns 0 upon a successful return, and stores the value
of the process-shared attribute of attr in the object referenced by the process-shared pa-
rameter; otherwise, an error number is returned.

modified 30 Dec 1996

SunOS 5.6

3T-1201
ERRORS

pthread_condattr_init() returns an error number if any of the following conditions are detected:

ENOMEM Insufficient memory exists to initialize the condition variable attributes object.

pthread_condattr_destroy(), pthread_condattr_getpshared(), and pthread_condattr_setpshared() return an error number if the following condition is detected:

EINVAL The value specified by attr is invalid.

EINVAL The new value specified for the attribute is outside the range of legal values for that attribute.

ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO

cond_init(3T), pthread_create(3T), pthread_cond_init(3T), pthread_mutex_init(3T), attributes(5)
Thread creation adds a new thread of control to the current process. The procedure `main()` itself, is a single thread of control. Each thread executes simultaneously with all the other threads within the calling process, and with other threads from other active processes.

A newly created thread shares all of the calling process' global data with the other threads in this process; however, it has its own set of attributes and private execution stack. The new thread inherits the calling thread's signal mask, possibly, and scheduling priority. Pending signals for a new thread are not inherited and will be empty.

The call to create a thread takes the address of a user-defined function, specified by `start_func`, as one of its arguments, which is the complete execution routine for the new thread.

The lifetime of a thread begins with the successful return from `pthread_create()` or `thr_create()`, which calls `start_func()` and ends with either:

- the normal completion of `start_func()`,
- the return from an explicit call to `pthread_exit(3T)` or `thr_exit(3T)`,
- a thread cancellation (see `pthread_cancel(3T)`). or
- the conclusion of the calling process (see `exit(2)`).

The new thread performs by calling the function defined by `start_func` with one argument, `arg`. If more than one argument needs to be passed to `start_func`, the arguments can be packed into a structure, and the address of that structure can be passed to `arg`.

If `start_func` returns, the thread will terminate with the exit status set to the `start_func` return value (see `pthread_exit(3T)` or `thr_exit(3T)`).

Note that when the thread returns in which `main()` originated from, the effect is the same as if there were an implicit call to `exit()` using the return value of `main()` as the exit status. This differs from a `start_func` return. However, if `main()` itself calls either `pthread_exit(3T)` or `thr_exit(3T)`, only the `main` thread exits, not the entire process.
If the thread creation itself fails, a new thread is not created and the contents of the location referenced by the pointer to the new thread are undefined.

### Attributes

The configuration of a set of attributes defines the behavior of a thread. At creation, each attribute of a new thread may be user-defined or set to the default. All attributes are defined upon thread creation, however, some may be dynamically modified after creation. Establishing these attributes varies depending upon whether POSIX or Solaris threads are used. Both implementations offer a few attributes the other does not.

The available attributes are:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>API</th>
</tr>
</thead>
<tbody>
<tr>
<td>contentionscope</td>
<td>Scheduled by threads library (local scope) or scheduled by the OS (global scope)</td>
<td>both</td>
</tr>
<tr>
<td>detachstate</td>
<td>Allows other threads to wait for a particular thread to terminate</td>
<td>both</td>
</tr>
<tr>
<td>stackaddr</td>
<td>Sets a pointer to the thread’s stack</td>
<td>both</td>
</tr>
<tr>
<td>stacksizex</td>
<td>Sets the size of the thread’s stack</td>
<td>both</td>
</tr>
<tr>
<td>concurrency</td>
<td>Elevates concurrency, if possible</td>
<td>Solaris</td>
</tr>
<tr>
<td>priority</td>
<td>Sets ranking within the policy (scheduling class)</td>
<td>both</td>
</tr>
<tr>
<td>policy</td>
<td>Sets scheduling class; SCHED_OTHER</td>
<td>POSIX</td>
</tr>
<tr>
<td>inheritsched</td>
<td>Determines whether scheduling parameters are inherited or explicitly defined</td>
<td>POSIX</td>
</tr>
<tr>
<td>suspended</td>
<td>Sets thread to runnable vs. suspended</td>
<td>Solaris</td>
</tr>
<tr>
<td>daemon</td>
<td>Defines a thread’s behavior to be like a daemon</td>
<td>Solaris</td>
</tr>
</tbody>
</table>

POSIX

`pthread_create()` creates a new thread within a process with attributes defined by `attr`. Default attributes are used if `attr` is `NULL`. If any attributes specified by `attr` are changed in the attribute object prior to the call to `pthread_create()`, the new thread will acquire those changes. However, if any attributes specified by `attr` are changed after the call to `pthread_create()`, the attributes of existing threads will not be affected. Since `pthread_create()` can use an attribute object in its call, a user-defined thread creation must be preceded by a user-defined attribute object (see `pthread_attr_init(3T)`). Upon successful completion, and if the return value is not `NULL`, `pthread_create()` will store the ID of the created thread in the location referenced by `new_thread_ID`.

It is recommended that for POSIX thread creation, all attribute objects, `attrs`, which will be used later during creation calls, be initialized and modified in the early stages of program execution.

The default creation attributes for `pthread_create()` are:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Default Value</th>
<th>Meaning of Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>contentionscope</td>
<td>PTHREAD_SCOPE_PROCESS</td>
<td>Resource competition within process</td>
</tr>
<tr>
<td>detachstate</td>
<td>PTHREAD_CREATE_JOINABLE</td>
<td>Joinable by other threads</td>
</tr>
<tr>
<td>stackaddr</td>
<td>NULL</td>
<td>Allocated by system</td>
</tr>
<tr>
<td>stacksizex</td>
<td>NULL</td>
<td>1 megabyte</td>
</tr>
<tr>
<td>priority</td>
<td>NULL</td>
<td>Parent (calling) thread’s priority</td>
</tr>
<tr>
<td>policy</td>
<td>SCHED_OTHER</td>
<td>Determined by system</td>
</tr>
</tbody>
</table>
Default thread creation:

```c
pthread_t tid;
void *start_func(void *), *arg;

pthread_create(&tid, NULL, start_func, arg);
```

This would have the same effect as:

```c
pthread_attr_t attr;

pthread_attr_init(&attr); /* initialize attr with default attributes */
 pthread_create(&tid, &attr, start_func, arg);
```

User-defined thread creation:

To create a thread that is scheduled on a system-wide basis (i.e., a bound thread, as per the Solaris API), use:

```c
pthread_attr_init(&attr); /* initialize attr with default attributes */
 pthread_attr_setscope(&attr, PTHREAD_SCOPE_SYSTEM); /* system-wide contention */
 pthread_create(&tid, &attr, start_func, arg);
```

To customize the attributes for POSIX threads, see `pthread_attr_init(3T)`.

A new thread created with `pthread_create()` uses the stack specified by the `stackaddr` attribute, and the stack continues for the number of bytes specified by the `stacksize` attribute. By default, the stack size is 1 megabyte (see `pthread_attr_setstacksize(3T)`). If the default is used for both the `stackaddr` and `stacksize` attributes, `pthread_create()` creates a stack for the new thread with at least 1 megabyte. (For customizing stack sizes, see NOTES).

**Solaris**

In the Solaris API, `thr_create()` either results in the creation of a default thread or a thread whose attributes are defined by the flags passed to `thr_create()`. There is no attribute object to configure, as there is in POSIX. The attributes are either the separate arguments, `stackaddr` or `stacksize`, or the result of bitwise inclusive OR-ing the possible values for `flags`.

The creation attributes for `thr_create()` are:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Default Value</th>
<th>Meaning of Default Value</th>
<th>Specified Via</th>
</tr>
</thead>
<tbody>
<tr>
<td>contentionscope</td>
<td>NULL</td>
<td>Resource competition within process</td>
<td>flags</td>
</tr>
<tr>
<td>detachstate</td>
<td>NULL</td>
<td>Joinable by other threads</td>
<td>flags</td>
</tr>
<tr>
<td>stackaddr</td>
<td>NULL</td>
<td>Allocated by system</td>
<td>separate argument</td>
</tr>
<tr>
<td>stacksize</td>
<td>NULL</td>
<td>1 megabyte</td>
<td>separate argument</td>
</tr>
<tr>
<td>priority</td>
<td>NULL</td>
<td>Parent (calling) thread’s priority</td>
<td>flags</td>
</tr>
<tr>
<td>concurrency</td>
<td>NULL</td>
<td>Determined by system</td>
<td>flags</td>
</tr>
</tbody>
</table>

modified 8 May 1997

SunOS 5.6

3T-1205
flags specifies which attributes are modifiable for the created thread. The value in flags is determined by the bitwise inclusive OR of the following:

- **THR_BOUND**: This flag affects the contentionscope attribute of the thread. The new thread is created permanently bound to an LWP (i.e., it is a bound thread). This thread will now contend among system-wide resources. The bind flag is equivalent to setting the contentionscope to the `PTHREAD_SCOPE_SYSTEM` in POSIX.

- **THR_DETACHED**: This flag affects the detachstate attribute of the thread. The new thread is created detached. The exit status of a detached thread is not accessible to other threads. Its thread ID and other resources may be re-used as soon as the thread terminates. `thr_join(3T)` (not `pthread_join(3T)`) will not wait for a detached thread.

- **THR_NEW_LWP**: This flag affects the concurrency attribute of the thread. The desired concurrency level for unbound threads is increased by one. This is similar to incrementing concurrency by one via `thr_setconcurrency(3T)`. Typically, this adds a new LWP to the pool of LWPs running unbound threads.

- **THR_SUSPENDED**: This flag affects the suspended attribute of the thread. The new thread is created suspended and will not execute `start_func` until it is started by `thr_continue()`.

- **THR_DAEMON**: This flag affects the daemon attribute of the thread. The thread is marked as a daemon. The process will exit when all non-daemon threads exit. `thr_join(3T)` will not wait for a daemon thread. Daemon threads do not interfere with the exit conditions for a process. A process will terminate when all regular threads exit or the process calls `exit()`. Daemon threads are most useful in libraries that want to use threads.

Default thread creation:

```c
thread_t tid;
void *start_func(void *), *arg;
thr_create(NULL, NULL, start_func, arg, NULL, &tid);
```

User-defined thread creation:

To create a thread scheduled on a system-wide basis (i.e., a bound thread), use:

```c
thr_create(NULL, NULL, start_func, arg, THR_BOUND, &tid);
```

Another example of customization is, if both **THR_BOUND** and **THR_NEW_LWP** are specified then, typically, two LWPs are created, one for the bound thread and another for the pool of LWPs running unbound threads.
thr_create(NULL, NULL, start_func, arg, THR_BOUND | THR_NEW_LWP, &tid);

With `thr_create()`, the new thread will use the stack starting at the address specified by `stack_base` and continuing for `stack_size` bytes. `stack_size` must be greater than the value returned by `thr_min_stack(3T)`. If `stack_base` is NULL then `thr_create()` allocates a stack for the new thread with at least `stack_size` bytes. If `stack_size` is zero then a default size is used. If `stack_size` is not zero then it must be greater than the value returned by `thr_min_stack(3T)` (see NOTES).

When `new_thread_ID` is not NULL then it points to a location where the ID of the new thread is stored if `thr_create()` is successful. The ID is only valid within the calling process.

**RETURN VALUES**

Zero indicates a successful return and a non-zero value indicates an error.

**ERRORS**

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

- **EAGAIN**
  
  The system-imposed limit on the total number of threads in a process has been exceeded or some system resource has been exceeded (e.g., too many LWPs were created).

- **EINVAL**
  
  The value specified by `attr` is invalid.

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

- **ENOMEM**
  
  Not enough memory was available to create the new thread.

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

- **EINVAL**
  
  * `stack_base` is not NULL and `stack_size` is less than the value returned by `thr_min_stack(3T)`.
  
  * `stack_base` is NULL and `stack_size` is not zero and is less than the value returned by `thr_min_stack(3T)`.

**EXAMPLES**

This is an example of concurrency with multi-threading. Since POSIX threads and Solaris threads are fully compatible even within the same process, this example uses `pthread_create()` if you execute `a.out 0`, or `thr_create()` if you execute `a.out 1`.

Five threads are created that simultaneously perform a time-consuming function, `sleep(10)`. If the execution of this process is timed, the results will show that all five individual calls to sleep for ten-seconds completed in about ten seconds, even on a uniprocessor. If a single-threaded process calls `sleep(10)` five times, the execution time will be about 50-seconds.
The command-line to time this process is:
```
/usr/bin/time a.out 0 (for POSIX threading)
or
/usr/bin/time a.out 1 (for Solaris threading)
```

```c
/* cc thisfile.c -lthread -lpthread */
#define _REENTRANT /* basic 3-lines for threads */
#include <pthread.h>
#include <thread.h>

#define NUM_THREADS 5
#define SLEEP_TIME 10

void *sleeping(void *); /* thread routine */
void test_argv(); /* optional */
int i;
thread_t tid[NUM_THREADS]; /* array of thread IDs */

main( int argc, char *argv[] ) {
    test_argv(argv[1]);

    switch (*argv[1]) {
    case '0': /* POSIX */
        for ( i = 0; i < NUM_THREADS; i++)
            pthread_create(&tid[i], NULL, sleeping, SLEEP_TIME);
        for ( i = 0; i < NUM_THREADS; i++)
            pthread_join(tid[i], NULL);
        break;
    case '1': /* Solaris */
        for ( i = 0; i < NUM_THREADS; i++)
            thr_create(NULL,0,sleeping,NULL,NULL,&tid[i]);
        while (thr_join(NULL, NULL, NULL) == 0);
        break;
    } /* switch */

    printf("main() reporting that all %d threads have terminated\n", i);
} /* main */

void *sleeping(int sleep_time) { 
    printf("thread %d sleeping %d seconds ...
", thr_self(), SLEEP_TIME);
    sleep(sleep_time);
    printf("\nthread %d awakening\n", thr_self());
} 
```
void test_argv(char argv1[]) {
    /* optional */
    if (argv1 == NULL) {
        printf("use 0 as arg1 to use thr_create(); \n or use 1 as arg1 for use pthread_create() \n")
        exit(NULL);
    }
}

If main() had not waited for the completion of the other threads (using pthread_join(3T) or thr_join(3T)), it would have continued to process concurrently until it reached the end of its routine and the entire process would have exited prematurely (see exit(2)).

The following example shows how to create a default thread with a new signal mask.

new_mask is assumed to have a different value than the creator’s signal mask (orig_mask). new_mask is set to block all signals except for SIGINT. The creator’s signal mask is changed so that the new thread inherits a different mask, and is restored to its original value after thr_create() returns.

This example assumes that SIGINT is also unmasked in the creator. If it is masked by the creator, then unmasking the signal opens the creator up to this signal. The other alternative is to have the new thread set its own signal mask in its start routine.

    thread_t tid;
    sigset_t new_mask, orig_mask;
    int error;

    (void)sigfillset(&new_mask);
    (void)sigdelset(&new_mask, SIGINT);
    (void)thr_sigsetmask(SIG_SETMASK, &new_mask, &orig_mask):
    error = thr_create(NULL, 0, do_func, NULL, 0, &tid);
    (void)thr_sigsetmask(SIG_SETMASK, &orig_mask, NULL);

ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO

_lwp_create(2), exit(2), exit(3C), pthread_attr_init(3T), pthread_cancel(3T),
pthread_exit(3T), pthread_join(3T), sleep(3C), thr_min_stack(3T),
thr_setconcurrency(3T), thr_suspend(3T), threads(3T), attributes(5), standards(5)

NOTES

MT application threads execute independently of each other, thus their relative behavior is unpredictable. Therefore, it is possible for the thread executing main() to finish before all other user application threads.

Using thr_join(3T) in the following syntax,
while (thr_join(NULL, NULL, NULL) == 0);
will cause the invoking thread (which may be main()) to wait for the termination of all other undetached and non-daemon threads; however, the second and third arguments to thr_join(3T) need not necessarily be NULL.

pthread_join(3T), on the other hand, must specify the terminating thread (IDs) for which it will wait.

A thread has not terminated until thr_exit() has finished. The only way to determine this is by thr_join(). When thr_join() returns a departed thread, it means that this thread has terminated and its resources are reclaimable. For instance, if a user specified a stack to thr_create(), this stack can only be reclaimed after thr_join() has reported this thread as a departed thread. It is not possible to determine when a detached thread has terminated. A detached thread disappears without leaving a trace.

Typically, thread stacks allocated by thr_create() begin on page boundaries and any specified (a red-zone) size is rounded up to the next page boundary. A page with no access permission is appended to the top of the stack so that most stack overflows will result in a SIGSEGV signal being sent to the offending thread. Thread stacks allocated by the caller are used as is.

Using a default stack size for the new thread, instead of passing a user-specified stack size, results in much better thr_create() performance. The default stack size for a user-thread is 1 megabyte, in this implementation.

A user-specified stack size must be greater than the value THR_MIN_STACK or PTHREAD_STACK_MIN. A minimum stack size may not accommodate the stack frame for the user thread function start_func. If a stack size is specified, it must accommodate start_func requirements and the functions that it may call in turn, in addition to the minimum requirement.

It is usually very difficult to determine the runtime stack requirements for a thread. THR_MIN_STACK or PTHREAD_STACK_MIN specifies how much stack storage is required to execute a NULL start_func. The total runtime requirements for stack storage are dependent on the storage required to do runtime linking, the amount of storage required by library runtimes (like printf()) that your thread calls. Since these storage parameters are not known before the program runs, it is best to use default stacks. If you know your runtime requirements or decide to use stacks that are larger than the default, then it makes sense to specify your own stacks.
**NAME**
pthread_detach – dynamically detaching a thread

**SYNOPSIS**
POSIX
cc [ flag ... ] file ... -lpthread [ library ... ]
#include <pthread.h>
int pthread_detach(pthread_t threadID);

**DESCRIPTION**
 pthread_detach() can dynamically reset the detachstate attribute of a thread to
PTHREAD_CREATE_DETACHED. For example, a thread could detach itself as follows:
 pthread_detach(pthread_self());

**RETURN VALUES**
Upon successful completion, 0 is returned; otherwise, a non-zero value indicates an error.

**ERRORS**
These functions fail and return the corresponding value, if any of the following conditions are detected:
EINVAL The value specified by threadID is not a joinable thread.
ESRCH The value specified by threadID is not an existing thread ID.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**SEE ALSO**
pthread_create(3T), pthread_join(3T), attributes(5), standards(5)

modified 8 May 1997

SunOS 5.6

3T-1211
NAME  pthread_equal – compare thread IDs

SYNOPSIS  #include <pthread.h>
            int pthread_equal(pthread_t t1, pthread_t t2);

DESCRIPTION  The pthread_equal() function compares the thread IDs t1 and t2.

RETURN VALUES  If t1 and t2 are equal, pthread_equal() returns a non-zero value; otherwise, 0 is returned.
                If either t1 or t2 is an invalid thread ID, the result is unpredictable.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  pthread_create(3T), pthread_self(3T), attributes(5)

NOTES  Solaris thread IDs do not require an equal function because the thread_t structure is really an unsigned int.
**NAME**  
(pthread_exit, thr_exit) – thread termination

**SYNOPSIS**  

POSIX

```
cc [ flag ... ] file ... -lpthread [ library ... ]
#include <pthread.h>
void pthread_exit(void *status);
```

Solaris

```
cc [ flag ... ] file ... -lthread [ library ... ]
#include <thread.h>
void thr_exit(void *status);
```

**DESCRIPTION**  

 pthread_exit() and thr_exit() terminates the calling threads, similar to how exit(3C) terminates calling processes. If the calling thread is not detached, then the thread’s ID and the exit status specified by status are retained. The value status is then made available to any successful join with the terminating thread (see pthread_join(3T)); otherwise, status is disregarded allowing the thread’s ID to be reclaimed immediately.

Upon thread termination, all thread-specific data bindings are released (see pthread_key_create(3T)), and its cancellation routines are called, but application visible process resources, including, but not limited to, mutexes and file descriptors are not released.

The cleanup handlers are called before the thread-specific data bindings are released (see pthread_cancel(3T)). Any cancellation cleanup handlers that have been pushed and not yet popped will be popped in reverse order of when they were pushed and then executed. If the thread still has any thread-specific data after all cancellation cleanup handlers have been executed, appropriate destructor functions will be called in an unspecified order. If any thread, including the main() thread, calls pthread_exit(), only that thread will exit.

If main() returns or exits (either implicitly or explicitly), or any thread explicitly calls exit(), the entire process will exit.

If any thread (except the main() thread) implicitly or explicitly returns, the result is the same as if the thread called pthread_exit() and it will return the value of status as the exit code.

The process will terminate with an exit status of 0 after the last thread has terminated (including the main() thread). This action is the same as if the application had called exit() with a zero argument at any time.

**RETURN VALUES**  

pthread_exit() or thr_exit() does not return to its caller.

---

modified 8 May 1997  
SunOS 5.6  
3T-1213
ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO

exit(3C), pthread_cancel(3T), pthread_create(3T), pthread_join(3T), pthread_key_create(3T), attributes(5), standards(5)

NOTES

Although only POSIX implements cancellation, cancellation can be used with Solaris threads, due to their interoperability.

Do not call pthread_exit() from a cancellation cleanup handler or destructor function that will be invoked as a result of either an implicit or explicit call to pthread_exit().

status should not reference any variables local to the calling thread.
NAME

pthread_join, thr_join – wait for thread termination

SYNOPSIS

POSIX

cc [ flag ...] file ... -lpthread [ library ... ]
#include <pthread.h>
int pthread_join(pthread_t target_thread, void **status);

Solaris

cc [ flag ...] file ... -lthread [ library ... ]
#include <thread.h>
int thr_join(thread_t target_thread, thread_t *departed, void **status);

DESCRIPTION

The pthread_join() and thr_join() functions suspend processing of the calling thread until the target target_thread completes. target_thread must be a member of the current process and it cannot be a detached or daemon thread (see pthread_create(3T)).

Several threads cannot wait for the same thread to complete; one thread will complete successfully and the others will terminate with an error of ESRCH. pthread_join() or thr_join() will not block processing of the calling thread if the target target_thread has already terminated.

pthread_join() or thr_join() will return successfully when the target target_thread terminates.

POSIX

If a pthread_join() call returns successfully with a non-null status argument, the value passed to pthread_exit(3T) by the terminating thread will be placed in the location referenced by status.

If the pthread_join() calling thread is cancelled, then the target target_thread will remain joinable by pthread_join(). However, the calling thread may set up a cancellation cleanup handler on target_thread prior to the join call, which may detach the target thread by calling pthread_detach(3T). (See pthread_detach(3T) and pthread_cancel(3T).)

pthread_join() does not return the target_thread’s ID, as does the Solaris threads’ function thr_join(), and it does not cause the calling thread to wait for detached threads.

pthread_join() returns ESRCH if the target is detached.

Solaris

If a thr_join() call returns successfully with a non-null status argument, the value passed to thr_exit(3T) by the terminating thread will be placed in the location referenced by status.

If the target target_thread ID is 0, thr_join() waits for any undetached thread in the process to terminate.

If departed is not NULL, it points to a location that is set to the ID of the terminated thread if thr_join() returns successfully.

RETURN VALUES

If successful, both pthread_join() and thr_join() would return 0; otherwise, an error number is returned to indicate the error.

modified 8 May 1997 SunOS 5.6 3T-1215
ERRORS | ESRCH  No undetached thread could be found corresponding to that specified by the given thread ID. If the target target_thread ID is 0, pthread_join() will return with error ESRCH.
EDEADLK A deadlock was detected or the value of target_thread specifies the calling thread. (See NOTES section below.)

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO | wait(2), pthread_create(3T), pthread_exit(3T), pthread_join(3T), attributes(5), standards(5)

NOTES | Using thr_join(3T) in the following syntax,

```
while (thr_join(NULL, NULL, NULL) == 0);
```

will wait for the termination of all other undetached and non-daemon threads; after which, EDEADLK will be returned.

pthread_join(3T), on the other hand, must specify the target_thread ID for whose termination it will wait.

Calling pthread_join() also "detaches" the thread, that is, pthread_join() includes the effect of pthread_detach(). Hence, if a thread were to be cancelled when blocked in pthread_join(), an explicit detach would have to be done in the cancellation cleanup handler. In fact, the routine pthread_detach() exists mainly for this reason.
NAME
pthread_key_create, pthread_setspecific, pthread_getspecific, pthread_key_delete,
thr_keycreate, thr_setspecific, thr_getspecific – thread-specific-data functions

SYNOPSIS
POSIX
c [ flag ...] file ... -lpthread [ library ...]
#include <pthread.h>
int pthread_key_create(pthread_key_t *keyp, void (*destructor)(void *value));
int pthread_setspecific(pthread_key_t key, const void *value);
void *pthread_getspecific(pthread_key_t key);
int pthread_key_delete(pthread_key_t key);

Solaris
c [ flag ...] file ... -lthread [ library ...]
#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);

DESCRIPTION
Create Key
In general, thread key creation allocates a key that locates data specific to each thread in
the process. The key is global to all threads in the process, which allows each thread to
bind a value to the key once the key has been created. The key independently maintains
specific values for each binding thread. pthread_key_create() or thr_keycreate() allo-
cates a global key namespace, pointed to by keyp, that is visible to all threads in the pro-
cess. Each thread is initially bound to a private element of this key, which allows access
to its thread-specific data.

Upon key creation, a new key is assigned the value NULL for all active threads. Addi-
tionally, upon thread creation, all previously created keys in the new thread are assigned
the value NULL.

Optionally, a destructor function, destructor, may be associated with each key. Upon
thread exit, 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. If more than one destructor exists for a thread when it exits, the order of destructor
calls is unspecified.

Set Value
Once a key has been created, each thread may bind a new value to the key using
pthread_setspecific() or thr_setspecific(). The values are unique to the binding thread
and are individually maintained. These values continue for the life of the calling thread.

Proper synchronization of key storage and access must be ensured by the caller. The value
argument to either pthread_setspecific() or thr_setspecific() is generally a pointer to a
block of dynamically allocated memory reserved by the calling thread for its own use.
(see "Examples" section below).

modified 8 May 1997 SunOS 5.6 3T-1217
At thread exit, the destructor function, which is associated at time of creation, is called and it uses the specific key value as its sole argument.

**POSIX Get Value**  
`pthread_getspecific()` returns the current value bound to the designated `key` specified by the calling thread. If the key has no value bound to it, the value NULL is returned. (see "Warnings" section below).

**Solaris Get Value**  
`thr_getspecific()` stores the current value bound to `key` for the calling thread into the location pointed to by `valuep`.

**POSIX Delete Key**  
`pthread_key_delete()` deletes a thread-specific data key formerly created by `pthread_key_create()` or `thr_keycreate()`. At the time `pthread_key_delete()` is called, the thread-specific data values associated with `key` do not have to be NULL. It is the application's responsibility to perform cleanup actions related to the deleted key or associated thread-specific data in any threads. Cleanup can be done either before or after calling `pthread_key_delete()`. `pthread_key_delete()` does not invoke a destructor function. Although `pthread_key_create()`'s or `thr_keycreate()`'s destructor function should clean-up the key's thread-specific-data storage, `pthread_key_delete()` needs to be used to free the storage associated with the `key`.

Solaris threads do not have a similar delete function.

**RETURN VALUES**  
**POSIX/Solaris**  
If successful, `pthread_key_create()`, `pthread_setspecific()`, `pthread_key_delete()`, `thr_keycreate()`, `thr_setspecific()`, or `thr_getspecific()` returns 0; otherwise, an error number is returned to indicate the error. `pthread_getspecific()` does not return any errors.

**ERRORS**  
If the following conditions occur, `pthread_key_create()` or `thr_keycreate()` return the corresponding error number:

- **EAGAIN**  
The system lacked the necessary resources to create another thread-specific data key, or the number of keys exceeds the pre-process limit of PTHREAD_KEYS_MAX.

- **ENOMEM**  
Insufficient memory exists to create the key.

If the following conditions occur, `pthread_key_create()`, `pthread_setspecific()`, `thr_keycreate()`, or `thr_setspecific()` return the corresponding error number:

- **ENOMEM**  
Insufficient memory exists to associate the value with the key.

For each of the following conditions, if the condition is detected, `pthread_setspecific()`, `thr_setspecific()`, or `pthread_key_delete()` return the corresponding error number:

- **EINVAL**  
The `key` value is invalid.

**EXAMPLES**  
In this example, the thread-specific data in this function can be called from more than one thread without special initialization. POSIX threads are used exclusively in this example.
For each argument you pass to the executable of this example, a thread is created and
privately bound to the string-value of that argument.

```c
/* cc thisfile.c -lpthread */

#define _REENTRANT
#include <pthread.h>
void *thread_specific_data(), free();
#define MAX_ARGC 20
pthread_t tid[MAX_ARGC];
int num_threads;

main( int argc, char *argv[] ) {
    int i;
    num_threads = argc - 1;
    for( i = 0; i < num_threads; i++)
        pthread_create(&tid[i], NULL, thread_specific_data, argv[i+1]);
    for( i = 0; i < num_threads; i++)
        pthread_join(tid[i], NULL);
} /* end main */

void *thread_specific_data(char private_data[])
{
    static pthread_mutex_t keylock; /* static ensures only one copy of keylock */
    static pthread_key_t key;
    static int once_per_keyname = 0;
    void *tsd = NULL;
    if (!once_per_keyname) { /* see pthread_once(3T) */
        pthread_mutex_lock(&keylock);
        if (!once_per_keyname++) /* retest with lock */
            pthread_key_create(&key, free);
        pthread_mutex_unlock(&keylock);
    }
    tsd = pthread_getspecific(key);
    if (tsd == NULL) {
        tsd = (void *)malloc(strlen(private_data) + 1);
        strcpy(tsd, private_data);
        pthread_setspecific(key, tsd);
        printf("tsd for %d = %s\n",thr_self(),(char *)pthread_getspecific(key));
        sleep(2);
        printf("tsd for %d remains %s\n",thr_self(),(char *)pthread_getspecific(key));
    }
} /* end thread_specific_data */
```

modified 8 May 1997 SunOS 5.6 3T-1219
void
free(void *v) { 
    /* application-specific clean-up function */
}

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
pthread_exit(3T), attributes(5), standards(5)

WARNINGS
pthread_setspecific(), pthread_getspecific(), thr_setspecific(), and thr_getspecific(),
may be called either explicitly, or implicitly from a thread-specific data destructor func-
tion. However, calling pthread_setspecific() or thr_setspecific() from a destructor may
result in lost storage or infinite loops.
NAME
pthread_kill, thr_kill – send a signal to a thread

SYNOPSIS
POSIX
cc [flag ...] file ... -lpthread [ library ...]
#include <signal.h>
#include <pthread.h>
int pthread_kill(pthread_t thread, int sig);

Solaris
cc [flag ...] file ... -lthread [ library ...]
#include <signal.h>
#include <thread.h>
int thr_kill(thread_t thread, int sig);

DESCRIPTION
pthread_kill() sends the sig signal to the thread designated by thread. thread must be a member of the same process as the calling thread. sig must be one of the signals listed in signal(5); with the exception of SIGLWP, SIGCANCEL, and SIGWAITING being reserved and off limits to thr_kill() or pthread_kill(). If sig is 0, a validity check is done for the existence of the target thread; no signal is sent.

thr_kill() performs the same function as pthread_kill().

RETURN VALUES
Upon successful completion, pthread_kill() and thr_kill() return 0; otherwise, they return an error number. In the event of failure, no signal is sent.

ERRORS
ESRCH No thread was found that corresponded to the thread designated by thread ID.
EINVAL The sig argument value is not zero and is an invalid or an unsupported signal number.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe and Async-Signal-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
kill(2), sigaction(2), pthread_self(3T), pthread_sigmask(3T), raise(3C), attributes(5), signal(5), standards(5)

NOTES
Although pthread_kill() is Async-Signal-Safe with respect to the Solaris environment, this safeness is not guaranteed to be portable to other POSIX domains.

modified 8 May 1997
SunOS 5.6

3T-1221
NAME

pthread_mutexattr_init, pthread_mutexattr_destroy, pthread_mutexattr_setpshared,
pthread_mutexattr_getpshared, pthread_mutexattr_setprotocol,
pthread_mutexattr_getprotocol, pthread_mutexattr_setprioceiling,
pthread_mutexattr_getprioceiling – mutex initialization attributes

SYNOPSIS

#include <pthread.h>

int pthread_mutexattr_init(pthread_mutexattr_t *attr);
int pthread_mutexattr_destroy(pthread_mutexattr_t *attr);
int pthread_mutexattr_setpshared(pthread_mutexattr_t *attr, int process-shared);
int pthread_mutexattr_getpshared(const pthread_mutexattr_t *attr, int *process-shared);

DESCRIPTION

Initialize

pthread_mutexattr_init() initializes a mutex attributes object, attr, with the default value
for its attribute, which is PTHREAD_PROCESS_PRIVATE. If the process-shared attribute is
PTHREAD_PROCESS_PRIVATE, only threads created within the same process as the
thread that initialized the mutex can access the mutex. If threads of differing processes
attempt to access the mutex, the behavior is unpredictable.

Attempts to initialize an already initialized mutex variable attributes object will leave the
storage allocated by the previous initialization unallocated.

Once a mutex attributes object is used to initialize one or more mutexes, any function that
affects the attributes object (including destruction) will not affect any previously initial-
ized mutexes.

Destroy

pthread_mutexattr_destroy() destroys a mutex attributes object; the object will then
become uninitialized. A destroyed mutex attributes object can be reinitialized using
pthread_mutexattr_init(). The results of referencing the object after it has been des-
troyed are undefined.

Set/Get Scope

pthread_mutexattr_setpshared() and pthread_mutexattr_getpshared() sets the process-
shared attribute in an initialized attributes object pointed to by attr, and gets the value of
the process-shared attribute from the attributes object pointed to by attr, respectively.

At present, only the attribute process-shared is defined.

Unsupported Interfaces

Currently, the following interfaces, which are optional under POSIX, are not supported:

int pthread_mutexattr_setprotocol (pthread_mutexattr_t *attr, int protocol);

RETURN VALUES

Upon successful completion, pthread_mutexattr_init(), pthread_mutexattr_destroy(),
pthread_mutexattr_setprotocol(), pthread_mutexattr_getprotocol(),
pthread_mutexattr_setprioceiling(), pthread_mutexattr_getprioceiling(), and
pthread_mutexattr_setpshared() return 0; otherwise, an error number is returned.

Upon successful completion, pthread_mutexattr_getpshared() returns 0 and stores the
value of the process-shared attribute of attr in the object pointed to by the process-shared
parameter; otherwise, an error number is returned.
The function `pthread_mutexattr_init()` returns an error number if the following condition is detected:

**ENOMEM** Insufficient memory exists to initialize the mutex attributes object.

The functions `pthread_mutexattr_destroy()`, `pthread_mutexattr_getpshared()`, and `pthread_mutexattr_setpshared()` return an error number if the following condition is detected:

**EINVAL** The value specified by `attr` is invalid.

The function `pthread_mutexattr_setpshared()` returns an error number if the following condition is detected:

**EINVAL** The new value specified for the attribute is outside the range of legal values for that attribute.

Currently, the functions `pthread_mutexattr_setprotocol()`, `pthread_mutexattr_getprotocol()`, `pthread_mutexattr_setprioceiling()`, and `pthread_mutexattr_getprioceiling()` always return the following error code:

**ENOSYS** These optional interfaces are not supported.

### ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### SEE ALSO

`pthread_cond_init(3T)`, `pthread_create(3T)`, `pthread_mutex_init(3T)`, attributes(5), standards(5)

### NOTES

The functions `pthread_mutexattr_setprotocol()`, `pthread_mutexattr_getprotocol()`, `pthread_mutexattr_setprioceiling()`, and `pthread_mutexattr_getprioceiling()` return ENOSYS in the current implementation, i.e., this function is not currently implemented.
NAME  pthread_mutex_setprioceiling, pthread_mutex_getprioceiling – change the priority ceiling of a mutex

SYNOPSIS  
#include <pthread.h>

int pthread_mutex_setprioceiling(pthread_mutex_t *mutex, int prioceiling, int *old_ceiling);

int pthread_mutex_getprioceiling(const pthread_mutex_t *mutex, int *prioceiling);

DESCRIPTION  
In the current implementation, \texttt{POSIX\_THREAD\_PRIO\_PROTECT} is undefined and the functions \texttt{pthread_mutex_setprioceiling()} and \texttt{pthread_mutex_getprioceiling()} return \texttt{ENOSYS}.

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

\begin{tabular}{|c|c|}
\hline
ATTRIBUTE TYPE & ATTRIBUTE VALUE \\
\hline
MT-Level & MT-Safe \\
\hline
\end{tabular}

SEE ALSO  
\texttt{pthread_mutex_init(3T)}, attributes(5), standards(5)
NAME    pthread_once – dynamic package initialization

SYNOPSIS #include <pthread.h>
            pthread_once_t once_control = PTHREAD_ONCE_INIT;
            int pthread_once(pthread_once_t *once_control, void (*init_routine)(void));

DESCRIPTION If any thread in a process with a once_control parameter makes a call to pthread_once(),
the first call will summon the init_routine(), but subsequent calls will not. The
once_control parameter determines whether the associated initialization routine has been
called. The init_routine() is complete upon return of pthread_once().

pthread_once() is not a cancellation point; however, if the function init_routine() is a
 cancellation point and is canceled, the effect on once_control is the same as if
 pthread_once() had never been called.

The constant PTHREAD_ONCE_INIT is defined in the <pthread.h> header.
If once_control has automatic storage duration or is not initialized by
PTHREAD_ONCE_INIT, the behavior of pthread_once() is undefined.

RETURN VALUES pthread_once() returns 0 upon successful completion; otherwise, an error number is
returned.

ERRORS EINVAL once_control or init_routine is NULL.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO attributes(5)

NOTES Solaris threads do not offer this functionality.
**NAME**
 pthread_self, thr_self – get calling thread’s ID

**SYNOPSIS**

POSIX

```c
#include <pthread.h>
pthread_t pthread_self(void);
typedef unsigned int pthread_t;
```

Solaris

```c
#include <thread.h>
thread_t thr_self(void)
typedef unsigned int thread_t;
```

**DESCRIPTION**

thr_self() returns the thread ID of the calling thread.

pthread_self() performs the same function as thr_self().

**ATTRIBUTES**

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**SEE ALSO**

pthread_create(3T), pthread_equal(3T), attributes(5), standards(5)
NAME
pthread_setcancelstate – enable or disable cancellation

SYNOPSIS
#include <pthread.h>
int pthread_setcancelstate(int state, int *oldstate);

DESCRIPTION
pthread_setcancelstate() atomically sets the calling thread’s cancellation state to the
specified state and, if oldstate is not NULL, stores the previous cancellation state in oldstate.
The state can be either of the following:

PTHREAD_CANCEL_ENABLE
This is the default. When cancellation is deferred (deferred cancellation is also
the default), cancellation occurs when the target thread reaches a cancellation
point and a cancel is pending. When cancellation is asynchronous, receipt of a
pthread_cancel(3T) call causes immediate cancellation.

PTHREAD_CANCEL_DISABLE
When cancellation is deferred, all cancellation requests to the target thread are
held pending. When cancellation is asynchronous, all cancellation requests to the
target thread are held pending; as soon as cancellation is re-enabled, pending
cancellations are executed immediately.

See cancellation(3T) for the definition of a cancellation point. See
pthread_setcanceltype(3T) for explanations of deferred and asynchronous cancellation.

pthread_setcancelstate() is a cancellation point when it is called with
PTHREAD_CANCEL_ENABLE and the cancellation type is
PTHREAD_CANCELASYNCHRONOUS.

RETURN VALUES
When successful, pthread_setcancelstate(), returns 0; otherwise, an error number is
returned.

ERRORS
For the following condition, pthread_setcancelstate() returns the corresponding error
when the condition is detected:

EINVAL
The specified state is not PTHREAD_CANCEL_ENABLE or
PTHREAD_CANCEL_DISABLE.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
cancellation(3T), condition(3T), pthread_cancel(3T), pthread_cleanup_pop(3T),
pthread_cleanup_push(3T), pthread_exit(3T), pthread_join(3T),
pthread_setcanceltype(3T), pthread_testcancel(3T), setjmp(3C), attributes(5)

modified 6 Jan 1997
SunOS 5.6
3T-1227
NOTES

See the cancellation(3T) page for a discussion of cancellation concepts.
NAME (pthread_setcanceltype) — set the cancellation type of a thread

SYNOPSIS

```c
#include <pthread.h>

int pthread_setcanceltype(int type, int *oldtype);
```

DESCRIPTION

pthread_setcanceltype() atomically sets the calling thread’s cancellation type to the specified type and, if oldtype is not NULL, stores the previous cancellation type in oldtype. The type can be either of the following:

**PTHREAD_CANCEL_DEFERRED**

This is the default. When cancellation is enabled (enabled cancellation is also the default), cancellation occurs when the target thread reaches a cancellation point and a cancel is pending. When cancellation is disabled, all cancellation requests to the target thread are held pending.

**PTHREAD_CANCELASYNCHRONOUS**

When cancellation is enabled, receipt of a pthread_cancel(3T) call causes immediate cancellation. When cancellation is disabled, all cancellation requests to the target thread are held pending; as soon as cancellation is re-enabled, pending cancellations are executed immediately.

See cancellation(3T) for the definition of a cancellation point. See pthread_setcancelstate(3T) for explanations of enabling and disabling cancellation.

pthread_setcanceltype() is a cancellation point if type is called with PTHREAD_CANCELASYNCHRONOUS and the cancellation state is PTHREADCANCEL_ENABLE.

RETURN VALUES

When successful, pthread_setcanceltype() returns 0; otherwise, an error number is returned.

ERRORS

For the following condition, pthread_setcanceltype() returns the corresponding error when the condition is detected:

**EINVAL**

The specified type is not PTHREADCANCEL_DEFERRED or PTHREADCANCELASYNCHRONOUS.

ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO

cancellation(3T), condition(3T), pthread_cancel(3T), pthread_cleanup_pop(3T), pthread_cleanup_push(3T), pthread_exit(3T), pthread_join(3T), pthread_setcancelstate(3T), pthread_testcancel(3T), setjmp(3C), attributes(5)
| NOTES | See `cancellation(3T)` for a discussion of cancellation concepts. |
NAME

pthread_setschedparam, pthread_getschedparam, thr_setprio, thr_getprio – dynamic access to thread scheduling

SYNOPSIS

POSIX

cc [ flag ...] file ... -lpthread [ library ...]
#include <pthread.h>

int pthread_setschedparam(pthread_t target_thread, int policy,
const struct sched_param *param);
int pthread_getschedparam(pthread_t target_thread, int *policy,
struct sched_param *param);

Solaris

cc [ flag ...] file ... -lthread [ library ...]
#include <thread.h>

int thr_setprio(thread_t target_thread, int priority);
int thr_getprio(thread_t target_thread, int *priority);

DESCRIPTION

Thread scheduling is controlled by three attributes: its scope of contention, being either inter-process or intra-process (bound vs. unbound), (see priocntl(2)); a relative scheduling priority; and a scheduling policy.

Contentionscope

Bound threads, which are inter-process, compete system-wide for scheduling resources and must be set at creation, for example:

pthread_attr_setscope(&attr, PTHREAD_SCOPE_SYSTEM);
pthread_create(NULL, &attr, thread_routine, arg);

OR

thr_create(NULL, NULL, thread_routine, arg, THR_BOUND, NULL);

A bound thread is bound to an LWP and its scheduling is dependent upon the scheduling of the LWP to which it is bound. LWPs compete with other LWPs in other processes, however, their scheduling may be dynamically controlled by priocntl(2), or sched_setscheduler(3R).

By default, the scope for newly-created threads are unbound, or intra-process, and their setting is PTHREAD_SCOPE_PROCESS or NULL. An unbound thread is scheduled by libthread or libpthread on an underlying LWP, which competes with other LWPs in the same process.

The following dynamic scheduling functions should be used only with unbound threads: pthread_setschedparam(), pthread_getschedparam(), thr_setprio(), and thr_getprio().

Priority

Priority scheduling is determined as follows:

• Higher priority threads are scheduled before lower priority threads.
• Both POSIX and Solaris assume that the priority is inherited across a thread create.
• POSIX can modify priority at creation time (see

modified 8 May 1997

SunOS 5.6

3T-1231
**pthread_setschedparam** and **thr_setprio** can dynamically modify an unbound thread’s priority, and **pthread_getschedparam** and **thr_getprio** can read an unbound thread’s priority.

**Policy**

The scheduling policy setting is:

**SCHED_OTHER** (system default, often time-sharing)

Competing threads in this class are multiplexed according to their relative priority.

POSIX specifies, under an option, the additional policies, **SCHED_FIFO** and **SCHED_RR**. Solaris has chosen to not implement these options at this time. Equivalent functionality may be obtained by creating bound threads (i.e., threads with the PTHREAD_SCOPE_SYSTEM value for the contentionscope attribute), which use **priocntl**. See **pthread_create** and **priocntl**.

**POSIX Scheduling**

The **pthread_setschedparam()** and **pthread_getschedparam()** functions allow the scheduling policy and scheduling priority parameters to be retrieved and set for individual threads within a multi-threaded process.

The **pthread_setschedparam()** function sets the scheduling policy and related scheduling priority for the thread ID given by **target_thread** to the policy and associated priority provided in **policy**, and the **sched_priority** member of **param**, respectively.

No scheduling parameters are changed for the target thread if **pthread_setschedparam()** fails.

For **SCHED_OTHER**, the affected scheduling parameter is the **sched_priority** member of the **sched_param** structure.

Presently, **SCHED_OTHER** is the only policy supported. An **ENOSUP** error will occur following an attempt to set policy as **SCHED_FIFO** or **SCHED_RR**. (The latter two policies are optional under POSIX.)

The **pthread_getschedparam()** function retrieves the scheduling policy and scheduling priority parameters for the thread ID given by **target_thread**, and then stores the values in **policy** and the **sched_priority** member of **param**, respectively.

**Solaris Scheduling**

Solaris scheduling may only dynamically affect priority. There is no functionality to alter the policy of any thread; by default, a Solaris thread’s schedule is equivalent to **SCHED_OTHER**, which is the only available Solaris policy.

**thr_setprio()** changes the priority of the thread, specified by **target_thread**, within the current process to the priority specified by **priority**. Currently, by default, threads are scheduled based on fixed priorities that range from zero, the least significant, to 127. The **target_thread** will preempt lower priority threads, and will yield to higher priority threads in their contention for LWPs, not CPUs.
The function `thr_getprio()` stores the current priority for the thread specified by `target_thread` in the location pointed to by `priority`. Note that thread priorities regulate access to LWPs, not CPUs, and hence are different from real-time priorities, which regulate and enforce access to CPU resources. A thread’s priority set via these functions is more like a hint in terms of guaranteed access to execution resources. Programs that need access to "real" priorities should use bound threads in the real-time class (see `priocntl(2)`).

**RETURN VALUES**  
Zero is returned upon successful completion; otherwise, an error number is returned.

**ERRORS**  
For each of the following conditions, these functions return an error number if the condition is detected.

- **ESRCH**  
The value specified by `target_thread` does not refer to an existing thread.

For each of the following conditions, `pthread_setschedparam()` and `pthread_getschedparam()` return an error number if the condition is detected.

- **ENOSUP**  
The only policy supported is `SCHED_OTHER`. Attempts to set policy as `SCHED_FIFO` or `SCHED_RR` will result in the error **ENOSUP**.

- **EINVAL**  
The `policy` or `param` specified value is invalid.

For each of the following conditions, if the condition is detected, `thr_setprio()` returns an error number.

- **EINVAL**  
The value of `priority` makes no sense for the scheduling class associated with the `target_thread`.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**SEE ALSO**  
`priocntl(2), sched_setparam(3R), sched_setscheduler(3R), pthread_attr_init(3T), pthread_create(3T), thr_suspend(3T), thr_yield(3T), attributes(5), standards(5)`

**NOTES**  
Currently, the only supported policy is `SCHED_OTHER`. Attempts to set policy as `SCHED_FIFO` or `SCHED_RR` will result in the error **ENOSUP**.
NAME	pthread_sigmask, thr_sigsetmask – change and/or examine calling thread’s signal mask

SYNOPSIS

POSIX

cc [ flag . . . ] file . . . -lpthread [ library . . . ]
#include <pthread.h>
#include <signal.h>
int pthread_sigmask(int how, const sigset_t *set, sigset_t *oset);

Solaris

c c [ flag . . . ] file . . . -lt thread [ library . . . ]
#include <thread.h>
#include <signal.h>
int thr_sigsetmask(int how, const sigset_t *set, sigset_t *oset);

DESCRIPTION

pthread_sigmask() and thr_sigsetmask() change and/or examine a calling thread’s signal mask. Each thread has its own signal mask. A new thread inherits the calling thread’s signal mask and priority; however, pending signals are not inherited. Signals pending for a new thread will be empty.

If the value of the argument set is not NULL, set points to a set of signals that can modify the currently blocked set. If the value of set is NULL, the value of how is insignificant and the thread’s signal mask is unmodified; thus, pthread_sigmask() or thr_sigsetmask() can be used to inquire about the currently blocked signals.

The value of the argument how specifies the method in which the set is changed. how takes one of the following values:

SIG_BLOCK set corresponds to a set of signals to block. They are added to the current signal mask.

SIG_UNBLOCK set corresponds to a set of signals to unblock. These signals are deleted from the current signal mask.

SIG_SETMASK set corresponds to the new signal mask. The current signal mask is replaced by set.

If the value of oset is not NULL, it points to the location where the previous signal mask is stored.

RETURN VALUES

0 Successful completion.
Non-zero Error.

ERRORS

If any of the following conditions occur, pthread_sigmask() or thr_sigsetmask() fails and returns the corresponding value:

EINVAL Value of how is not defined

If any of the following conditions are detected, pthread_sigmask() or thr_sigsetmask() fails and returns the corresponding value:

EFAULT set or oset are not valid addresses
The following example shows how to create a default thread that can serve as a signal catcher/handler with its own signal mask. "new" will have a different value from the creator's signal mask.

```c
#include <pthread.h>
#include <thread.h>

thread_t user_threadID;
sigset_t new;
void *handler(), interrupt();

main(int argc, char *argv[]) {
    test_argv(argv[1]);
    sigemptyset(&new);
    sigaddset(&new, SIGINT);
    switch(*argv[1]) { 
        case '0': /* POSIX */
            pthread_sigmask(SIG_BLOCK, &new, NULL);
            pthread_create(&user_threadID, NULL, handler, argv[1]);
            pthread_join(user_threadID, NULL);
            break;
        case '1': /* Solaris */
            thr_sigsetmask(SIG_BLOCK, &new, NULL);
            thr_create(NULL, 0, handler, argv[1], 0, &user_threadID);
            thr_join(user_threadID, NULL, NULL);
            break;
    } /* switch */

    printf("thread handler, # %d, has exited\n", user_threadID);
    sleep(2);
    printf("main thread, # %d is done\n", thr_self());
} /* end main */
```

struct sigaction act;

void *
handler(char argv1[])
{
    act.sa_handler = interrupt;
    sigaction(SIGINT, &act, NULL);
    switch(*argv1){
        case '0': /* POSIX */
            pthread_sigmask(SIG_BLOCK, &new, NULL);
            pthread_create(&user_threadID, NULL, handler, argv[1]);
            pthread_join(user_threadID, NULL);
            break;
        case '1': /* Solaris */
            thr_sigsetmask(SIG_BLOCK, &new, NULL);
            thr_create(NULL, 0, handler, argv[1], 0, &user_threadID);
            thr_join(user_threadID, NULL, NULL);
            break;
    } /* switch */
    printf("thread handler, # %d, has exited\n", user_threadID);
    sleep(2);
    printf("main thread, # %d is done\n", thr_self());
} /* end main */
```
case '0': /* POSIX */
    pthread_sigmask(SIG_UNBLOCK, &new, NULL);
    break;
  case '1': /* Solaris */
    thr_sigsetmask(SIG_UNBLOCK, &new, NULL);
    break;
  }
  printf("\n Press cntrl-C to deliver SIGINT signal to the process\n");
  sleep(8); /* give user time to hit cntrl-C */
}

void
interrupt(int sig)
{
    printf("thread %d caught signal %d\n", thr_self(), sig);
}

void test_argv(char argv1[])
{
    if(argv1 == NULL)
    {
        printf("use 0 as arg1 to use thr_create();\n    or use 1 as arg1 to use pthread_create();\n")
        exit(NULL);
    }
}

Since POSIX threads and Solaris threads are fully compatible even within the same process, this example uses pthread_create(3T) if you execute a.out 0, or thr_create(3T) if you execute a.out 1.

Here's an explanation of the above example:

- sigemptyset(3C) initializes a null signal set, "new". sigaddset(3C) packs the signal, SIGINT, into that new set.
- Either pthread_sigmask() or thr_sigsetmask() is used to mask the signal, SIGINT (cntrl-C), from the calling thread, which is main(). The signal is masked to guarantee that only the new thread will receive this signal.
- pthread_create() or thr_create() creates the signal-handling thread.
- Using pthread_join(3T) or thr_join(3T), main() then waits for the termination of that signal-handling thread, whose ID number is "user_threadID"; after which, main() will sleep(3C) for 2 seconds, and then the program terminates.
The signal-handling thread, "handler":
- Assigns the handler "interrupt()" to handle the signal SIGINT, via the call to sigaction(2).
- Resets its own signal set to not block the signal, SIGINT.
- Sleeps for 8 seconds to allow time for the user to deliver the signal, SIGINT, by pressing the cntrl-C keys.

In the example, the "handler" thread served as a signal-handler while also taking care of activity of its own (in this case, sleeping, although it could have been some other activity). A thread could be completely dedicated to signal-handling simply by waiting for the delivery of a selected signal by blocking with sigwait(2). Thus, the two subroutines in the previous example, "handler()" and "interrupt()", could have been replaced with the following routine:

```c
void *handler()
{
    int signal;
    printf("thread %d is waiting for you to press the cntrl-C keys\n", thr_self());
    sigwait(&new, &signal);
    printf("thread %d has received the signal %d \n", thr_self(), signal);
}
```

In this routine, one thread is dedicated to catching and handling the signal specified by the set "new", which allows main() and all of its other sub-threads, created after pthread_sigmask() or thr_sigsetmask() masked that signal, to continue uninterrupted. In fact, any use of sigwait(2) should be such that all threads block the signals passed to sigwait(2) at all times. Only the thread that calls sigwait() will get the signals. Note that the call to sigwait(2) takes two arguments.

For this type of background dedicated signal-handling routine, you may wish to use a Solaris daemon thread by passing the argument, THR_DAEMON, to thr_create(3T).

**ATTRIBUTES**

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe and Async-Signal-Safe</td>
</tr>
</tbody>
</table>

**SEE ALSO**
sigaction(2), sigprocmask(2), sigwait(2), cond_wait(3T), pthread_create(3T), pthread_join(3T), pthread_self(3T), sigsetops(3C), sleep(3C), attributes(5), standards(5)

**NOTES**

It is not possible to block signals that cannot be ignored (see sigaction(2)). If using the threads library, it is not possible to block the signals SIGLWP or SIGCANCEL, which are reserved by the threads library. Additionally, it is impossible to unblock the signal SIGWAITING, which is always blocked on all threads. This restriction is quietly enforced by the threads library.

modified 8 May 1997

SunOS 5.6

3T-1237
Using `sigwait(2)` in a dedicated thread allows asynchronously generated signals to be managed synchronously; however, `sigwait(2)` should never be used to manage synchronously generated signals.

Synchronously generated signals are exceptions that are generated by a thread and are directed at the thread causing the exception. Since `sigwait()` blocks waiting for signals, the blocking thread cannot receive a synchronously generated signal.

If `sigprocmask(2)` is used in a multi-threaded program, it will be the same as if `thr_sigsetmask()` or `pthread_sigmask()` has been called. Note that POSIX leaves the semantics of the call to `sigprocmask(2)` unspecified in a multi-threaded process, so programs that care about POSIX portability should not depend on this semantic.

If a signal is delivered while a thread is waiting on a condition variable, the `cond_wait()` will be interrupted (see `cond_wait(3T)`) and the handler will be executed. The handler should assume that the lock protecting the condition variable is held.

Although `pthread_sigmask()` is Async-Signal-Safe with respect to the Solaris environment, this safeness is not guaranteed to be portable to other POSIX domains.
NAME
pthread_testcancel – create cancellation point in the calling thread

SYNOPSIS
#include <pthread.h>
void pthread_testcancel();

DESCRIPTION
pthread_testcancel() allows you to force testing for cancellation. This is useful when you need to execute code that runs for long periods without encountering cancellation points; such as a library routine that executes long-running computations without cancellation points. This type of code can block cancellation for unacceptable long periods of time. One strategy for avoiding blocking cancellation for long periods, is to insert calls to pthread_testcancel() in the long-running computation code and to setup a cancellation handler in the library code, if required.

RETURN VALUES
pthread_testcancel() returns a void.

ERRORS
pthread_testcancel() returns no errors.

EXAMPLES
See cancellation(3T) for an example of using pthread_testcancel() to force testing for cancellation.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
Intro(3), cancellation(3T), condition(3T), pthread_cleanup_pop(3T),
pthread_cleanup_push(3T), pthread_exit(3T), pthread_join(3T),
pthread_setcancelstate(3T), pthread_setcanceltype(3T), setjmp(3C), attributes(5)

NOTES
See cancellation(3T) for a discussion of cancellation concepts.

pthread_testcancel() has no effect if cancellation is disabled.

Use pthread_testcancel() with pthread_setcanceltype() called with its cancelspec set to PTHREAD_CANCEL_DEFERRED. pthread_testcancel() operation is undefined if pthread_setcanceltype() was called with its cancelspec argument set to PTHREAD_CANCEL_ASYNCHRONOUS.

It is possible to kill a thread when it is holding a resource, such as lock or allocated memory. If that thread has not setup a cancellation cleanup handler to release the held resource, the application is "cancel-unsafe". See attributes(5) for a discussion of Cancel-Safety, Deferred-Cancel-Safety, and Asynchronous-Cancel-Safety.

modified 27 Jan 1997

SunOS 5.6

3T-1239
NAME ptsname – get name of the slave pseudo-terminal device

SYNOPSIS

#include <stdlib.h>

char *ptsname(int fd);

DESCRIPTION

The ptsname() function returns the name of the slave pseudo-terminal device associated with a master pseudo-terminal device. fd 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 fd is an invalid file descriptor or if the slave device name does not exist in the file system.

ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO

open(2), grantpt(3C), ttyname(3C), unlockpt(3C), attributes(5)
STREAMS Programming Guide
NAME  putc, putc_unlocked, putchar, putchar_unlocked, fputc, putw – put character or word on a stream

SYNOPSIS
#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);

DESCRIPTION
The putc() function writes \( c \) (converted to an unsigned char) onto the output \( stream \) (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 \( stream \) was opened with append mode, the character is appended to the output \( stream \). putchar(c) is defined as putc(c, stdout). putc() and putchar() are macros.

The putc_unlocked() and putchar_unlocked() functions are variants of putc() and putchar(), respectively, that do not lock the stream. It is the caller’s responsibility to acquire the stream lock before calling these functions and releasing the lock afterwards; see flockfile(3S) and stdio(3S).

The fputc() function behaves like putc(), but is a function rather than a macro. It runs more slowly than putc(), but it takes less space per invocation and its name can be passed as an argument to a function.

The putw() function writes the C int (word) \( w \) to the standard I/O output \( stream \) (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. The putw() function neither assumes nor causes special alignment in the file.

RETURN VALUES
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.

Failure will occur, for example, if the file \( stream \) is not open for writing or if the output file cannot grow.

ERRORS
The fputc(), putc(), putchar(), and putw() functions will fail if either the \( stream \) is unbuffered or the \( stream \)’s buffer needed to be flushed and:

EBFIGN The file is a regular file and an attempt was made to write at or beyond the offset maximum.

modified 30 Dec 1996

SunOS 5.6

3S-1241
putc (3S)  Standard I/O Functions

ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>See NOTES below.</td>
</tr>
</tbody>
</table>

SEE ALSO

write(2), intro(3), fclose(3S), ferror(3S), flockfile(3S), fopen(3S), printf(3S), puts(3S), setbuf(3S), stdio(3S), attributes(5)

NOTES

Because it is implemented as a macro, putc() evaluates a stream 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).

The fputc(), putc(), putchar(), and putw() functions are MT-Safe in multi-thread applications. The putc_unlocked() and putchar_unlocked() functions are unsafe in multi-thread applications.
NAME    putenv – change or add value to environment

SYNOPSIS #include <stdlib.h>
            int putenv(const char *string);

DESCRIPTION putenv() makes the value of the environment variable name equal to value by altering an existing variable or creating a new one. In either case, the string pointed to by string becomes part of the environment, so altering the string will change the environment. string points to a string of the form "name=value." The space used by string is no longer used once a new string-defining name 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.

ERRORS The putenv() function may fail if:
ENOMEM Insufficient memory was available.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO exec(2), getenv(3C), malloc(3C), attributes(5), environ(5)

NOTES This routine uses malloc(3C) to enlarge the environment.
After putenv() is called, environment variables are not in alphabetical order.
string should not be an automatic variable.
string 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 string 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  putp, tputs – apply padding information and output string

SYNOPSIS  #include <curses.h>
int putp (const char *str);
int tputs (const char *str, int affcnt, int (*putfunc) (int));

ARGUMENTS  str Is a pointer to a terminfo variable or return value from tgetstr(3XC), tgoto(3XC),
tigetstr(3XC), or tparm(3XC).
affcnt Is the number of lines affected, or 1 if not relevant.
putfunc Is the output function.

DESCRIPTION  The putp() and tputs() functions are low-level functions used to deal directly with the
terminfo database. The use of appropriate X/Open Curses functions is recommended for
most situations.
The tputs() function adds padding information and then outputs str. str must be a ter-
minfo string variable or the result value from tgetstr(), tgoto(), tigetstr(), or tparm().
The tputs() function replaces the padding specification (if one exists) with enough char-
acters to produce the specified delay. Characters are output one at a time to putfunc, a
user-specified function similar to putchar(3S).
The putp() function calls tputs() as follows:
   tputs(str, 1, putchar)

RETURN VALUES  On success, these functions return OK.

ERRORS  None.

USAGE  The output of putp() goes to stdout, not to the file descriptor, fildes, specified in
setupterm(3XC).

SEE ALSO  putchar(3S), setupterm(3XC), tgetent(3XC), tigetflag(3XC), terminfo(4)
NAME  putpwent – write password file entry

SYNOPSIS  
```c
#include <pwd.h>
int putpwent(const struct passwd *p, FILE *f);
```

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 f, which matches the format of /etc/passwd.

RETURN VALUES  putpwent() returns non-zero if an error was detected during its operation, otherwise zero.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO  getpwnam(3C), putspent(3C), attributes(5)

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

```c
#include <stdio.h>

int puts(const char *s);
int fputs(const char *s, FILE *stream);
```

DESCRIPTION

The `puts()` function writes the string pointed to by `s`, followed by a NEWLINE character, to the standard output stream `stdout` (see `intro(3)`).

The `fputs()` function writes the null-terminated string pointed to by `s` to the named output stream `stream`.

Neither function writes the terminating null character.

RETURN VALUES

On success both routines return the number of characters written; otherwise they return EOF.

ERRORS

The `puts()` and `fputs()` functions will fail if either the `stream` is unbuffered or the `stream`'s buffer needed to be flushed and:

- `EFBIG` The file is a regular file and an attempt was made to write at or beyond the offset maximum.

ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO

`write(2)`, `intro(3)`, `fclose(3S)`, `ferror(3S)`, `fopen(3S)`, `printf(3S)`, `putc(3S)`, `stdio(3S)`, `attributes(5)`

NOTES

The `puts()` function appends a NEWLINE character while `fputs()` does not.
NAME
putspent – write shadow password file entry

SYNOPSIS
#include <shadow.h>

int putspent(const struct spwd *p, FILE *fp);

DESCRIPTION
The putspent() routine is the inverse of getspent(). Given a pointer to a spwd structure
created by the getspent() routine (or the getspnam() routine), the putspent() routine
writes a line on the stream fp, which matches the format of /etc/shadow. The spwd struc-
ture contains the following members:

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;

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.

RETURN VALUES
The putspent() routine returns non-zero if an error was detected during its operation,
otherwise zero.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
getpwnam(3C), getspnam(3C), putpwent(3C), attributes(5)

NOTES
This routine is for internal use only, compatibility is not guaranteed.
Do not use without also using putpwent() to update the password file.
The use of this function is discouraged.
NAME  putwc – put wide character on a stream

SYNOPSIS  
```c
#include <stdio.h>
#include <wchar.h>
wint_t putwc(wint_t wc, FILE *stream);
```

DESCRIPTION  The `putwc()` function is equivalent to `fputwc(3S)`, except that if it is implemented as a macro it may evaluate `stream` more than once, so the argument should never be an expression with side-effects.

RETURN VALUES  Refer to `fputwc(3S)`.

ERRORS  Refer to `fputwc(3S)`.

USAGE  This interface is provided in order to align with some current implementations, and with possible future ISO standards. Because it may be implemented as a macro, `putwc()` may treat a `stream` argument with side-effects incorrectly. In particular, `putwc(wc, *f++)` may not work correctly. Therefore, use of this function is not recommended; `fputwc(3S)` should be used instead.

SEE ALSO  `fputwc(3S)`
NAME        putwchar – put wide character on stdout stream
SYNOPSIS    #include <wchar.h>
             wint_t putwchar(wint_t wc);
DESCRIPTION  The function call putwchar(wc) is equivalent to putwc(wc, stdout).
RETURN VALUES Refer to fputwc(3S).
ATTRIBUTES  See attributes(5) for descriptions of the following attributes:
             ATTRIBUTE TYPE | ATTRIBUTE VALUE
             MT-Level        | MT-Safe
SEE ALSO    fputwc(3S), putwc(3S), attributes(5)
NAME  putws – convert a string of Process Code characters to EUC characters

SYNOPSIS  
```c
#include <stdio.h>
#include <widec.h>
int putws(wchar_t *s);
```

DESCRIPTION  The `putws()` function converts the Process Code string (terminated by a `wchar_t)NULL`) pointed to by `s`, to an Extended Unix Code (EUC) string followed by a NEWLINE character, and writes it to the standard output stream `stdout`. It does not write the terminal null character.

RETURN VALUES  The `putws()` function returns the number of Process Code characters transformed and written. It returns `EOF` if it attempts to write to a file that has not been opened for writing.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  `ferror(3S), fopen(3S), fread(3S), getws(3S), printf(3S), putwc(3S), attributes(5)`
NAME  qsort – quick sort

SYNOPSIS  
#include <stdlib.h>

void qsort(void *base, size_t nel, size_t width,
        int (*compar)(const void *, const void *));

DESCRIPTION  The qsort( ) function is an implementation of the quick-sort algorithm. It sorts a table of data in place. The contents of the table are sorted in ascending order according to the user-supplied comparison function.

The base argument points to the element at the base of the table. The nel argument is the number of elements in the table. The width argument specifies the size of each element in bytes. The compar argument 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.

The contents of the table are sorted in ascending order according to the user supplied comparison function.

EXAMPLES  The following program sorts a simple array:

 static int intcompare(int *i, int *j)
 {  
   if (*i > *j)  
     return (1);
   if (*i < *j)  
     return (-1);
   return (0);
 }

 main()
 {
   int a[10];
   int i;
   a[0] = 9;
   a[1] = 8;
   a[2] = 7;
   a[3] = 6;
   a[4] = 5;
   a[5] = 4;
   a[6] = 3;
   a[7] = 2;
   a[8] = 1;
   a[9] = 0;
   }
qsort((char *) a, 10, sizeof(int), intcompare);
for (i=0; i<10; i++) printf(" %d",a[i]);
printf("\n");
}

ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO

sort(1), bsearch(3C), lsearch(3C), string(3C), attributes(5)

NOTES

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

The relative order in the output of two items that compare as equal is unpredictable.
NAME raise – send signal to program

SYNOPSIS
#include <signal.h>

int raise(int sig);

DESCRIPTION raise( ) sends the signal sig 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.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO getpid(2), kill(2), signal(3C), attributes(5)
### NAME
rand, srand – simple random number generator

### SYNOPSIS
```
/usr/ucb/cc [ flag ... ] file ...
int rand()
int srand( seed)
unsigned seed;
```

### DESCRIPTION
`rand()` uses a multiplicative congruential random number generator with period \(2^{32}\) to return successive pseudo-random numbers in the range from 0 to \(2^{31} - 1\).

`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.

### SEE ALSO
drand48(3C), rand(3C), random(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.

The spectral properties of `rand()` leave a great deal to be desired. drand48(3C) and random(3C) provide much better, though more elaborate, random-number generators.

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.
NAME  rand, srand, rand_r – simple random-number generator

SYNOPSIS  
```c
#include <stdlib.h>

int rand(void);
void srand(unsigned int seed);
int rand_r(unsigned int *seed);
```

DESCRIPTION  
rand( ) uses a multiplicative congruential random-number generator with period $2^{32}$ that returns successive pseudo-random numbers in the range from 0 to \texttt{RAND\_MAX} (defined in \texttt{<stdlib.h>}).

The function srand( ) uses the argument \texttt{seed} 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 \texttt{seed} 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 \texttt{seed} value of 1.

rand_r( ) has the same functionality as rand( ) except that a pointer to a seed \texttt{seed} must be supplied by the caller. The seed to be supplied is not the same seed as in srand( ).

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>See NOTES below.</td>
</tr>
</tbody>
</table>

SEE ALSO  drand48(3C), attributes(5)

NOTES  
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.

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.

The spectral properties of rand( ) are limited. drand48(3C) provides a much better, though more elaborate, random-number generator.

rand( ) is unsafe in multi-thread applications. rand_r( ) is MT-Safe, and should be used instead. srand( ) is unsafe in multi-thread applications.

modified 29 Dec 1996          SunOS 5.6          3C-1255
NAME  random, srandom, initstate, setstate – pseudorandom number functions

SYNOPSIS  #include <stdlib.h>
            long random(void);
            void srandom(unsigned int seed);
            char *initstate(unsigned int seed, char *state, size_t size);
            char *setstate(const char *state);

DESCRIPTION  The random() function uses a nonlinear additive feedback random-number generator employing a default state array size of 31 long integers to return successive pseudorandom numbers in the range from 0 to $2^{31}-1$. The period of this random-number generator is approximately $16 \times (2^{31}-1)$. The size of the state array determines the period of the random-number generator. Increasing the state array size increases the period.

The srandom() function initializes the current state array using the value of seed.

The random() and srandom() functions have (almost) the same calling sequence and initialization properties as rand() and srand() (see rand(3C)). The difference is that rand(3C) produces a much less random sequence—in fact, the low dozen bits generated by rand go through a cyclic pattern. All the bits generated by random() are usable. For example,

        random()&01

will produce a random binary value.

Unlike srand(), srandom() does not return the old seed because the amount of state information used is much more than a single word. Two other routines are provided to deal with restarting/changing random number generators. With 256 bytes of state information, the period of the random-number generator is greater than $2^{69}$.

Like rand(3C), random() produces by default a sequence of numbers that can be duplicated by calling srandom() with 1 as the seed.

The initstate() and setstate() functions handle restarting and changing random-number generators. The initstate() function allows a state array, pointed to by the state argument, to be initialized for future use. The size argument, which specifies the size in bytes of the state array, is used by initstate() to decide what type of random-number generator to use; the larger the state array, the more random the numbers. Values for the amount of state information are 8, 32, 64, 128, and 256 bytes. Other values greater than 8 bytes are rounded down to the nearest one of these values. For values smaller than 8, random() uses a simple linear congruential random number generator. The seed argument specifies a starting point for the random-number sequence and provides for restarting at the same point. The initstate() function returns a pointer to the previous state information array.

If initstate() has not been called, then random() behaves as though initstate() had been called with seed = 1 and size = 128.
If `initstate()` is called with `size < 8`, then `random()` uses a simple linear congruential random number generator.

Once a state has been initialized, `setstate()` allows switching between state arrays. The array defined by the `state` argument is used for further random-number generation until `initstate()` is called or `setstate()` is called again. The `setstate()` function returns a pointer to the previous state array.

**RETURN VALUES**

The `random()` function returns the generated pseudo-random number.

The `srandom()` function returns no value.

Upon successful completion, `initstate()` and `setstate()` return a pointer to the previous state array. Otherwise, a null pointer is returned.

**ERRORS**

No errors are defined.

**USAGE**

After initialization, a state array can be restarted at a different point in one of two ways:

- The `initstate()` function can be used, with the desired seed, state array, and size of the array.
- The `setstate()` function, with the desired state, can be used, followed by `srandom()` with the desired seed. The advantage of using both of these functions is that the size of the state array does not have to be saved once it is initialized.

**EXAMPLES**

```c
/* Initialize an array and pass it in to initstate. */
static long state1[32] = {
    3,
    0x9a319039, 0x32d9c024, 0xb663182, 0x5da1f342,
    0x749e56b, 0xbeb1dab0, 0xab5c5918, 0x946554fd,
    0x8c2e680f, 0xeb3d799f, 0xb11ee0b7, 0xd36886,
    0xda672e2a, 0x1588ca88, 0xe369735d, 0x904f35f7,
    0x7589d6, 0x6fa6f051, 0x616e6b96, 0x9ac9efdc,
    0xde3b81e0, 0xdfa6f5b0, 0xf103bc02, 0x48f340fb,
    0x36413f93, 0xc622c98, 0x5a42ab8, 0x8a88d77b,
    0xf5ad9d0e, 0x8999220b, 0x27fb47b9
};
main() {
    unsigned seed;
    int n;
    seed = 1;
    n = 128;
    initstate(seed, state1, n);
    setstate(state1);
    printf("%%d0,random()");
}
```
ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>See NOTES below.</td>
</tr>
</tbody>
</table>

SEE ALSO

drand48(3C), rand(3C), attributes(5)

NOTES

random() and srandom() are unsafe in multi-thread applications. Use of these interfaces 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**
```c
cc [ flag ... ] file ... -lsocket -lnsi [ library ... ]
int rcmd(char **ahost, unsigned short inport, const char*luser, const char *ruser,
const char *cmd, int *fd2p);
int rresvport(int *port);
int ruserok(const char *rhost, int suser, const char *ruser, const char *luser);
```

**DESCRIPTION**
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).

rcmd() looks up the host *ahost using gethostbyname(3N), returning −1 if the host does not exist. Otherwise *ahost is set to the standard name of the host and a connection is established to a server residing at the well-known Internet port inport.

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 fd2p is non-zero, then an auxiliary channel to a control process will be set up, and a descriptor for it will be placed in *fd2p. 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 fd2p is 0, then the standard error (file descriptor 2) of the remote command will be made the same as its standard output and no provision is made for sending arbitrary signals to the remote process, although you may be able to get its attention by using out-of-band data.

The protocol is described in detail in in.rshd(1M).

The rresvport() routine is used to obtain a socket bound to a privileged port number. This socket is suitable for use by rcmd() and several other routines. Privileged Internet ports are those in the range 1 to 1023. Only the super-user is allowed to bind a socket to a privileged port number. The application must pass in port, which must be in the range 512 to 1023. The system first tries to bind to that port number. If it fails, it then tries to bind to port numbers less than port until either it succeeds or port number 512 is reached.

ruserok() takes a remote host’s name, as returned by a gethostbyaddr() (see gethostbyname(3N)) routine, two user names and a flag indicating whether the local user’s name is that of the super-user. It then checks the files /etc/hosts.equiv and possibly .rhosts in the local user’s home directory to see if the request for service is allowed. 0 is returned if the machine name is listed in the /etc/hosts.equiv file, or the host and remote user name are found in the .rhosts file; otherwise ruserok() returns −1. If the
super-user flag is 1, the checking of the /etc/hosts.equiv file is bypassed.

**RETURN VALUES**

rcmd() returns a valid socket descriptor on success. It returns -1 on error and prints a diagnostic message on the standard error.

rresvport() returns a valid, bound socket descriptor on success. It returns -1 on error with the global value errno set according to the reason for failure.

**FILES**

/etc/hosts.equiv system trusted hosts and users

~/.rhosts user’s trusted hosts and users

**ATTRIBUTES**

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

**SEE ALSO**

rlogin(1), rsh(1), in.rexedc(1M), in.rshd(1M), intro(2), gethostbyname(3N), rexec(3N), attributes(5)

**NOTES**

The error code EAGAIN is overloaded to mean “All network ports in use.” These interfaces are unsafe in multithreaded applications. Unsafe interfaces should be called only from the main thread.
NAME  readdir – read a directory entry

SYNOPSIS  
/usr/ucb/cc [flag ...] file ...
#include <sys/types.h>
#include <sys/dir.h>
struct direct *readdir(dirp);
DIR *dirp;

DESCRIPTION  The readdir() function returns a pointer to a structure representing the directory entry at the current position in the directory stream to which dirp refers, and positions the directory stream at the next entry, except on read-only file systems. It returns a NULL pointer upon reaching the end of the directory stream, or upon detecting an invalid location in the directory. The readdir() function shall not return directory entries containing empty names. It is unspecified whether entries are returned for dot (.) or dot-dot (..). The pointer returned by readdir() points to data that may be overwritten by another call to readdir() on the same directory stream. This data shall not be overwritten by another call to readdir() on a different directory stream. The readdir() function may buffer several directory entries per actual read operation. The readdir() function marks for update the st_atime field of the directory each time the directory is actually read.

RETURN VALUES  The readdir() function returns NULL on failure and sets errno to indicate the error.

ERRORS  The readdir() function 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  buf() points to an illegal address.
EINVAL  A signal was caught during the read() or readv() function.
EINVAL  Attempted to read from a stream linked to a multiplexer.
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 `fildes` is on a remote machine and the link to that machine is no longer active.

ENXIO The device associated with `fildes` is a block special or character special file and the value of the file pointer is out of range.

EOVERFLOW The value of the `direct` structure member `d_ino` cannot be represented in an `ino_t`.

**USAGE**
The `readdir()` function has an explicit 64-bit equivalent. See `interface64`(5).

**SEE ALSO**
`getdents(2), readdir(3C), scandir(3B), interface64(5)`

**NOTES**
Use of these interfaces should be restricted to only applications written on BSD platforms. Use of these interfaces with any of the system libraries or in multi-thread applications is unsupported.
NAME  readdir, readdir_r – read directory

SYNOPSIS  
#include <sys/types.h>
#include <dirent.h>

struct dirent *readdir(DIR *dirp);
struct dirent *readdir_r(DIR *dirp, struct dirent *entry);

POSIX  
cc [ flag ... ] file ... -D_POSIX_PTHREAD_SEMANTICS [ library ... ]

int *readdir_r(DIR *dirp, struct dirent *entry, struct dirent **result);

DESCRIPTION  
The type DIR, which is defined in the header <dirent.h>, represents a directory stream, which is an ordered sequence of all the directory entries in a particular directory. Directory entries represent files; files may be removed from a directory or added to a directory asynchronously to the operation of readdir() and readdir_r().

readdir()  
The readdir() function returns a pointer to a structure representing the directory entry at the current position in the directory stream specified by the argument dirp, and positions the directory stream at the next entry. It returns a null pointer upon reaching the end of the directory stream. The structure dirent defined by the <dirent.h> header describes a directory entry.

If entries for dot or dot-dot exist, one entry will be returned for dot and one entry will be returned for dot-dot; otherwise they will not be returned.

The pointer returned by readdir() points to data which may be overwritten by another call to readdir() on the same directory stream. This data is not overwritten by another call to readdir() on a different directory stream.

If a file is removed from or added to the directory after the most recent call to opendir(3C) or rewinddir(3C), whether a subsequent call to readdir() returns an entry for that file is unspecified.

The readdir() function may buffer several directory entries per actual read operation; readdir() marks for update the st_atime field of the directory each time the directory is actually read.

After a call to fork(2), either the parent or child (but not both) may continue processing the directory stream using readdir(), rewinddir() or seekdir(3C). If both the parent and child processes use these functions, the result is undefined.

If the entry names a symbolic link, the value of the d_ino member is unspecified.

readdir_r()  
The readdir_r() function is equivalent to readdir() except that a buffer result must be supplied by the caller to store the result. The size should be sizeof(struct dirent) + (NAME_MAX) (that is, pathconf(_PC_NAME_MAX)) + 1. _PC_NAME_MAX is defined in <unistd.h>.

modified 20 Mar 1997
SunOS 5.6
The POSIX version (see `standards(5)`) of the `readdir_r()` function initializes the structure referenced by `entry` and stores a pointer to this structure in `result`.

**RETURN VALUES**

Upon successful completion, `readdir()` and `readdir_r()` return a pointer to an object of type `struct dirent`. When an error is encountered, a null pointer is returned and `errno` is set to indicate the error. When the end of the directory is encountered, a null pointer is returned and `errno` is not changed. The POSIX `readdir_r()` returns 0 if successful or an error number to indicate failure.

**ERRORS**

The `readdir()` function will fail if:

- **EOVERFLOW**  
  One of the values in the structure to be returned cannot be represented correctly.

The `readdir()` and `readdir_r()` functions will fail if:

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

- **ENOENT**  
  The current file pointer for the directory is not located at a valid entry.

The `readdir()` and `readdir_r()` functions may fail if:

- **EBADF**  
  The `dirp` argument does not refer to an open directory stream.

- **ENOENT**  
  The current position of the directory stream is invalid.

**USAGE**

The `readdir()` function should be used in conjunction with `opendir()`, `closedir()`, and `rewinddir()` to examine the contents of the directory. As `readdir()` returns a null pointer both at the end of the directory and on error, an application wishing to check for error situations should set `errno` to 0, then call `readdir()`, then check `errno` and if it is non-zero, assume an error has occurred.

The `readdir()` and `readdir_r()` functions have explicit 64-bit equivalents. See `interface64(5)`.

**EXAMPLES**

The following sample code will search the current directory for the entry `name`:

```c
    dirp = opendir(".");
    while ((dp = readdir(dirp)) != NULL)
      if (strcmp(dp->d_name, name) == 0) {
        closedir(dirp);
        return FOUND;
      }
    closedir(dirp);
    return NOT_FOUND;
```

3C-1264 SunOS 5.6 modified 20 Mar 1997
ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>See NOTES below.</td>
</tr>
</tbody>
</table>

SEE ALSO
fork(2), lstat(2), symlink(2), Intro(3), closedir(3C), opendir(3C), rewinddir(3C), seekdir(3C), attributes(5), interface64(5), standards(5)

NOTES
When compiling multithread programs, see Intro(3), Notes On Multithread Applications. readdir() is unsafe in multithread applications. readdir_r() is safe, and should be used instead.

Solaris 2.4 and earlier releases provided a readdir_r() interface as specified in POSIX.1c Draft 6. The final POSIX.1c standard changed the interface as described above. Support for the Draft 6 interface is provided for compatibility only and may not be supported in future releases. New applications and libraries should use the POSIX standard interface.

For POSIX.1c complaint applications, the _POSIX_PTHREAD_SEMANTICS and _REENTRANT flags are automatically turned on by defining the _POSIX_C_SOURCE flag with a value >= 199506L.

modified 20 Mar 1997
SunOS 5.6
3C-1265
NAME  
read_vtoc, write_vtoc – read and write a disk’s VTOC

SYNOPSIS  
#include <sys/vtoc.h>

cc [ flag ... ] file ... -ladm [ library ... ]
int read_vtoc(int fd, struct vtoc *vtoc);
int write_vtoc(int fd, struct vtoc *vtoc);

DESCRIPTION  
read_vtoc() returns the VTOC structure that is stored on the disk associated with the 
open file descriptor fd.
write_vtoc() stores the VTOC structure on at disk associated with the open file descriptor 
fd.
fd refers to any slice on a raw disk.

RETURN VALUES  
read_vtoc returns:
  positive number  Success. The positive number is the slice index associated 
                  with the open file descriptor.
  negative number  There are two possible error returns. VT_EIO indicates an 
                  I/O error occurred and VT_ERROR indicates an 
                  unknown error.
write_vtoc returns:
  0  Success
  negative number  There are three possible error returns. VT_EIO indicates 
                  an I/O error occurred, VT_ERROR indicates an unknown 
                  error, and VT EINVAL indicates an incorrect field within 
                  the VTOC.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO  
format(1M), fmthard(1M), prtvtoc(1M), ioctl(2), attributes(5), dkio(7I)

BUGS  
write_vtoc cannot write a VTOC on an unlabeled disk. Use format(1M) for this purpose.
NAME
realpath – resolve pathname

SYNOPSIS
#include <stdlib.h>

char *realpath(const char *file_name, char *resolved_name);

DESCRIPTION
The realpath() function derives, from the pathname pointed to by file_name, an absolute
pathname that names the same file, whose resolution does not involve ".", "..", or sym-

bolic links. The generated pathname is stored, up to a maximum of PATH_MAX bytes,
in the buffer pointed to by resolved_name.

The realpath() function can handle both relative and absolute path names. For absolute path names and the relative names whose resolved name cannot be expressed relatively
(for example, ../reldir), it returns the resolved absolute name. For the other relative path
names, it returns the resolved relative name.

RETURN VALUES
On successful completion, realpath() returns a pointer to the resolved name. Otherwise,
realpath() returns a null pointer and sets errno to indicate the error, and the contents of
the buffer pointed to by resolved_name are undefined.

ERRORS
The realpath() function will fail if:

EACCES Read or search permission was denied for a component of file_name.
EINVAL Either the file_name or resolved_name argument is a null pointer.
EIO An error occurred while reading from the file system.
ELOOP Too many symbolic links were encountered in resolving path.
ENAMETOOLONG
The file_name argument is longer than PATH_MAX or a pathname component
is longer than NAME_MAX.
ENOENT A component of file_name does not name an existing file or file_name points to
an empty string.
ENOTDIR A component of the path prefix is not a directory.
The realpath() function may fail if:
ENAMETOOLONG
Pathname resolution of a symbolic link produced an intermediate result
whose length exceeds PATH_MAX.
ENOMEM Insufficient storage space is available.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

modified 29 Dec 1996

SunOS 5.6

3C-1267
realpath(3C)  C Library Functions

SEE ALSO  getcwd(3C), sysconf(3C), attributes(5)

NOTES  realpath() operates on null-terminated strings.
One should have execute permission on all the directories in the given and the resolved path.
realpath() may fail to return to the current directory if an error occurs.
NAME reboot – reboot system or halt processor

SYNOPSIS
#include <sys/reboot.h>

int reboot(int howto, char *bootargs);

DESCRIPTION reboot() reboots the system. howto is an option passed to specify the behaviour of the system while rebooting. The function interface permits only one of RB_HALT, RB_ASKNAME or RB_AUTOBOOT to be passed. RB_AUTOBOOT is the default.

The howto options are:

RE_AUTOBOOT The machine is rebooted from the root filesystem on the default boot device. See boot(1M) and kernel(1M).

RB_HALT the processor is simply halted; no reboot takes place. RB_HALT should be used with caution.

RB_ASKNAME Interpreted by the bootstrap program and kernel, causing the user to be asked for pathnames during the bootstrap.

The interpretation of the bootargs argument is platform dependent.

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.

SEE ALSO intro(1M), boot(1M), halt(1M), init(1M), kernel(1M), reboot(1M), uadmin(2)

NOTES Any other howto argument causes the kernel file to boot.
Only the super-user may reboot() a machine.
NAME
re_comp, re_exec – compile and execute regular expressions

SYNOPSIS
```
#include <re_comp.h>
char *re_comp(const char *string);
int re_exec(const char *string);
```

DESCRIPTION
The `re_comp()` function converts a regular expression string (RE) into an internal form suitable for pattern matching. The `re_exec()` function compares the string pointed to by the `string` argument with the last regular expression passed to `re_comp()`. If `re_comp()` is called with a null pointer argument, the current regular expression remains unchanged.

Strings passed to both `re_comp()` and `re_exec()` must be terminated by a null byte, and may include NEWLINE characters.

The `re_comp()` and `re_exec()` functions support simple regular expressions, which are defined on the `regexp(5)` manual page. The regular expressions of the form `{m}`, `{m,n}`, or `{m,n}` are not supported.

RETURN VALUES
The `re_comp()` function returns a null pointer when the string pointed to by the `string` argument is successfully converted. Otherwise, a pointer to one of the following error message strings is returned:
- No previous regular expression
- Regular expression too long
- unmatched \(
- missing ]
- too many \() pairs
- unmatched \)

Upon successful completion, `re_exec()` returns 1 if `string` matches the last compiled regular expression. Otherwise, `re_exec()` returns 0 if `string` fails to match the last compiled regular expression, and −1 if the compiled regular expression is invalid (indicating an internal error).

ERRORS
No errors are defined.

USAGE
For portability to implementations conforming to X/Open standards prior to XPG4v2, `regcomp(3C)` and `regexec(3C)` are preferred to these functions.

SEE ALSO `grep(1)`, `regcmp(1)`, `regcmp(3C)`, `regcomp(3C)`, `regexec(3C)`, `regexpr(3G)`, `regexp(5)`, `standards(5)`
recv(3N)  recv, recvfrom, recvmsg – receive a message from a socket

SYNOPSIS

cc [ flag ...] file ... -lsocket -lssl [ library ... ]
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/uio.h>
int recv(int s, char *buf, int len, int flags);
int recvfrom(int s, char *buf, int len, int flags, struct sockaddr *from, int *fromlen);
int recvmsg(int s, struct msghdr *msg, int flags);

DESCRIPTION
recv(), recvfrom(), and recvmsg() are used to receive messages from another socket. recv() may be used only on a connected 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. s is a socket created with socket(3N).

If from is not a NULL pointer, the source address of the message is filled in. fromlen is a value-result parameter, initialized to the size of the buffer associated with from, 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)).

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.

The select() call may be used to determine when more data arrives.

The flags parameter is formed by ORing one or more of the following:

MSG_OOB  Read any “out-of-band” data present on the socket rather than the regular “in-band” data.

MSG_PEEK  “Peek” at the data present on the socket; the data is returned, but not consumed, so that a subsequent receive operation will see the same data.

The recvmsg() call uses a msghdr structure to minimize the number of directly supplied parameters. This structure is defined in <sys/socket.h> and includes the following members:

caddr_t    msg_name;  /* optional address */
int        msg_name len;  /* size of address */
struct iovec *msg_iov;  /* scatter/gather array */
int        msg_iovlen;  /* # elements in msg_iov */
caddr_t    msg_accrights;  /* access rights sent/received */
int        msg_accrightslen;

Here msg_name and msg_name len 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

modified 16 May 1997  SunOS 5.6  3N-1271
recv (3N)  Network Functions

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 s 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 s is not a socket.
ESTALE A stale NFS file handle exists.
EWOULDBLOCK The socket is marked non-blocking and the requested operation would block.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO fnctl(2), ioctl(2), read(2), connect(3N), getsockopt(3N), send(3N), socket(3N), attributes(5), socket(5)
NAME  recv – receive a message from a connected socket

SYNOPSIS  cc [ flag … ] file … −lxnet [ library … ]
#include <sys/socket.h>
ssize_t recv(int socket, void *buffer, size_t length, int flags);

DESCRIPTION  The recv() function receives messages from a connected socket. The function takes the following arguments:

socket  Specifies the socket file descriptor.
buffer  Points to a buffer where the message should be stored.
length  Specifies the length in bytes of the buffer pointed to by the buffer argument.
flags  Specifies the type of message reception. Values of this argument are formed by logically OR’ing zero or more of the following values:

MSG_PEEK  Peeks at an incoming message. The data is treated as unread and the next recv() or similar function will still return this data.

MSG_OOB  Requests out-of-band data. The significance and semantics of out-of-band data are protocol-specific.

MSG_WAITALL  Requests that the function block until the full amount of data requested can be returned. The function may return a smaller amount of data if a signal is caught, the connection is terminated, or an error is pending for the socket.

The recv() function returns the length of the message written to the buffer pointed to by the buffer argument. For message-based sockets such as SOCK_DGRAM and SOCK_SEQPACKET, the entire message must be read in a single operation. If a message is too long to fit in the supplied buffer, and MSG_PEEK is not set in the flags argument, the excess bytes are discarded. For stream-based sockets such as SOCK_STREAM, message boundaries are ignored. In this case, data is returned to the user as soon as it becomes available, and no data is discarded.

If the MSG_WAITALL flag is not set, data will be returned only up to the end of the first message.

If no messages are available at the socket and O_NONBLOCK is not set on the socket’s file descriptor, recv() blocks until a message arrives. If no messages are available at the socket and O_NONBLOCK is set on the socket’s file descriptor, recv() fails and sets errno to EAGAIN.

RETURN VALUES  Upon successful completion, recv() returns the length of the message in bytes. If no messages are available to be received and the peer has performed an orderly shutdown, recv() returns 0. Otherwise, −1 is returned and errno is set to indicate the error.

modified 16 May 1997
SunOS 5.6
ERRORS

The recv() function will fail if:

EBADF The socket argument is not a valid file descriptor.
ECONNRESET A connection was forcibly closed by a peer.
EINVAL The MSG_OOB flag is set and no out-of-band data is available.
ENOTCONN A receive is attempted on a connection-mode socket that is not connected.
ENOTSOCK The socket argument does not refer to a socket.
EOPNOTSUPP The specified flags are not supported for this socket type or protocol.
ETIMEDOUT The connection timed out during connection establishment, or due to a transmission timeout on active connection.
EAGAIN The socket’s file descriptor is marked O_NONBLOCK and no data is waiting to be received; or MSG_OOB is set and no out-of-band data is available and either the socket’s file descriptor is marked O_NONBLOCK or the socket does not support blocking to await out-of-band data.

The recv() function may fail if:

EIO An I/O error occurred while reading from or writing to the file system.
ENOBUFS Insufficient resources were available in the system to perform the operation.
ENOMEM Insufficient memory was available to fulfill the request.
ENOSR There were insufficient STREAMS resources available for the operation to complete.

USAGE

The recv() function is identical to recvfrom(3XN) with a zero address_len argument, and to read(2) if no flags are used.

The select(3C) and poll(2) functions can be used to determine when data is available to be received.

ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO

poll(2), read(2), write(2), recvmsg(3XN), recvfrom(3XN), select(3C), send(3XN), sendmsg(3XN), sendto(3XN), shutdown(3XN), socket(3XN), attributes(5), socket(5)
NAME
recvfrom — receive a message from a socket

SYNOPSIS
cc [ flag ...] file ... -lxnet [ library ... ]
#include <sys/socket.h>
ssize_t recvfrom(int socket, void * buffer, size_t length, int flags, struct sockaddr * address, size_t * address_len);

DESCRIPTION
The recvfrom() function receives a message from a connection-mode or connectionless-mode socket. It is normally used with connectionless-mode sockets because it permits the application to retrieve the source address of received data.

The function takes the following arguments:

- **socket**: Specifies the socket file descriptor.
- **buffer**: Points to the buffer where the message should be stored.
- **length**: Specifies the length in bytes of the buffer pointed to by the **buffer** argument.
- **flags**: Specifies the type of message reception. Values of this argument are formed by logically OR'ing zero or more of the following values:
  - **MSG_PEEK**: Peeks at an incoming message. The data is treated as unread and the next recvfrom() or similar function will still return this data.
  - **MSG_OOB**: Requests out-of-band data. The significance and semantics of out-of-band data are protocol-specific.
  - **MSG_WAITALL**: Requests that the function block until the full amount of data requested can be returned. The function may return a smaller amount of data if a signal is caught, the connection is terminated, or an error is pending for the socket.
- **address**: A null pointer, or points to a sockaddr structure in which the sending address is to be stored. The length and format of the address depend on the address family of the socket.
- **address_len**: Specifies the length of the sockaddr structure pointed to by the **address** argument.

The recvfrom() function returns the length of the message written to the buffer pointed to by the **buffer** argument. For message-based sockets such as SOCK_DGRAM and SOCK_SEQPACKET, the entire message must be read in a single operation. If a message is too long to fit in the supplied buffer, and **MSG_PEEK** is not set in the **flags** argument, the excess bytes are discarded. For stream-based sockets such as SOCK_STREAM, message boundaries are ignored. In this case, data is returned to the user as soon as it becomes available, and no data is discarded.
recvfrom (3XN)  X/Open Networking Services Library Functions

If the MSG_WAITALL flag is not set, data will be returned only up to the end of the first message.

Not all protocols provide the source address for messages. If the address argument is not a null pointer and the protocol provides the source address of messages, the source address of the received message is stored in the sockaddr structure pointed to by the address argument, and the length of this address is stored in the object pointed to by the address_len argument.

If the actual length of the address is greater than the length of the supplied sockaddr structure, the stored address will be truncated.

If the address argument is not a null pointer and the protocol does not provide the source address of messages, the value stored in the object pointed to by address is unspecified.

If no messages are available at the socket and O_NONBLOCK is not set on the socket’s file descriptor, recvfrom() blocks until a message arrives. If no messages are available at the socket and O_NONBLOCK is set on the socket’s file descriptor, recvfrom() fails and sets errno to EAGAIN.

RETURN VALUES
Upon successful completion, recvfrom() returns the length of the message in bytes. If no messages are available to be received and the peer has performed an orderly shutdown, recvfrom() returns 0. Otherwise the function returns −1 and sets errno to indicate the error.

ERRORS
The recvfrom() function will fail if:
- EBADF: The socket argument is not a valid file descriptor.
- ECONNRESET: A connection was forcibly closed by a peer.
- EINTR: A signal interrupted recvfrom() before any data was available.
- EINVAL: The MSG_OOB flag is set and no out-of-band data is available.
- ENOTCONN: A receive is attempted on a connection-mode socket that is not connected.
- ENOTSOCK: The socket argument does not refer to a socket.
- EOPNOTSUPP: The specified flags are not supported for this socket type.
- ETIMEDOUT: The connection timed out during connection establishment, or due to a transmission timeout on active connection.
- EAGAIN: The socket’s file descriptor is marked O_NONBLOCK and no data is waiting to be received; or MSG_OOB is set and no out-of-band data is available and either the socket’s file descriptor is marked O_NONBLOCK or the socket does not support blocking to await out-of-band data.

The recvfrom() function may fail if:
- EIO: An I/O error occurred while reading from or writing to the file system.
- ENOBUFS: Insufficient resources were available in the system to perform the operation.
- ENOMEM: Insufficient memory was available to fulfill the request.
ENOSR  There were insufficient STREAMS resources available for the operation to complete.

**USAGE**  The `select(3C)` and `poll(2)` functions can be used to determine when data is available to be received.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**SEE ALSO**  `poll(2)`, `read(2)`, `write(2)`, `recv(3XN)`, `recvmsg(3XN)`, `select(3C)`, `send(3XN)`, `sendmsg(3XN)`, `sendto(3XN)`, `shutdown(3XN)`, `socket(3XN)`, `attributes(5)`, `socket(5)`
recvmsg (3XN) X/Open Networking Services Library Functions

NAME recvmsg – receive a message from a socket

SYNOPSIS cc [ flag ...] file ... -lxnet [ library ...]
#include <sys/socket.h>
ssize_t recvmsg(int socket, struct msghdr *message, int flags);

DESCRIPTION The recvmsg() function receives a message from a connection-mode or connectionless-
mode socket. It is normally used with connectionless-mode sockets because it permits
the application to retrieve the source address of received data.
The function takes the following arguments:
socket Specifies the socket file descriptor.
message Points to a msghdr structure, containing both the buffer to store the
source address and the buffers for the incoming message. The length
and format of the address depend on the address family of the socket.
The msg_flags member is ignored on input, but may contain meaningful
values on output.
flags Specifies the type of message reception. Values of this argument are
formed by logically OR’ing zero or more of the following values:
MSG_OOB Requests out-of-band data. The significance and
semantics of out-of-band data are protocol-specific.
MSG_PEEK Peeks at the incoming message.
MSG_WAITALL Requests that the function block until the full amount
of data requested can be returned. The function may
return a smaller amount of data if a signal is caught,
the connection is terminated, or an error is pending for
the socket.
The recvmsg() function receives messages from unconnected or connected sockets and
returns the length of the message.
The recvmsg() function returns the total length of the message. For message-based sock-
ets such as SOCK_DGRAM and SOCK_SEQPACKET, the entire message must be read in a
single operation. If a message is too long to fit in the supplied buffers, and MSG_PEEK is
not set in the flags argument, the excess bytes are discarded, and MSG_TRUNC is set in
the msg_flags member of the msghdr structure. For stream-based sockets such as
SOCK_STREAM, message boundaries are ignored. In this case, data is returned to the
user as soon as it becomes available, and no data is discarded.
If the MSG_WAITALL flag is not set, data will be returned only up to the end of the first
message.
If no messages are available at the socket and O_NONBLOCK is not set on the socket’s file
descriptor, recvfrom(3XN) blocks until a message arrives. If no messages are available at
the socket and O_NONBLOCK is set on the socket’s file descriptor, recvfrom(3XN) func-
tion fails and sets errno to EAGAIN.
In the msghdr structure, the msg_name and msg_namelen members specify the source address if the socket is unconnected. If the socket is connected, the msg_name and msg_namelen members are ignored. The msg_name member may be a null pointer if no names are desired or required. The msg_iov and msg_iovlen members describe the scatter/gather locations.

On successful completion, the msg_flags member of the message header is the bitwise-inclusive OR of all of the following flags that indicate conditions detected for the received message:

- **MSG_EOR**  End of record was received (if supported by the protocol).
- **MSG_OOB**  Out-of-band data was received.
- **MSG_TRUNC**  Normal data was truncated.
- **MSG_CTRUNC**  Control data was truncated.

**RETURN VALUES**  Upon successful completion, recvmsg() returns the length of the message in bytes. If no messages are available to be received and the peer has performed an orderly shutdown, recvmsg() returns 0. Otherwise, −1 is returned and errno is set to indicate the error.

**ERRORS**  The recvmsg() function will fail if:

- **EBADF**  The socket argument is not a valid open file descriptor.
- **ENOTSOCK**  The socket argument does not refer to a socket.
- **EINVAL**  The sum of the iov_len values overflows an ssize_t.
- **EAGAIN**  The socket’s file descriptor is marked O_NONBLOCK and no data is waiting to be received; or MSG_OOB is set and no out-of-band data is available and either the socket’s file descriptor is marked O_NONBLOCK or the socket does not support blocking to await out-of-band data.
- **EINTR**  This function was interrupted by a signal before any data was available.
- **EOPNOTSUPP**  The specified flags are not supported for this socket type.
- **ENOTCONN**  A receive is attempted on a connection-mode socket that is not connected.
- **ETIMEDOUT**  The connection timed out during connection establishment, or due to a transmission timeout on active connection.
- **EINVAL**  The MSG_OOB flag is set and no out-of-band data is available.
- **ECONNRESET**  A connection was forcibly closed by a peer.

The recvmsg() function may fail if:

- **EINVAL**  The msg_iovlen member of the msghdr structure pointed to by msg is less than or equal to 0, or is greater than IOV_MAX.
- **EIO**  An I/O error occurred while reading from or writing to the file system.
- **ENOBUSFS**  Insufficient resources were available in the system to perform the operation.
- **ENOMEM**  Insufficient memory was available to fulfill the request.
ENOSR  There were insufficient STREAMS resources available for the operation to complete.

USAGE  The 
select(3C) and 
poll(2) functions can be used to determine when data is available to be received.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  poll(2), recv(3XN), recvfrom(3XN), select(3C), send(3XN), sendmsg(3XN), sendto(3XN), shutdown(3XN), socket(3XN), attributes(5), socket(5)
NAME  redrawwin, wredrawln – redraw screen or portion of screen

SYNOPSIS  
#include <curses.h>

int redrawwin(WINDOW *win);
int wredrawln(WINDOW *win, int beg_line, int num_lines);

ARGUMENTS  

win  Is a pointer to the window in which to redraw.
beg_line  Is the first line to redraw.
num_lines  Is the number of lines to redraw.

DESCRIPTION  
The redrawwin() and wredrawln() functions force portions of a window to be redrawn
to the terminal when the next refresh operation is performed.

The redrawwin() function forces the entire window win to be redrawn, while the
wredrawln() function forces only num_lines lines starting with beg_line to be redrawn.

Normally, refresh operations use optimization methods to reduce the actual amount of
the screen to redraw based on the current screen contents. These functions tell the refresh
operations not to attempt any optimization when redrawing the indicated areas.

These functions are useful when the data that exists on the screen is believed to be cor-
rupt and for applications such as screen editors that redraw portions of the screen.

RETURN VALUES  

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

ERRORS  

None.

SEE ALSO  
doupdate(3XC)
NAME
regcmp, regex – compile and execute regular expression

SYNOPSIS
#include <libgen.h>
char *regcmp(const char *string1, /* char *string2 */..., int (*)(char *)0*);
char *regex(const char *re, const char *subject, /* char *ret */...);
extern char *__loc1;

DESCRIPTION
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.

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.

[ ] . * $ Match the end of the string; \n matches a newline.
Within brackets the minus means through. 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 –.

+ A regular expression followed by + means one or more times. For example, [0-9]+ is equivalent to [0-9][0-9]∗.

{m} {m,} {m,u} Integer values enclosed in { } indicate the number of times the preceding regular expression is to be applied. The value m is the minimum number and u is a number, less than 256, which is the maximum. If only m is present (that is, {m}), it indicates the exact number of times the regular expression is to be applied. The value {m,u} is analogous to {m,infinity}. The plus (+) and star (*) operations are equivalent to {1} and {0,} respectively.

(...)$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.
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.

```c
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`.

```c
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`.

```c
#include "file.i"
char *string, *newcursor;
... 
newcursor = regex(name, string);
```

**FILES**

/usr/ccs/lib/libgen.a

**ATTRIBUTES**

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**SEE ALSO**
ed(1), regcmp(1), malloc(3C), attributes(5), regexp(5)

**NOTES**

The user program may run out of memory if `regcmp()` is called iteratively without freeing the vectors no longer required.

When compiling multi-thread applications, the `_REENTRANT` flag must be defined on the compile line. This flag should only be used in multi-thread applications.

modified 29 Dec 1996
SunOS 5.6
3C-1283
NAME regcomp, regexec, regerror, regfree – regular expression matching

SYNOPSIS #include <sys/types.h>
#include <regex.h>
int regcomp(regex_t *preg, const char *pattern, int flags);
int regexec(const regex_t *preg, const char *string, size_t nmatch, regmatch_t *pmatch[],
int flags);
size_t regerror(int errcode, const regex_t *preg, char *errbuf, size_t errbuf_size);
void regfree(regex_t *preg);

DESCRIPTION These functions interpret basic and extended regular expressions (described on the 
regex(5) manual page).
The structure type regex_t contains at least the following member:
size_t re_nsub Number of parenthesised subexpressions.
The structure type regmatch_t contains at least the following members:
regoff_t rm_so Byte offset from start of string to start of substring.
regoff_t rm_eo Byte offset from start of string of the first character after the end of substring.

regcomp() The regcomp() function will compile the regular expression contained in the string 
pointed to by the pattern argument and place the results in the structure pointed to by 
preg. The flags argument is the bitwise inclusive OR of zero or more of the following 
flags, which are defined in the header <regex.h>:
REG_EXTENDED Use Extended Regular Expressions.
REG_ICASE Ignore case in match.
REG_NOSUB Report only success/fail in regexec().
REG_NEWLINE Change the handling of NEWLINE characters, as described in the text.

The default regular expression type for pattern is a Basic Regular Expression. The application 
can specify Extended Regular Expressions using the REG_EXTENDED flags flag.
If the REG_NOSUB flag was not set in flags, then regcomp() will set re_nsub to the 
number of parenthesised subexpressions (delimited by \( \) in basic regular expressions 
or ( ) in extended regular expressions) found in pattern.

regexec() The regexec() function compares the null-terminated string specified by string with the 
compiled regular expression preg initialized by a previous call to regcomp(). The flags 
argument is the bitwise inclusive OR of zero or more of the following flags, which are 
defined in the header <regex.h>: 

3C-1284 SunOS 5.6 modified 20 Dec 1996
REG_NOTBOL  The first character of the string pointed to by string is not the beginning of the line. Therefore, the circumflex character (^), when taken as a special character, will not match the beginning of string.

REG_NOTEOL  The last character of the string pointed to by string is not the end of the line. Therefore, the dollar sign ($), when taken as a special character, will not match the end of string.

If nmatch is zero or REG_NOSUB was set in the flags argument to regcomp(), then regexec() will ignore the pmatch argument. Otherwise, the pmatch argument must point to an array with at least nmatch elements, and regexec() will fill in the elements of that array with offsets of the substrings of string that correspond to the parenthesised subexpressions of pattern: pmatch[i].rm_so will be the byte offset of the beginning and pmatch[i].rm_eo will be one greater than the byte offset of the end of substring i. (Subexpression i begins at the ith matched open parenthesis, counting from 1.) Offsets in pmatch[0] identify the substring that corresponds to the entire regular expression. Unused elements of pmatch up to pmatch[nmatch-1] will be filled with −1. If there are more than nmatch subexpressions in pattern (pattern itself counts as a subexpression), then regexec() will still do the match, but will record only the first nmatch substrings.

When matching a basic or extended regular expression, any given parenthesised subexpression of pattern might participate in the match of several different substrings of string, or it might not match any substring even though the pattern as a whole did match. The following rules are used to determine which substrings to report in pmatch when matching regular expressions:

1. If subexpression i in a regular expression is not contained within another subexpression, and it participated in the match several times, then the byte offsets in pmatch[i] will delimit the last such match.

2. If subexpression i is not contained within another subexpression, and it did not participate in an otherwise successful match, the byte offsets in pmatch[i] will be −1. A subexpression does not participate in the match when:

   * or \ \ appears immediately after the subexpression in a basic regular expression, or *, ?, or { appears immediately after the subexpression in an extended regular expression, and the subexpression did not match (matched zero times)

   or

   | is used in an extended regular expression to select this subexpression or another, and the other subexpression matched.

3. If subexpression i is contained within another subexpression j, and i is not contained within any other subexpression that is contained within j, and a match of subexpression j is reported in pmatch[j], then the match or non-match of subexpression i reported in pmatch[i] will be as described in 1. and 2. above, but within the substring reported in pmatch[j] rather than the whole string.

4. If subexpression i is contained in subexpression j, and the byte offsets in pmatch[j] are −1, then the pointers in pmatch[i] also will be −1.
5. If subexpression \(i\) matched a zero-length string, then both byte offsets in \(p\text{match}_i\) will be the byte offset of the character or NULL terminator immediately following the zero-length string.

If, when \texttt{regexec()} is called, the locale is different from when the regular expression was compiled, the result is undefined.

If \texttt{REG_NEWLINE} is not set in \texttt{flags}, then a NEWLINE character in \texttt{pattern or string} will be treated as an ordinary character. If \texttt{REG_NEWLINE} is set, then newline will be treated as an ordinary character except as follows:

1. A NEWLINE character in \texttt{string} will not be matched by a period outside a bracket expression or by any form of a non-matching list.
2. A circumflex (\(^\circ\)) in \texttt{pattern}, when used to specify expression anchoring will match the zero-length string immediately after a newline in \texttt{string}, regardless of the setting of \texttt{REG_NOTBOL}.
3. A dollar-sign ($) in \texttt{pattern}, when used to specify expression anchoring, will match the zero-length string immediately before a newline in \texttt{string}, regardless of the setting of \texttt{REG_NOTEOL}.

\texttt{regfree()} The \texttt{regfree()} function frees any memory allocated by \texttt{regcomp()} associated with \texttt{preg}.

The following constants are defined as error return values:

- \texttt{REG_NOMATCH}: \texttt{regexec()} failed to match.
- \texttt{REG_BADPAT}: Invalid regular expression.
- \texttt{REG_ECOLLATE}: Invalid collating element referenced.
- \texttt{REG_ECTYPE}: Invalid character class type referenced.
- \texttt{REG_EESCAPE}: Trailing \(\\) in \texttt{pattern}.
- \texttt{REG_ESUBREG}: Number in \texttt{\(\text{	extbackslash digit}\)} invalid or in error.
- \texttt{REG_EBRACK}: \[\] imbalance.
- \texttt{REG_ENOSYS}: The function is not supported.
- \texttt{REG_EPAREN}: \(\(\) or \(\) imbalance.
- \texttt{REG_EBRACE}: \(\{\} imbalance.
- \texttt{REG_BADBR}: Content of \(\{\} invalid: not a number, number too large, more than two numbers, first larger than second.
- \texttt{REG_ERANGE}: Invalid endpoint in range expression.
- \texttt{REG_ESPACE}: Out of memory.
- \texttt{REG_BADRPT}: ?, *, + not preceded by valid regular expression.

\texttt{regerror()} The \texttt{regerror()} function provides a mapping from error codes returned by \texttt{regcomp()} and \texttt{regexec()} to unspecified printable strings. It generates a string corresponding to the value of the \texttt{errcode} argument, which must be the last non-zero value returned by \texttt{regcomp()} or \texttt{regexec()} with the given value of \texttt{preg}. If \texttt{errcode} is not such a value, an error message indicating that the error code is invalid is returned.
If `preg` is a NULL pointer, but `errcode` is a value returned by a previous call to `regexec()` or `regcomp()`, the `regerror()` still generates an error string corresponding to the value of `errcode`.

If the `errbuf_size` argument is not zero, `regerror()` will place the generated string into the buffer of size `errbuf_size` bytes pointed to by `errbuf`. If the string (including the terminating NULL) cannot fit in the buffer, `regerror()` will truncate the string and null-terminate the result.

If `errbuf_size` is zero, `regerror()` ignores the `errbuf` argument, and returns the size of the buffer needed to hold the generated string.

If the `preg` argument to `regexec()` or `regfree()` is not a compiled regular expression returned by `regcomp()`, the result is undefined. A `preg` is no longer treated as a compiled regular expression after it is given to `regfree()`.

See `regex(5)` for BRE (Basic Regular Expression) Anchoring.

**RETURN VALUES**

The following values are returned by `regcomp()`:

- 0: successful completion
- non-zero: an error has occurred. The value returned is described in `<regex.h>`, and the content of `preg` is undefined.

The following values are returned by `regexec()`:

- 0: successful completion.
- REG_NOMATCH: no match
- REG_ENOSYS: the function is not supported.

The following values are returned by `regerror()`:

- 0: the function is not implemented.

Upon successful completion, the function returns the number of bytes needed to hold the entire generated string.

The `regfree()` function returns no value.

**USAGE**

An application could use:

```
regerror(code,preg,(char *)NULL,(size_t)0)
```

to find out how big a buffer is needed for the generated string, `malloc` a buffer to hold the string, and then call `regerror()` again to get the string (see `malloc(3C)`). Alternately, it could allocate a fixed, static buffer that is big enough to hold most strings, and then use `malloc()` to allocate a larger buffer if it finds that this is too small.
EXAMPLES

#include <regex.h>

/*
 * Match string against the extended regular expression in
 * pattern, treating errors as no match.
 *
 * return 1 for match, 0 for no match
 */

int
match(const char *string, char *pattern)
{
    int status;
    regex_t re;

    if (regcomp(&re, pattern, REG_EXTENDED | REG_NOSUB) != 0) {
        return(0); /* report error */
    }
    status = regexec(&re, string, (size_t) 0, NULL, 0);
    regfree(&re);
    if (status != 0) {
        return(0); /* report error */
    }
    return(1);
}

The following demonstrates how the REG_NOTBOL flag could be used with regexec() to
find all substrings in a line that match a pattern supplied by a user. (For simplicity of the
example, very little error checking is done.)

(void) regcomp (&re, pattern, 0);
/* this call to regexec() finds the first match on the line */
error = regexec (&re, &buffer[0], 1, &pm, 0);
while (error == 0) { /* while matches found */
    /* substring found between pm.rm_so and pm.rm_eo */
    /* This call to regexec() finds the next match */
    error = regexec (&re, buffer + pm.rm_eo, 1, &pm, REG_NOTBOL);
}

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe with exceptions</td>
</tr>
<tr>
<td>CSI</td>
<td>Enabled</td>
</tr>
</tbody>
</table>
SEE ALSO  
fnmatch(3C), glob(3C), malloc(3C), setlocale(3C), attributes(5), regex(5)

NOTES  
regcomp() can be used safely in a multi-thread application as long as setlocale(3C) is not being called to change the locale.
NAME  
regexpr, compile, step, advance – regular expression compile and match routines

SYNOPSIS  
cc [ flag ...] file ... -Igen [ library ... ]
#include <regexpr.h>

char *compile(char *instring, char *expbuf, const char *endbuf);
int step(const char *string, const char *expbuf);
int advance(const char *string, const char *expbuf);
extern char *loc1, *loc2, *locs;
extern int nbra, regerrno, reglength;
extern char *braslist[], *braelist[];

DESCRIPTION  
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).

The parameter instring is a null-terminated string representing the regular expression. The parameter expbuf points to the place where the compiled regular expression is to be placed. If expbuf 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.

The parameter endbuf is one more than the highest address where the compiled regular expression may be placed. This argument is ignored if expbuf is NULL. If the compiled expression cannot fit in (endbuf–expbuf) bytes, compile() returns NULL and regerrno (see below) is set to 50.

The parameter string is a pointer to a string of characters to be checked for a match. This string should be null-terminated.

The parameter expbuf is the compiled regular expression obtained by a call of the function compile().

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 string and loc2 points to the null at the end of string.

The purpose of step() is to step through the string argument until a match is found or until the end of string is reached. If the regular expression begins with ^, step() tries to match the regular expression at the beginning of the string only.
The `advance()` function is similar to `step()`; but, it only sets the variable `loc2` and always restricts matches to the beginning of the string.

If one is looking for successive matches in the same string of characters, `locs` should be set equal to `loc2`, and `step()` should be called with `string` equal to `loc2`. `locs` is used by commands like `ed` and `sed` so that global substitutions like `s/y*//g` do not loop forever, and is `NULL` by default.

The external variable `nbra` is used to determine the number of subexpressions in the compiled regular expression. `braslist` and `braelist` are arrays of character pointers that point to the start and end of the `nbra` subexpressions in the matched string. For example, after calling `step()` or `advance()` with string `sabcdefg` and regular expression `\(abcdef\)`, `braslist[0]` will point at `a` and `braelist[0]` will point at `g`. These arrays are used by commands like `ed` and `sed` for substitute replacement patterns that contain the `\n` notation for subexpressions.

Note that it is not necessary to use the external variables `regerrno`, `nbra`, `loc1`, `loc2`, `locs`, `braelist`, and `braslist` if one is only checking whether or not a string matches a regular expression.

### EXAMPLES

The following is similar to the regular expression code from `grep`:

```c
#include <regex.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:

<table>
<thead>
<tr>
<th>ERROR</th>
<th>MEANING</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>Range endpoint too large.</td>
</tr>
<tr>
<td>16</td>
<td>Bad number.</td>
</tr>
<tr>
<td>25</td>
<td>&quot;\digit&quot; out of range.</td>
</tr>
<tr>
<td>36</td>
<td>Illegal or missing delimiter.</td>
</tr>
<tr>
<td>41</td>
<td>No remembered search string.</td>
</tr>
<tr>
<td>42</td>
<td>(^\backslash) imbalance.</td>
</tr>
<tr>
<td>43</td>
<td>Too many (</td>
</tr>
</tbody>
</table>

modified 29 Dec 1996          SunOS 5.6            3G-1291
44 More than 2 numbers given in \{^\}.  
45 } expected after \.  
46 First number exceeds second in \{^\}.  
49 [ ] imbalance.  
50 Regular expression overflow.

ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO ed(1), grep(1), sed(1), malloc(3C), attributes(5), regexp(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  remainder – remainder function

SYNOPSIS  
```
#include <math.h>
double remainder(double x, double y);
```

DESCRIPTION  The `remainder()` function returns the floating point remainder \( r = x - ny \) when \( y \) is non-zero. The value \( n \) is the integral value nearest the exact value \( x/y \). When 
\[ |n - x/y| = \frac{1}{2}, \] the value \( n \) is chosen to be even.

The behaviour of `remainder()` is independent of the rounding mode.

RETURN VALUES  The `remainder()` function returns the floating point remainder \( r = x - ny \) when \( y \) is non-zero.

When \( y \) is 0, `remainder()` returns NaN. and sets `errno` to EDOM.

If the value of \( x \) is ±Inf, `remainder()` returns NaN and sets `errno` to EDOM.

If \( x \) or \( y \) is NaN, then the function returns NaN.

ERRORS  The `remainder()` function will fail if:

EDOM  The \( y \) argument is 0 or the \( x \) argument is positive or negative infinity.

USAGE  The `remainder()` function computes the remainder \( x \) REM \( y \) required by ANSI/IEEE 754 (IEC 559).

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  `fmod`(3M), `attributes`(5)
NAME    remove – remove file

SYNOPSIS #include <stdio.h>
int remove(const char *path);

DESCRIPTION remove() causes the file or empty directory whose name is the string pointed to by path
to be no longer accessible by that name. A subsequent attempt to open that file using that
name will fail, unless the file is created anew.

For files, remove() is identical to unlink(). For directories, remove() is identical to
rmdir().

See rmdir(2) and unlink(2) for a detailed list of failure conditions.

RETURN VALUES Upon successful completion, remove() returns a value of 0; otherwise, it returns a value
of −1 and sets errno to indicate an error.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO rmdir(2), unlink(2), attributes(5)
NAME
resetty, savetty – restore/save terminal modes

SYNOPSIS
#include <curses.h>
int resetty(void);
int savetty(void);

DESCRIPTION
The savetty() and resetty() functions save and restore the terminal state, respectively.
The savetty() function saves the current state in a buffer; the resetty() function restores
the state to that stored in the buffer at the time of the last savetty() call.

RETURN VALUES
On success, these functions return OK. Otherwise, they return ERR.

ERRORS
None.
NAME
resolver, res_init, res_mkquery, res_query, res_search, res_send, dn_comp, dn_expand – resolver routines

SYNOPSIS
cc [ flag ... ] file ... -lresolv -lsocket -lnsl [ library ... ]
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/nameser.h>
#include <resolv.h>

int res_init(void);
int res_mkquery(int op, const char *dname, int class, int type, const char *data,
    int datalen, struct rrec *newrr, u_char *buf, int buflen);
int res_query(const char *dname, int class, int type, u_char *answer, int anslen);
int res_search(const char *dname, int class, int type, u_char *answer, int anslen);
int res_send(u_char *msg, int msglen, u_char *answer, int anslen);
int dn_comp(const char *exp_dn, u_char *comp_dn, int length, u_char **dnptrs,
    u_char **lastdnptr);
int dn_expand(const u_char *msg, const u_char *eomorig, u_char *comp_dn,
    char exp_dn, int length);

DESCRIPTION
These routines are used for making, sending, and interpreting query and reply messages passed to and from Internet domain name servers.

The global structure _res holds options and state information. Option values can be set to affect the collective behavior of groups of resolver library routines. However, most resolver library routines use reasonable defaults so that the explicit enabling of an option is rarely required.

The library manual page entry for the resolver library (see libresolv(4)) includes public domain routines beyond those described here. Those function names that are exported but are not explained here are lower-level routines called by these routines. Their direct use is discouraged. If you do make direct use of unsupported routines, you do so at considerable added risk and with no expectation of documentation or other support beyond that available publicly.

Options for the resolver library are stored as a single bit mask containing the bitwise-OR sum of the options enabled. The options stored in _res.options are those defined in <resolv.h> and as follows. (The field _res.options is a member of the _res structure.)

RES_INIT True if the initial name server address and default domain name are initialized (that is, res_init() has been called).
RES_DEBUG Print debugging messages.
RES_AAAONLY  Accept authoritative answers only. With this option, `res_send()` will continue until it finds an authoritative answer or finds an error. Currently this option is not implemented.

RES_USEVC   Use TCP connections for queries instead of UDP datagrams.

RES_PRIMARY  Query primary server only. This option is not implemented.

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. `res_send()` does not do iterative queries and expects the name server to handle recursion.

RES_DEFNAMES If set, `res_search()` appends the default domain name to single-component names (names that do not contain a dot). This is useful only in programs that regularly do many queries. UDP should be the normal mode used.

RES_DNSRCH   Enables searching up through the current domain tree. If this option is set, `res_search()` searches for host names in the current domain and in parent domains. This is used by the standard host lookup routine `gethostbyname(3N)`. This option is enabled by default.

RES_NOALIASES This option turns off the user level aliasing feature controlled by the `HOSTALIASES` environment variable. Network daemons should set this option.

`res_init` If the system initialization file `resolv.conf` exists, `res_init()` reads it to get the default domain name, the search list, and the Internet address of the local name server or servers (see `resolv.conf(4)`). If no server is configured by the local `resolv.conf` file, `res_init()` tries to obtain name resolution services from the host on which it is running.

The `res_init()` function also sets the `RES_INIT` field of the `_res` global structure so that other service routines (`res_search()`) can determine for certain whether it needs to be called first before other processing begins.

In the absence of a `resolv.conf` configuration file, the current domain is either set to the value of the environmental variable `LOCALDOMAIN`, derived from the domain name (see

modified 30 Dec 1996  SunOS 5.6  3N-1297
NAME
rewind – reset file position indicator in a stream

SYNOPSIS
#include <stdio.h>
void rewind(FILE *stream);

DESCRIPTION
The call:

rewind(stream)

is equivalent to:

(void) fseek(stream, 0L, SEEK_SET)

except that rewind() also clears the error indicator.

RETURN VALUES
The rewind() function returns no value.

ERRORS
Refer to fseek(3S) with the exception of EINVAL which does not apply.

USAGE
Because rewind() does not return a value, an application wishing to detect errors should clear errno, then call rewind(), and if errno is non-zero, assume an error has occurred.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
fseek(3S), attributes(5)
NAME
rewinddir – reset position of directory stream to the beginning of a directory

SYNOPSIS
#include <sys/types.h>
#include <dirent.h>
void rewinddir(DIR *dirp);

DESCRIPTION
The rewinddir() function resets the position of the directory stream to which dirp refers
to the beginning of the directory. It also causes the directory stream to refer to the
current state of the corresponding directory, as a call to opendir(3C) would have done. If
dirp does not refer to a directory stream, the effect is undefined.

After a call to the fork(2) function, either the parent or child (but not both) may continue
processing the directory stream using readdir(3C), rewinddir() or seekdir(3C). If both
the parent and child processes use these functions, the result is undefined.

RETURN VALUES
The rewinddir() function does not return a value.

ERRORS
No errors are defined.

USAGE
The rewinddir() function should be used in conjunction with opendir(), readdir() and
closedir() to examine the contents of the directory. This method is recommended for
portability.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
fork(2), closedir(3C), opendir(3C), readdir(3C), seekdir(3C), attributes(5)
NAME
rexec – return stream to a remote command

SYNOPSIS
cc [ flag . . . ] file . . . -ls -lnsl [ library . . . ]
    int rexec(char **ahost, unsigned short inport, const char *user, const char *passwd,
    const char *cmd, int *fd2p);

DESCRIPTION
rexec() looks up the host *ahost using gethostbyname(3N), returning −1 if the host does
not exist. Otherwise *ahost is set to the standard name of the host. If a username and
password are both specified, then these are used to authenticate to the foreign host; oth-
erwise the user’s .netrc file in his home directory is searched for appropriate information.
If all this fails, the user is prompted for the information.
The port inport specifies which well-known DARPA Internet port to use for the connec-
tion. The protocol for connection is described in detail in in.rexecd(1M).
If the call succeeds, a socket of type SOCK_STREAM is returned to the caller, and given to
the remote command as its standard input and standard output. If fd2p is non-zero, then
an auxiliary channel to a control process will be setup, and a file descriptor for it will be
placed in *fd2p. The control process will return diagnostic output (file descriptor 2, the
standard error) from the command on this channel, and will also accept bytes on this
channel as signal numbers, to be forwarded to the process group of the command. If fd2p
is 0, then the standard error (file descriptor 2 of the remote command) will be made the
same as its standard output and no provision is made for sending arbitrary signals to the
remote process, although you may be able to get its attention by using out-of-band data.

RETURN VALUES
If rexec() succeeds, a file descriptor number, which is a socket of type SOCK_STREAM, is
returned by the routine. *ahost is set to the standard name of the host, and if fd2p is not
NULL, a file descriptor number is placed in *fd2p which represents the command’s stan-
dard error stream.
If rexec() fails, −1 is returned.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
in.rexecd(1M), gethostbyname(3N), getservbyname(3N), attributes(5)

NOTES
There is no way to specify options to the socket() call that rexec() makes.
This interface is unsafe in multithreaded applications. Unsafe interfaces should be called
only from the main thread.

3N-1300 SunOS 5.6 modified 30 Dec 1996
NAME  
rint – round-to-nearest integral value

SYNOPSIS  
cc [ flag ... ] file ... -lm [ library ... ]
#include <math.h>

double rint(double x);

DESCRIPTION  
The rint() function returns the integral value (represented as a double) nearest x in the direction of the current IEEE754 rounding mode.
If the current rounding mode rounds toward negative infinity, then rint() is identical to floor(3M). If the current rounding mode rounds toward positive infinity, then rint() is identical to ceil(3M).

RETURN VALUES  
Upon successful completion, the rint() function returns the integer (represented as a double precision number) nearest x in the direction of the current IEEE754 rounding mode.
When x is \pm \infty, rint() returns x.
If the value of x is NaN, NaN is returned.

ERRORS  
No errors will occur.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  
ceil(3M), floor(3M), isnan(3M), attributes(5)
NAME

ripofline – reserve screen line for dedicated purpose

SYNOPSIS

#include <curses.h>

int ripofline(int line, int (*init)(WINDOW *win, int width));

ARGUMENTS

line determines whether the screen line being reserved comes from the top of stdscr
(line is positive) or the bottom (line is negative).

init Is a pointer to a function that initializes the one-line window.

win Is a pointer to one-line window created by this function.

width Is the number of columns in the window pointed to by the win parameter.

DESCRIPTION

The ripofline() function reserves a screen line as a one line window.
To use this function, it must be called before you call initscr(3XC) or newterm(3XC).
When initscr() or newterm() is called, so is the function pointed to by init. The function
pointed to by init takes two arguments: a pointer to the one-line window and the number
of columns in that window. This function cannot use the LINES or COLS variables and
cannot call wrefresh(3XC) or doupdate(3XC), but may call wnoutrefresh(3XC).

RETURN VALUES

The ripofline() function always returns OK.

ERRORS

None.

SEE ALSO

doupdate(3XC), initscr(3XC), slk_atroff(3XC)
**NAME**
rpc – library routines for remote procedure calls

**SYNOPSIS**
```c
cc [ flag ... ] file ... -lnsl [ library ... ]
#include <rpc/rpc.h>
#include <netconfig.h>
```

**DESCRIPTION**
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.

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.

**Multithread Considerations**
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_createerr` (see `rpc_clnt_create(3N)`).

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).

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)`).

**Nettype**
Some of the high-level RPC interface routines take a nettype 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.

`nettype` can be one of the following:

- **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 nettype.

- **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.

modified 30 Dec 1996
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 side authenticators.
 */
typedef struct {
    struct opaque_auth ah_cred;
    struct opaque_auth ah_verf;
    union des_block ah_key;
    struct auth_ops {
        void (*ah_nextverf)();
        int (*ah_marshall)(); /* nextverf & serialize */
        int (*ah_validate)(); /* validate verifier */
        int (*ah_refresh)(); /* refresh credentials */
        void (*ah_destroy)(); /* destroy this structure */
    } *ah_ops;

caddr_t ah_private;
} AUTH;

The CLIENT Structure

/*
 * Client rpc handle.
 * Created by individual implementations.
 * Client is responsible for initializing auth.
 */
typedef struct {
    AUTH *cl_auth; /* authenticator */
    struct clnt_ops {
        enum clnt_stat (*cl_call)(); /* call remote procedure */
        void (*cl_abort)(); /* abort a call */
        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 xprt_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_recv)(); /* receive incoming requests */
        enum xprt_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;

modified 30 Dec 1996 SunOS 5.6 3N-1305
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_reg 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)

/**
 * 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_handly;     /* extra private word */
} XDR;

modified 30 Dec 1996  SunOS 5.6  3N-1307
<table>
<thead>
<tr>
<th>RPC Routine</th>
<th>Manual Reference Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>auth_destroy</td>
<td>rpc_clnt_auth(3N)</td>
</tr>
<tr>
<td>authdes_create</td>
<td>rpc_soc(3N)</td>
</tr>
<tr>
<td>authdes_getucred</td>
<td>secure_rpc(3N)</td>
</tr>
<tr>
<td>authdes_seccreate</td>
<td>secure_rpc(3N)</td>
</tr>
<tr>
<td>authkerb_getucred</td>
<td>kerberos_rpc(3N)</td>
</tr>
<tr>
<td>authkerb_seccreate</td>
<td>kerberos_rpc(3N)</td>
</tr>
<tr>
<td>authnone_create</td>
<td>rpc_clnt_auth(3N)</td>
</tr>
<tr>
<td>authsys_create</td>
<td>rpc_clnt_auth(3N)</td>
</tr>
<tr>
<td>authsys_create_default</td>
<td>rpc_clnt_auth(3N)</td>
</tr>
<tr>
<td>authunix_create</td>
<td>rpc_soc(3N)</td>
</tr>
<tr>
<td>authunix_create_default</td>
<td>rpc_soc(3N)</td>
</tr>
<tr>
<td>callrpc</td>
<td>rpc_soc(3N)</td>
</tr>
<tr>
<td>clnt_broadcast</td>
<td>rpc_soc(3N)</td>
</tr>
<tr>
<td>clnt_call</td>
<td>rpc_clnt_calls(3N)</td>
</tr>
<tr>
<td>clnt_control</td>
<td>rpc_clnt_create(3N)</td>
</tr>
<tr>
<td>clnt_create</td>
<td>rpc_clnt_create(3N)</td>
</tr>
<tr>
<td>clnt_destroy</td>
<td>rpc_clnt_create(3N)</td>
</tr>
<tr>
<td>clnt_dg_create</td>
<td>rpc_clnt_create(3N)</td>
</tr>
<tr>
<td>clnt_freeres</td>
<td>rpc_clnt_calls(3N)</td>
</tr>
<tr>
<td>clnt_geterr</td>
<td>rpc_clnt_calls(3N)</td>
</tr>
<tr>
<td>clnt_pcreateerror</td>
<td>rpc_clnt_create(3N)</td>
</tr>
<tr>
<td>clnt_perrno</td>
<td>rpc_clnt_calls(3N)</td>
</tr>
<tr>
<td>clnt_perror</td>
<td>rpc_clnt_calls(3N)</td>
</tr>
<tr>
<td>clnt_raw_create</td>
<td>rpc_clnt_create(3N)</td>
</tr>
<tr>
<td>clnt_spcreateerror</td>
<td>rpc_clnt_create(3N)</td>
</tr>
<tr>
<td>clnt_sperrno</td>
<td>rpc_clnt_calls(3N)</td>
</tr>
<tr>
<td>clnt_serror</td>
<td>rpc_clnt_calls(3N)</td>
</tr>
<tr>
<td>clnt_tli_create</td>
<td>rpc_clnt_create(3N)</td>
</tr>
<tr>
<td>clnt_tp_create</td>
<td>rpc_clnt_create(3N)</td>
</tr>
<tr>
<td>clnt_udpcreate</td>
<td>rpc_soc(3N)</td>
</tr>
<tr>
<td>clnt_vc_create</td>
<td>rpc_clnt_create(3N)</td>
</tr>
<tr>
<td>clntraw_create</td>
<td>rpc_soc(3N)</td>
</tr>
<tr>
<td>clnttcp_create</td>
<td>rpc_soc(3N)</td>
</tr>
<tr>
<td>clntudp_bufcreate</td>
<td>rpc_soc(3N)</td>
</tr>
<tr>
<td>get_myaddress</td>
<td>rpc_soc(3N)</td>
</tr>
<tr>
<td>getnetname</td>
<td>secure_rpc(3N)</td>
</tr>
<tr>
<td>host2netname</td>
<td>secure_rpc(3N)</td>
</tr>
<tr>
<td>key_decryptsession</td>
<td>secure_rpc(3N)</td>
</tr>
<tr>
<td>key_encryptsession</td>
<td>secure_rpc(3N)</td>
</tr>
<tr>
<td>key_gendes</td>
<td>secure_rpc(3N)</td>
</tr>
<tr>
<td>key_setsecret</td>
<td>secure_rpc(3N)</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>netname2host</td>
<td>secure_rpc(3N)</td>
</tr>
<tr>
<td>netname2user</td>
<td>secure_rpc(3N)</td>
</tr>
<tr>
<td>pmap_getmaps</td>
<td>rpc_soc(3N)</td>
</tr>
<tr>
<td>pmap_getport</td>
<td>rpc_soc(3N)</td>
</tr>
<tr>
<td>pmap_rmtcall</td>
<td>rpc_soc(3N)</td>
</tr>
<tr>
<td>pmap_set</td>
<td>rpc_soc(3N)</td>
</tr>
<tr>
<td>pmap_unset</td>
<td>rpc_soc(3N)</td>
</tr>
<tr>
<td>rac_drop</td>
<td>rpc_rac(3N)</td>
</tr>
<tr>
<td>rac_poll</td>
<td>rpc_rac(3N)</td>
</tr>
<tr>
<td>rac_recv</td>
<td>rpc_rac(3N)</td>
</tr>
<tr>
<td>rac_send</td>
<td>rpc_rac(3N)</td>
</tr>
<tr>
<td>registerrpc</td>
<td>rpc_soc(3N)</td>
</tr>
<tr>
<td>rpc_broadcast</td>
<td>rpc_clnt_calls(3N)</td>
</tr>
<tr>
<td>rpc_broadcast_exp</td>
<td>rpc_clnt_calls(3N)</td>
</tr>
<tr>
<td>rpc_call</td>
<td>rpc_clnt_calls(3N)</td>
</tr>
<tr>
<td>rpc_reg</td>
<td>rpc_svc_calls(3N)</td>
</tr>
<tr>
<td>svc_create</td>
<td>rpc_svc_create(3N)</td>
</tr>
<tr>
<td>svc_destroy</td>
<td>rpc_svc_create(3N)</td>
</tr>
<tr>
<td>svc_dg_create</td>
<td>rpc_svc_create(3N)</td>
</tr>
<tr>
<td>svc_dg_enablecache</td>
<td>rpc_svc_calls(3N)</td>
</tr>
<tr>
<td>svc_fd_create</td>
<td>rpc_svc_create(3N)</td>
</tr>
<tr>
<td>svc_fds</td>
<td>rpc_soc(3N)</td>
</tr>
<tr>
<td>svc_freeargs</td>
<td>rpc_svc_reg(3N)</td>
</tr>
<tr>
<td>svc_getargs</td>
<td>rpc_svc_reg(3N)</td>
</tr>
<tr>
<td>svc_getcaller</td>
<td>rpc_soc(3N)</td>
</tr>
<tr>
<td>svc_getreq</td>
<td>rpc_soc(3N)</td>
</tr>
<tr>
<td>svc_getreqset</td>
<td>rpc_svc_calls(3N)</td>
</tr>
<tr>
<td>svc_getrpccaller</td>
<td>rpc_svc_calls(3N)</td>
</tr>
<tr>
<td>svc_kerb_reg</td>
<td>kerberos_rpc(3N)</td>
</tr>
<tr>
<td>svc_raw_create</td>
<td>rpc_svc_create(3N)</td>
</tr>
<tr>
<td>svc_reg</td>
<td>rpc_svc_calls(3N)</td>
</tr>
<tr>
<td>svc_register</td>
<td>rpc_svc_reg(3N)</td>
</tr>
<tr>
<td>svc_run</td>
<td>rpc_svc_reg(3N)</td>
</tr>
<tr>
<td>svc_sendreply</td>
<td>rpc_svc_reg(3N)</td>
</tr>
<tr>
<td>svc_tli_create</td>
<td>rpc_svc_create(3N)</td>
</tr>
<tr>
<td>svc_tp_create</td>
<td>rpc_svc_create(3N)</td>
</tr>
<tr>
<td>svc_unreg</td>
<td>rpc_svc_calls(3N)</td>
</tr>
<tr>
<td>svc_unregister</td>
<td>rpc_soc(3N)</td>
</tr>
<tr>
<td>svc_vc_create</td>
<td>rpc_svc_create(3N)</td>
</tr>
<tr>
<td>svcerr_auth</td>
<td>rpc_svc_err(3N)</td>
</tr>
<tr>
<td>svcerr_decode</td>
<td>rpc_svc_err(3N)</td>
</tr>
<tr>
<td>svcerr_noproc</td>
<td>rpc_svc_err(3N)</td>
</tr>
<tr>
<td>svcerr_noprog</td>
<td>rpc_svc_err(3N)</td>
</tr>
<tr>
<td>svcerr_progvers</td>
<td>rpc_svc_err(3N)</td>
</tr>
<tr>
<td>svcerr_systemerr</td>
<td>rpc_svc_err(3N)</td>
</tr>
</tbody>
</table>

Modified 30 Dec 1996

SunOS 5.6

3N-1309
svcerr_weakauth            rpc_svc_err(3N)
svcfd_create               rpc_soc(3N)
svcraw_create              rpc_soc(3N)
svctcp_create              rpc_soc(3N)
svcudp_bufcreate           rpc_soc(3N)
svcudp_create              rpc_soc(3N)
user2netname               secure_rpc(3N)
xdr_accepted_reply         rpc_xdr(3N)
xdr_authsys_parms          rpc_xdr(3N)
xdr_authunix_parms         rpc_soc(3N)
xdr_callhdr                rpc_xdr(3N)
xdr_callmsg                rpc_xdr(3N)
xdr_opaque_auth            rpc_xdr(3N)
xdr_rejected_reply         rpc_xdr(3N)
xdr_replymsg               rpc_xdr(3N)
xprt_register              rpc_svc_calls(3N)
xprt_unregister            rpc_svc_calls(3N)

FILES /etc/netconfig

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe with exceptions</td>
</tr>
</tbody>
</table>

SEE ALSO getnetconfig(3N), getnetpath(3N), kerberos_rpc(3N), rpc_clnt_auth(3N),
rpc_clnt_calls(3N), rpc_clnt_create(3N), rpc_svc_calls(3N), rpc_svc_create(3N),
rpc_svc_err(3N), rpc_svc_reg(3N), rpc_xdr(3N), rpcbind(3N), secure_rpc(3N), xdr(3N),
netconfig(4), rpc(4), attributes(5), environ(5)
NAME

rpcbind, rpcb_getmaps, rpcb_getaddr, rpcb_gettime, rpcb_rmtcall, rpcb_set, rpcb_unset – library routines for RPC bind service

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

#include <rpc/rpc.h>

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 host. It uses the transport specified through netconf to contact the remote rpcbind service on host. This routine will return NULL, if the remote rpcbind could not be contacted.

bool_t rpcb_getaddr(const u_long progunm, 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 host that is registered with program number progunm, version versnum, and speaks the transport protocol associated with netconf. The address found is returned in svcaddr. svcaddr 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.

bool_t rpcb_gettime(const char *host, time_t *timep);
   
   This routine returns the time on host in timep. If host is NULL, rpcb_gettime() returns the time on its own machine. This routine returns TRUE if it succeeds, FALSE if it fails. rpcb_gettime() can be used to synchronize the time between the client and the remote server. This routine is particularly useful for secure RPC.

denum clnt_stat rpcb_rmtcall(const struct netconfig *netconf, const char *host, const u_long progunm, 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 host to make an RPC call on your behalf to a procedure on that host. The netconfig structure should correspond to a connectionless transport. The parameter *svcaddr will be modified to the server's address if the procedure succeeds (see rpc_call() and clnt_call() in rpc_clnt_calls(3N) for the definitions of other parameters).
This procedure should normally be used for a “ping” and nothing else. This routine allows programs to do lookup and call, all in one step.

Note: Even if the server is not running `rpcbind` does not return any error messages to the caller. In such a case, the caller times out.

Note: `rpcb_rmtcall()` is only available for connectionless transports.

```c
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.

```c
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)`).

**ATTRIBUTES**

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**SEE ALSO**

`rpcbind(1M), rpcinfo(1M), rpc_clnt_calls(3N), rpc_svc_calls(3N), attributes(5)`
NAME                          rpc_clnt_auth, auth_destroy, authnone_create, authsys_create, authsys_create_default — library routines for client side remote procedure call authentication

DESCRIPTION                  These routines are part of the RPC library that allows C language programs to make procedure calls on other machines across the network, with desired authentication.

These routines are normally called after creating the CLIENT handle. The cl_auth field of the CLIENT structure should be initialized by the AUTH structure returned by some of the following routines. The client’s authentication information is passed to the server when the RPC call is made.

Only the NULL and the SYS style of authentication is discussed here. For the DES style authentication, please refer to secure_rpc(3N). For the Kerberos style authentication, please refer to kerberos_rpc(3N).

The NULL and SYS style of authentication are safe in multithreaded applications. For the MT-level of the DES and Kerberos styles, see their respective pages.

Routines                    The following routines require that the header <rpc/rpc.h> be included (see rpc(3N) for the definition of the AUTH data structure).

#include <rpc/rpc.h>

void auth_destroy(AUTH *auth);

A function macro that destroys the authentication information associated with auth. Destruction usually involves deallocation of private data structures. The use of auth is undefined after calling auth_destroy().

AUTH *authnone_create(void);

Create and return an RPC authentication handle that passes nonusable authentication information with each remote procedure call. This is the default authentication used by RPC.

AUTH *authsys_create(const char *host, const uid_t uid, const gid_t gid,
const int len, const gid_t *aup_gids);

Create and return an RPC authentication handle that contains AUTH_SYS authentication information. The parameter host is the name of the machine on which the information was created; uid is the user’s user ID; gid is the user’s current group ID; len and aup_gids refer to a counted array of groups to which the user belongs.

AUTH *authsys_create_default(void);

Call authsys_create() with the appropriate parameters.
ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO

kerberos.rpc(3N), rpc(3N), rpc_clnt_calls(3N), rpc_clnt_create(3N), secure_rpc(3N), attributes(5)
RPC library routines allow C language programs to make procedure calls on other machines across the network. First, the client calls a procedure to send a request to the server. Upon receipt of the request, the server calls a dispatch routine to perform the requested service, and then sends back a reply.

The `clnt_call()`, `rpc_call()`, and `rpc_broadcast()` routines handle the client side of the procedure call. The remaining routines deal with error handling in the case of errors. Some of the routines take a CLIENT handle as one of the parameters. A CLIENT handle can be created by an RPC creation routine such as `clnt_create()` (see `rpc_clnt_create(3N)`). These routines are safe for use in multithreaded applications. CLIENT handles can be shared between threads, however in this implementation requests by different threads are serialized (that is, the first request will receive its results before the second request is sent).

The client issue a call using the `clnt_call()`, `rpc_call()`, or `rpc_broadcast()` macro. The client passes the procedure number, the input and output parameter XDR functions, and the addresses of the input and output parameters. The time allowed for the call to return is specified by `tout`.

If the call succeeds, the status returned is `RPC_SUCCESS`; otherwise, an appropriate status is returned.

The `clnt_freeres()` function frees any data allocated by the RPC/XDR system when it decoded the results of an RPC call. It passes the results and the XDR function describing the results as parameters. The function returns `1` if the results were successfully freed, and `0` otherwise.

The `clnt_geterr()` function copies the error structure out of the client handle to the

```c
#include <rpc/rpc.h>
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);

A function macro that calls the remote procedure `procnum` associated with the client handle, `clnt`, which is obtained with an RPC client creation routine such as `clnt_create()` (see `rpc_clnt_create(3N)`). The parameter `inproc` is the XDR function used to encode the procedure’s parameters, and `outproc` is the XDR function used to decode the procedure’s results; `in` is the address of the procedure’s argument(s), and `out` is the address of where to place the result(s). `tout` is the time allowed for results to be returned, which is overridden by a time-out set explicitly through `clnt_control()`, see `rpc_clnt_create(3N)`.

If the remote call succeeds, the status returned is `RPC_SUCCESS`, otherwise an appropriate status is returned.

bool_t clnt_freeres(CLIENT *clnt, const xdrproc_t outproc, caddr_t out);
A function macro that frees any data allocated by the RPC/XDR system when it decoded the results of an RPC call. The parameter `out` is the address of the results, and `outproc` is the XDR routine describing the results. This routine returns 1 if the results were successfully freed, and 0 otherwise.

void clnt_geterr(const CLIENT *clnt, struct rpc_err *errp);
A function macro that copies the error structure out of the client handle to the
structure at address *errp.

void clnt_perrno(const enum clnt_stat stat);
Print a message to standard error corresponding to the condition indicated by stat. A newline is appended. Normally used after a procedure call fails for a routine for which a client handle is not needed, for instance rpc_call().

void clnt_perror(const CLIENT *clnt, const char *s);
Print a message to the standard error indicating why an RPC call failed; clnt is the handle used to do the call. The message is prepended with string s and a colon. A newline is appended. Normally used after a remote procedure call fails for a routine which requires a client handle, for instance clnt_call().

char *clnt_sperrno(const enum clnt_stat stat);
Take the same arguments as clnt_perrno(), but instead of sending a message to the standard error indicating why an RPC call failed, return a pointer to a string which contains the message.

clnt_sperrno() is normally used instead of clnt_perrno() when the program does not have a standard error (as a program running as a server quite likely does not), or if the programmer does not want the message to be output with printf() (see printf(3S)), or if a message format different than that supported by clnt_perrno() is to be used. Note: unlike clnt_sperror() and clnt_spcreatererror() (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 \textit{out} is the same as \textit{out} passed to \texttt{rpc\_broadcast()}, except that the remote procedure’s output is decoded there; \textit{addr} points to the address of the machine that sent the results, and \textit{netconf} is the netconfig structure of the transport on which the remote server responded. If \texttt{eachresult()} returns 0, \texttt{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. \texttt{rpc\_broadcast()} uses \texttt{AUTH\_SYS} credentials by default (see \texttt{rpc\_clnt\_auth(3N)}).

\begin{verbatim}
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 resultproc\_t eachresult,
const int inittime, const int waittime, const char *nettype);
\end{verbatim}

Like \texttt{rpc\_broadcast()}, except that the initial timeout, \textit{inittime} and the maximum timeout, \textit{waittime} are specified in milliseconds.

\textit{inittime} is the initial time that \texttt{rpc\_broadcast\_exp()} waits before resending the request. After the first resend, the re-transmission interval increases exponentially until it exceeds \textit{waittime}.

\begin{verbatim}
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);
\end{verbatim}

Call the remote procedure associated with \texttt{prognum}, \texttt{versnum}, and \texttt{procnum} on the machine, \texttt{host}. The parameter \texttt{inproc} is used to encode the procedure’s parameters, and \texttt{outproc} is used to decode the procedure’s results; \texttt{in} is the address of the procedure’s argument(s), and \texttt{out} is the address of where to place the result(s). \texttt{nettype} can be any of the values listed on \texttt{rpc(3N)}. This routine returns \texttt{RPC\_SUCCESS} if it succeeds, or an appropriate status is returned. Use the \texttt{clnt\_permi()} routine to translate failure status into error messages.

Warning: \texttt{rpc\_call()} uses the first available transport belonging to the class \texttt{nettype}, on which it can create a connection. You do not have control of timeouts or authentication using this routine.

**ATTRIBUTES**

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**SEE ALSO**

\texttt{printf(3S), rpc(3N), rpc\_clnt\_auth(3N), rpc\_clnt\_create(3N), attributes(5)}

modified 30 Dec 1996

SunOS 5.6

3N-1317
RPC library routines allow C language programs to make procedure calls on other machines across the network. First a CLIENT handle is created and then the client calls a procedure to send a request to the server. On receipt of the request, the server calls a dispatch routine to perform the requested service, and then sends a reply.

These routines are MT-Safe. In the case of multithreaded applications, the _REENTRANT flag must be defined on the command line at compilation time (-D_REENTRANT). When the _REENTRANT flag is defined, rpc_createerr becomes a macro which enables each thread to have its own rpc_createerr.

Routines

See rpc(3N) for the definition of the CLIENT data structure.

```
#include <rpc/rpc.h>

bool_t clnt_control(CLIENT *clnt, const u_int req, char *info);
```

A function macro to change or retrieve various information about a client object. `req` indicates the type of operation, and `info` is a pointer to the information. For both connectionless and connection-oriented transports, the supported values of `req` and their argument types and what they do are:

- **CLSET_TIMEOUT** `struct timeval *` set total timeout
- **CLGET_TIMEOUT** `struct timeval *` get total timeout

Note: if you set the timeout using `clnt_control()`, the timeout argument passed by `clnt_call()` is ignored in all subsequent calls.

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.

- **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
Network Functions

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.

clint_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.

clint_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 clint_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_timed(const char *host,
const u_long prognum, const u_long versnum,
const char *nettype, const struct timeval *timeout);

Generic client creation routine which is similar to clnt_create() but which also has the additional parameter timeout that specifies the maximum amount of time allowed for each transport class tried. In all other respects, the clnt_create_timed() call behaves exactly like the clnt_create() call.

CLIENT *clnt_create_vers(const char *host, const u_long prognum,
 u_long *vers_outp, const u_long vers_low, const u_long vers_high,
 char *nettype);

modified 30 Dec 1996 SunOS 5.6 3N-1319
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.

```c
CLIENT *clnt_create_vers_timed(const char *host, const u_long prognum,
    u_long *vers_outp, const u_long vers_low, const u_long vers_high,
    char *nettype const struct timeval *timeout);
```

Generic client creation routine similar to `clnt_create_vers()` but with the additional parameter `timeout`, which specifies the maximum amount of time allowed for each transport class tried. In all other respects, the `clnt_create_vers_timed()` call behaves exactly like the `clnt_create_vers()` call.

```c
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)`).

```c
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_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_tp_create_timed(const char *host, const u_long prognum, const u_long versnum, const struct netconfig *netconf, const struct timeval *timeout);

Like clnt_tp_create() except clnt_tp_create_timed() has the extra parameter timeout which specifies the maximum time allowed for the creation attempt to succeed. In all other respects, the clnt_tp_create_timed() call behaves exactly like the clnt_tp_create() call.

CLIENT *clnt_vc_create(const int 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 connection-oriented transport. The remote program is located at address svcaddr. The parameter fildes is an open and bound file descriptor. The user may specify the size of the send and receive buffers with the parameters sendsz and recvsz; values of 0 choose suitable defaults. This routine returns NULL if it fails.

The address svcaddr should not be NULL and should point to the actual address of the remote program. clnt_vc_create() does not consult the remote rpcbind service for this information.

struct rpc_createerr rpc_createerr;

A global variable whose value is set by any RPC client handle creation routine that fails. It is used by the routine clnt_pcreateerror() to print the reason for the failure.

In multithreaded applications, rpc_createerr becomes a macro which enables each thread to have its own rpc_createerr.

ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

3N-1322 SunOS 5.6 modified 30 Dec 1996
SEE ALSO

rpcbind(1M), rpc(3N), rpc_clnt_auth(3N), rpc_clnt_calls(3N), rpc_svc_create(3N), svc_raw_create(3N), attributes(5)
NAME  
rpc_control – library routine for manipulating global RPC attributes for client and server applications

SYNOPSIS  
bool_t rpc_control(int op, void *info);

DESCRIPTION  
This RPC library routine allows applications to set and modify global RPC attributes that apply to clients as well as servers. At present, it supports only server side operations.

This function allows applications to set and modify global attributes that apply to client as well as server functions. \( op \) indicates the type of operation, and \( info \) is a pointer to the operation specific information. The supported values of \( op \) and their argument types, and what they do are:

- RPC_SVC_MTMODE_SET int * set multithread mode
- RPC_SVC_MTMODE_GET int * get multithread mode
- RPC_SVC_THRMAX_SET int * set maximum number of threads
- RPC_SVC_THRMAX_GET int * get maximum number of threads
- RPC_SVC_THRTOTAL_GET int * get number of active threads
- RPC_SVC_THRCREATES_GET int * get number of threads created
- RPC_SVC_THRERRORS_GET int * get number of thread create errors
- RPC_SVC_USE_POLLFD int * set number of file descriptors to unlimited

There are three multithread (MT) modes. These are:

- RPC_SVC_MT_NONE Single threaded mode (default)
- RPC_SVC_MT_AUTO Automatic MT mode
- RPC_SVC_MT_USER User MT mode

Unless the application sets the Automatic or User MT modes, it will stay in the default (single threaded) mode. See the Network Interfaces Programming Guide for the meanings of these modes and programming examples. Once a mode is set, it cannot be changed.

By default, the maximum number of threads that the server will create at any time is 16. This allows the service developer to put a bound on thread resources consumed by a server. If a server needs to process more than 16 client requests concurrently, the maximum number of threads must be set to the desired number. This parameter may be set at any time by the server.

Set and get operations will succeed even in modes where the operations don’t apply. For example, you can set the maximum number of threads in any mode, even though it makes sense only for the Automatic MT mode. All of the get operations except RPC_SVC_MTMODE_GET apply only to the Automatic MT mode, so values returned in other modes may be undefined.

By default, RPC servers are limited to a maximum of 1024 file descriptors or connections due to limitations in the historical interfaces svc_fdset(3N) and svc_getreqset(3N). Applications written to use the preferred interfaces of svc_pollfd(3N) and svc_getreq_poll(3N) can use an unlimited number of file descriptors. Setting \( info \) to point to a non-zero integer and \( op \) to RPC_SVC_USE_POLLFD removes the limitation.
RETURN VALUES

This routine returns **TRUE** if the operation was successful, and **FALSE** otherwise.

ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO

rpcbind(1M), rpc(3N), rpc_svc_calls(3N), attributes(5)
NAME  rpc_rac, rac_drop, rac_poll, rac_recv, rac_send – remote asynchronous calls

SYNOPSIS  

cc [ flag ...] file ... -lrac -lnsl [ library ... ]

#include <rpc/rpc.h>
#include <rpc/rac.h>

DESCRIPTION  

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.

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.

```c
#include <rpc/rpc.h>
void rac_drop(CLIENT *cl, void *h);
```

*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 *h*.

After a call to *rac_drop()* the handle referred to by *h* is invalid. It may no longer be used in any asynchronous operation.

```c
enum clnt_stat rac_poll(CLIENT *cl, void *h);
```

*rac_poll()* returns the status of the call currently in progress on the <CLIENT, asynchronous handle> tuple referred to by *cl* and *h*.

*rac_poll()* return values are:

- **RPC_SUCCESS**
  A reply has been received and is available for reading by *rac_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.

```c
enum clnt_stat rac_recv(CLIENT *cl, void *h);
```

`rac_recv()` 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_recv()` 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_recv()` will invoke the XDR decode procedure specified in the `rac_send()` call. After a call to `rac_recv()`, 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_send` calls on the same client handle are permitted, but may introduce unpredictable perturbations to the current timeout and retry model used by the RPC library.

The interface imposes a limit on the amount of time a call may be in progress before it is considered to have failed. This method was chosen over limitations on the number of retries because of a desire for transport independence.

**ATTRIBUTES**

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

**SEE ALSO**

`poll(2)`, `rpc(3N)`, `rpc_clnt_create(3N)`, `rpc_clnt_calls(3N)`, `xdr(3N)`, `attributes(5)`

**WARNINGS**

The RAC interface is not the recommended interface for having multiple RPC requests outstanding. The preferred method of accomplishing this in the Solaris environment is to use synchronous RPC calls with threads. The RAC interface is provided as a service to developers interested in porting RPC applications to Solaris 2.0. Use of this interface will degrade the performance of normal synchronous RPC calls (see `rpc_clnt_calls(3N)`). For these reasons, use of this interface is disparaged.

The library `librac` must be linked before `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.

3N-1328 SunOS 5.6 modified 30 Dec 1996
Network Functions

NAME
rpc_soc, authdes_create, authunix_create, authunix_create_default, callrpc,
clnt_broadcast, clntraw_create, clnttcp_create, clntudp_bufcreate, clntudp_create,
get_myaddress, getrpcport, pmap_getmaps, pmap_getport, pmap_rmtcall, pmap_set,
pmap_unset, regiserrpc, svc_fds, svc_getcaller, svc_getreq, svc_register, svc_unregister,
svcfd_create, svcraw_create, svctcp_create, svcudp_bufcreate, svcudp_create,
xdr_authunix_parms

DESCRIPTION
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.

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.

File Descriptors
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.

Routines
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.

#define PORTMAP
#include <rpc/rpc.h>

AUTH * authdes_create(char *name, unsigned window, struct sockaddr *syncaddr,
des_block *ckey);

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 keyserv daemon keyserv(1M) must be
running for the DES authentication system to work.

authdes_create(), used on the client side, returns 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() (see
secure_rpc(3N)). The second field is window on the validity of the client creden-
tial, given in seconds. A small window is more secure than a large one, but
choosing too small of a window will increase the frequency of resynchroniza-
tions because of clock drift. The third parameter syncaddr is optional. If it is

modified 30 Dec 1996
SunOS 5.6
3N-1329
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 .UX authentication information. The parameter host is the name of the machine on which the information was created; uid is the user’s user ID; gid is the user’s current group ID; grouplen and gidlistp refer to a counted array of groups to which the user belongs.

Warning: it is not very difficult to impersonate a user.

Warning: this routine exists for backward compatibility only, and is 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_perrno() (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 callers argument size is approximately 1500 bytes. Since the call message is sent to all connected networks, it may potentially lead to broadcast storms. *clnt_broadcast()* uses SB AUTH_SYS credentials by default (see *rpc_clnt_auth*(3N)).

Warning: this routine exists for backward compatibility only, and is obsoleted by *rpc_broadcast()* (see *rpc_clnt_calls*(3N)).

CLIENT * 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 *svcraw_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 progunm, u_long versnum, struct timeval wait, int *fdp);

This routine creates an RPC client handle for the remote program progunm, 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);

Placed the local system’s IP address into *addr, without consulting the library routines that deal with /etc/hosts. The port number is always set to htons(PMAPPORT).
Warning: this routine is only intended for use with the RPC library. It returns the local system’s address in a form compatible with the RPC library, and should not be taken as the system’s actual IP address. In fact, the *addr buffer’s host address part is actually zeroed. This address may have only local significance and should NOT be assumed to be an address that can be used to connect to the local system by remote systems or processes.

Warning: this routine remains for backward compatibility only. The routine netdir_getbyname() (see netdir(3N)) should be used with the name HOST_SELF to retrieve the local system’s network address as a netbuf structure.

u_short getrpcport(char *host, int prognum, int versnum, int proto)

callrpcport() returns the port number for the version versnum of the RPC program prognum running on host and using protocol proto. getrpcport() returns 0 if the RPC system failed to contact the remote portmap service, the program associated with prognum is not registered, or there is no mapping between the program and a port.

Warning: This routine exists for backward compatibility only. Enhanced functionality is provided by rpc_getaddr() (see rpcbind(3N)).

struct pmaplist * pmap_getmaps(struct sockaddr_in *addr);

A user interface to the portmap service, which returns a list of the current RPC program-to-port mappings on the host located at IP address addr. This routine can return NULL. The command ‘rpcinfo −p’ uses this routine.

Warning: this routine exists for backward compatibility only, enhanced functionality is provided by rpc_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 rpc_getaddr() (see rpcbind(3N)).

eenum clnt_stat pmap_rmtcall(struct sockaddr_in *addr, u_long prognum, u_long versnum, u_long procnum, char *in, xdrproc_t inproc, char *out, xdrproc_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 pmap_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 pmap_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 *xprt);
This routine returns the network address, represented as a struct sockaddr_in, of the caller of a procedure associated with the RPC service transport handle, xprt.
Warning: this routine exists for backward compatibility only, and is obsolete. The preferred interface is svc_getrpccaller() (see rpc_svc_reg(3N)), which returns the address as a struct netbuf.

void svc_getreq(int rdfds);
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; `rdfds` is the resultant read file descriptor bit mask. The routine 
returns when all file descriptors associated with the value of `rdfds` have been ser-
viced.
This routine is similar to `svc_getreqset()` but is limited to 32 descriptors.
Warning: this interface is obsoleted by `svc_getreqset()`.

```c
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)`).

```c
SVCXPRT * svcraw_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 rou-
tine 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.

```c
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 buf-
fered 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.

```c
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, \texttt{xprt}→\texttt{xp\_fd} is the transport’s file descriptor, and \texttt{xprt}→\texttt{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 \texttt{sendsz} and \texttt{recvsz} parameters.

Warning: this routine exists for backward compatibility only. \texttt{svc\_tli\_create()}, or \texttt{svc\_dg\_create()} (see \texttt{rpc\_svc\_create(3N)}) should be used instead.

\texttt{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 \texttt{fd}, which may be \texttt{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, \texttt{xprt}→\texttt{xp\_fd} is the transport’s file descriptor, and \texttt{xprt}→\texttt{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. \texttt{svc\_create()}, \texttt{svc\_tli\_create()}, or \texttt{svc\_dg\_create()} (see \texttt{rpc\_svc\_create(3N)}) should be used instead.

\texttt{registerrpc(u\_long prognum, u\_long versnum, u\_long procnum, char (**procname)(), xdrproc_t inproc, xdrproc_t outproc);} 
Register program \texttt{prognum}, procedure \texttt{procname}, and version \texttt{versnum} with the RPC service package. If a request arrives for program \texttt{prognum}, version \texttt{versnum}, and procedure \texttt{procnum}, \texttt{procname} is called with a pointer to its parameter(s); \texttt{procname} should return a pointer to its static result(s); \texttt{inproc} is used to decode the parameters while \texttt{outproc} is used to encode the results. This routine returns 0 if the registration succeeded, −1 otherwise.

\texttt{svc\_run()} must be called after all the services are registered.

Warning: this routine exists for backward compatibility only, and is obsoleted by \texttt{rpc\_reg()}.
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()`.

```c
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()`.

```c
xdr_authunix_parms(XDR *xdrs, struct authunix_parms *aupp);
```
Used for describing UNIX credentials. This routine is useful for users who wish to generate these credentials without using the RPC authentication package.
Warning: this routine exists for backward compatibility only, and is obsoleted by `xdr_authsys_parms()` (see `rpc_xdr(3N)`).

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

### See Also
`keyserv(1M), rpcbind(1M), rpcinfo(1M), netdir(3N), netdir_getbyname(3N), rpc(3N),
rpc_clnt_auth(3N), rpc_clnt_calls(3N), rpc_clnt_create(3N), rpc_svc_calls(3N),
rpc_svc_create(3N), rpc_svc_err(3N), rpc_svc_reg(3N), rpc_xdr(3N), rpcbind(3N),
secure_rpc(3N), select(3C), xdr_authsys_parms(3N), libnsl(4), librpcsoc(4), attributes(5)`

### Notes
These interfaces are unsafe in multi-threaded applications. Unsafe interfaces should be called only from the main thread.

modified 30 Dec 1996

SunOS 5.6

3N-1337
**NAME**

rpc_svc_calls, svc_dg_enablecache, svc_done, svc_exit, svc_fdset, svc_freeargs,
svc_getargs, svc_getreq_common, svc_getreq_poll, svc_getreqset, svc_getrpccaller,
svc_max_pollfd, svc_pollfd, svc_run, svc_sendreply – library routines for RPC servers

**DESCRIPTION**

These routines are part of the RPC library which allows C language programs to make procedure calls on other machines across the network.

These routines are associated with the server side of the RPC mechanism. Some of them are called by the server side dispatch function, while others (such as `svc_run()`) are called when the server is initiated.

In the current implementation, the service transport handle `SVCXPRT` contains a single data area for decoding arguments and encoding results. Therefore, this structure cannot be freely shared between threads that call functions that do this. However, when a server is operating in the Automatic or User MT modes, a copy of this structure is passed to the service dispatch procedure in order to enable concurrent request processing. Under these circumstances, some routines which would otherwise be unsafe, become safe. These are marked as such. Also marked are routines that are unsafe for MT applications, and are not to be used by such applications.

**Routines**

```
#include <rpc/rpc.h>

int svc_dg_enablecache(SVCXPRT *xprt, const unsigned long cache_size);
    This function allocates a duplicate request cache for the service endpoint xprt, large enough to hold cache_size entries. Once enabled, there is no way to disable caching. This routine returns 1 if space necessary for a cache of the given size was successfully allocated, and 0 otherwise.
    This function is safe in MT applications.

int svc_done(SVCXPRT *xprt);
    This function frees resources allocated to service a client request directed to the service endpoint xprt. This call pertains only to servers executing in the User MT mode. In the User MT mode, service procedures must invoke this call before returning, either after a client request has been serviced, or after an error or abnormal condition that prevents a reply from being sent. After `svc_done()` is invoked, the service endpoint xprt should not be referenced by the service procedure. Server multithreading modes and parameters can be set using the `rpc_control()` call.
    This function is safe in MT applications. It will have no effect if invoked in modes other than the User MT mode.

void svc_exit(void);
    This function when called by any of the RPC server procedure or otherwise, destroys all services registered by the server and causes `svc_run()` to return.
    If RPC server activity is to be resumed, services must be reregistered with the RPC library either through one of the `rpc_svc_create(3N)` functions, or using
```
xprt_register(3N).
svc_exit() has global scope and ends all RPC server activity.

fd_set svc_fdset;
A global variable reflecting the RPC server’s read file descriptor bit mask. This is only of interest if service implementors do not call svc_run(), but rather do their own asynchronous event processing. This variable is read-only, and it may change after calls to svc_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.

svc_fdset is limited to 1024 file descriptors and is considered obsolete. Use of svc_pollfd is recommended instead.

pollfd_t * svc_pollfd;
A global variable pointing to an array of pollfd_t structures reflecting the RPC server’s read file descriptor array. This is only of interest if service service implementors do not call svc_run() but rather do their own asynchronous event processing. This variable is read-only, and it may change after calls to svc_getreg_poll() or any creation routines. Do no pass its address to poll(2)!
Instead, pass the address of a copy.

By default, svc_pollfd is limited to 1024 entries. Use rpc_control(3N) to remove this limitation.

MT applications executing in either the Automatic MT mode or the user MT mode should never be read this variable. They should use auxiliary threads to do asynchronous event processing.

int svc_max_pollfd;
A global variable containing the maximum length of the svc_pollfd array. This variable is read-only, and it may change after calls to svc_getreg_poll() or any creation routines.

bool_t svc_freeargs(const SVCXPRT *xprt, const xdrproc_t inproc, caddr_t in);
A function macro that frees any data allocated by the RPC/XDR system when it decoded the arguments to a service procedure using svc_getargs(). This routine returns TRUE if the results were successfully freed, and FALSE otherwise.

This function macro is safe in MT applications utilizing the Automatic or User MT modes.

bool_t svc_getargs(const SVCXPRT *xprt, const xdrproc_t inproc, caddr_t in);
A function macro that decodes the arguments of an RPC request associated with the RPC service transport handle xprt. The parameter in is the address where the
arguments will be placed; \textit{inproc} is the XDR routine used to decode the arguments. This routine returns \texttt{TRUE} if decoding succeeds, and \texttt{FALSE} otherwise. This function macro is safe in MT applications utilizing the Automatic or User MT modes.

\texttt{void svc\_getreq\_common(const int \textit{fd});}

This routine is called to handle a request on the given file descriptor.

\texttt{void svc\_getreq\_poll(struct pollfd \ast pfds, const int \textit{pollretval});}

This routine is only of interest if a service implementor does not call \texttt{svc\_run()}, but instead implements custom asynchronous event processing. It is called when \texttt{poll(2)} has determined that an RPC request has arrived on one of the file descriptors; \textit{poll retval} is the return value from \texttt{poll(2)} and \textit{pfds} is the array of \texttt{pollfd} structures on which the \texttt{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.

\texttt{void svc\_getreqset(fd\_set \ast rdfds);}

This routine is only of interest if a service implementor does not call \texttt{svc\_run()}, but instead implements custom asynchronous event processing. It is called when \texttt{select(3C)} has determined that an RPC request has arrived on some RPC file descriptors; \textit{rdfds} is the resultant read file descriptor bit mask. The routine returns when all file descriptors associated with the value of \textit{rdfds} have been serviced.

This function macro is unsafe in MT applications.

\texttt{struct netbuf \ast svc\_getrpccaller(const SVCXPRT \ast xprt);}.

The approved way of getting the network address of the caller of a procedure associated with the RPC service transport handle \textit{xprt}.

This function macro is safe in MT applications.

\texttt{void svc\_run(\texttt{void});}

This routine never returns. In single threaded mode, it waits for RPC requests to arrive, and calls the appropriate service procedure using \texttt{svc\_getreq\_poll()} when one arrives. This procedure is usually waiting for the \texttt{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.

\texttt{bool\_t svc\_sendreply(const SVCXPRT \ast xprt, const xdrproc\_t \textit{outproc},
const caddr\_t \textit{out});}

Called by an RPC service’s dispatch routine to send the results of a remote
procedure call. The parameter xprt is the request’s associated transport handle; outproc is the XDR routine which is used to encode the results; and out is the address of the results. This routine returns TRUE if it succeeds, FALSE otherwise. This function macro is safe in MT applications utilizing the Automatic or User MT modes.

ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>See NOTES below.</td>
</tr>
</tbody>
</table>

SEE ALSO

rpcgen(1), poll(2), rpc(3N), rpc_control(3N), rpc_svc_create(3N), rpc_svc_err(3N), rpc_svc_reg(3N), select(3C), xprt_register(3N), attributes(5)

NOTES

svc_dg_enablecache() and svc_getrpccaller() are safe in multithreaded applications. svc_freeargs(), svc_getargs(), and svc_sendreply() are safe in MT applications utilizing the Automatic or User MT modes. svc_getreq_common(), svc_getreqset(), and svc_getreq_poll() are unsafe in multithreaded applications and should be called only from the main thread.
NAME
rpc_svc_create, svc_control, svc_create, svc_destroy, svc_sg_create, svc_fd_create,
svc_raw_create, svc_tli_create, svc_tp_create, svc_vc_create – library routines for the
creation of server handles

DESCRIPTION
These routines are part of the RPC library which allows C language programs to make
procedure calls on servers across the network. These routines deal with the creation of
service handles. Once the handle is created, the server can be invoked by calling
svc_run().

Routines
See rpc(3N) for the definition of the SVCXPRT data structure.

```
#include <rpc/rpc.h>

bool_t svc_control (SVCXPRT *svc, const u_int req, void *info);
```

A function to change or retrieve various information about a service object. `req`
indicates the type of operation and `info` is a pointer to the information. The sup-
ported values of `req`, their argument types, and what they do are:

**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 nor-
manly be returned. `info` should be a pointer to an integer. Upon
successful completion of the **SVCGET_VERSQUIET** request, `*info`
contains an integer which describes the server’s current
behavior: 0 indicates normal server behavior, that is, an
**RPC_PROGVERSMISMATCH** error will be returned; 1 indicates
that the out of range request will be silently ignored.

**SVCSET_VERSQUIET**
If a request is received for a program number served by this
server but the version number is outside the range registered
with the server, an **RPC_PROGVERSMISMATCH** error will nor-
manly be returned. It is sometimes desirable to change this
behavior. `info` should be a pointer to an integer which is either 0,
indicating normal server behavior and an
**RPC_PROGVERSMISMATCH** error will be returned, or 1, indi-
cating that the out of range request should be silently ignored.

**SVCGET_XID**
Returns the transaction ID of connection-oriented (vc) and con-
nectionless (dg) transport service calls. The transaction ID assists
in uniquely identifying client requests for a given RPC version,
program number, procedure, and client. The transaction ID is
extracted from the service transport handle `svc`; `info` must be a
pointer to an unsigned long. Upon successful completion of the
**SVCGET_XID** request, `*info` contains the transaction ID. Note that
rendezvous and raw service handles do not define a transaction
ID. Thus, if the service handle is of rendezvous or raw type, and
the request is of type SVC_GET_XID, svc_control() will return FALSE. Note also that the transaction ID read by the server can be set by the client through the suboption CLSET_XID in clnt_control(). See clnt_create(3N).

```c
int svc_create(const void (*dispatch)(const struct svc_req *, const SVCXPRT *),
               const u_long prognum, const u_long versnum, const char *nettype);
```

svc_create() creates server handles for all the transports belonging to the class nettype.

nettype defines a class of transports which can be used for a particular application. The transports are tried in left to right order in NETPATH variable or in top to bottom order in the netconfig database. If nettype is NULL, it defaults to netpath.

svc_create() registers itself with the rpcbind service (see rpcbind(1M)). dispatch is called when there is a remote procedure call for the given prognum and versnum; this requires calling svc_run() (see svc_run() in rpc_svc_reg(3N)). If svc_create() succeeds, it returns the number of server handles it created, otherwise it returns 0 and an error message is logged.

```c
void svc_destroy(SVCXPRT *xprt);
```

A function macro that destroys the RPC service handle xprt. Destruction usually involves deallocation of private data structures, including xprt itself. Use of xprt is undefined after calling this routine.

```c
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.

```c
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.

```c
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 clnt_create(3N)).
rpc_svc_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 bin-
daddr, 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 progrnum, 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 progrnum 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.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO rpcbind(1M), rpc(3N), rpc_clnt_create(3N), rpc_svc_calls(3N), rpc_svc_err(3N),
rpc_svc_reg(3N), attributes(5)
NAME  
rpc_svc_err, svcerr_auth, svcerr_decode, svcerr_noproc, svcerr_noprog, svcerr_progvers, svcerr_systemerr, svcerr_weakauth – library routines for server side remote procedure call errors

DESCRIPTION  
These routines are part of the RPC library which allows C language programs to make procedure calls on other machines across the network. These routines can be called by the server side dispatch function if there is any error in the transaction with the client.

Routines  
See rpc(3N) for the definition of the SVCXPRT data structure.

#include <rpc/rpc.h>

void svcerr_auth(const SVCXPRT *xprt, const enum auth_stat why);
   Called by a service dispatch routine that refuses to perform a remote procedure call due to an authentication error.

void svcerr_decode(const SVCXPRT *xprt);
   Called by a service dispatch routine that cannot successfully decode the remote parameters (see svc_getargs() in rpc_svc_reg(3N)).

void svcerr_noproc(const SVCXPRT *xprt);
   Called by a service dispatch routine that does not implement the procedure number that the caller requests.

void svcerr_noprog(const SVCXPRT *xprt);
   Called when the desired program is not registered with the RPC package. Service implementors usually do not need this routine.

void svcerr_progvers(const SVCXPRT *xprt, u_long low_vers, u_long high_vers);
   Called when the desired version of a program is not registered with the RPC package. low_vers is the lowest version number, and high_vers is the highest version number. Service implementors usually do not need this routine.

void svcerr_systemerr(const SVCXPRT *xprt);
   Called by a service dispatch routine when it detects a system error not covered by any particular protocol. For example, if a service can no longer allocate storage, it may call this routine.
void svcerr_weakauth(const SVCXPRT *xprt);

Called by a service dispatch routine that refuses to perform a remote procedure call due to insufficient (but correct) authentication parameters. The routine calls svcerr_auth(xprt, AUTH_TOOWEAK).

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO rpc(3N), rpc_svc_calls(3N), rpc_svc_create(3N), rpc_svc_reg(3N), attributes(5)
<table>
<thead>
<tr>
<th>NAME</th>
<th>rpc_svc_reg, rpc_reg, svc_reg, svc_unreg, svc_auth_reg, xprt_register, xprt_unregister – library routines for registering servers</th>
</tr>
</thead>
<tbody>
<tr>
<td>DESCRIPTION</td>
<td>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.</td>
</tr>
<tr>
<td>Routines</td>
<td>See rpc(3N) for the definition of the SVCXPRT data structure.</td>
</tr>
</tbody>
</table>

```c
#include <rpc/rpc.h>

bool_t rpc_reg(u_long prognum, u_long versnum, u_long procnum,
               char * const(*procname)(char *arg), xdrproc_t inproc, xdrproc_t outproc,
               const char *nettype);
```

Register program \texttt{prognum}, procedure \texttt{procname}, and version \texttt{versnum} with the RPC service package. If a request arrives for program \texttt{prognum}, version \texttt{versnum}, and procedure \texttt{procnum}, \texttt{procname} is called with a pointer to its parameters. \texttt{procname} should return a pointer to its static result(s). The \texttt{arg} parameter to \texttt{procname} is a pointer to the (decoded) procedure argument. \texttt{inproc} is the XDR function used to decode the parameters while \texttt{outproc} is the XDR function used to encode the results. Procedures are registered on all available transports of the class \texttt{nettype}. See rpc(3N). This routine returns 0 if the registration succeeded, \texttt{-1} otherwise.

```c
int svc_reg(const SVCXPRT *xprt, const u_long prognum, const u_long versnum,
            const void (*dispatch), const struct netconfig *netconf);
```

Associates \texttt{prognum} and \texttt{versnum} with the service dispatch procedure, \texttt{dispatch}. If \texttt{netconf} is \texttt{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 \texttt{netconf} is nonzero, then a mapping of the triple \texttt{[prognum, versnum, netconf]} to \texttt{xprt->xp_ltaddr} is established with the local rpcbind service.

The \texttt{svc_reg()} routine returns \texttt{1} if it succeeds, and \texttt{0} otherwise.

```c
void svc_unreg(const u_long prognum, const u_long versnum);
```

Remove from the rpcbind service, all mappings of the triple \texttt{[prognum, versnum, all-transports]} to network address and all mappings within the RPC service package of the double \texttt{[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.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO inetd(1M), rpcbind(1M), rpc(3N), rpc_svc_calls(3N), rpc_svc_create(3N), rpc_svc_err(3N), rpcbind(3N), select(3C), attributes(5)
### NAME
rpc_xdr, xdr_accepted_reply, xdr_authsys_Parms, xdr_callhdr, xdr_callmsg, xdr_opaque_auth, xdr_rejected_reply, xdr_replmsg – XDR library routines for remote procedure calls

### DESCRIPTION
These routines are used for describing the RPC messages in XDR language. They should normally be used by those who do not want to use the RPC package directly. These routines return TRUE if they succeed, FALSE otherwise.

### Routines
See rpc(3N) for the definition of the XDR data structure.

```c
#include <rpc/rpc.h>

bool_t xdr_accepted_reply(XDR *xdrs, const struct accepted_reply *ar);

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.

bool_t xdr_authsys_Parms(XDR *xdrs, struct authsys_Parms *aupp);

Used for describing UNIX operating system credentials. It includes machine-name, uid, gid list, etc.

void xdr_callhdr(XDR *xdrs, struct rpc_msg *chdr);

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.

bool_t xdr_callmsg(XDR *xdrs, struct rpc_msg *cmsg);

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.

bool_t xdr_opaque_auth(XDR *xdrs, struct opaque_auth *ap);

Used for describing RPC opaque authentication information messages.

bool_t xdr_rejected_reply(XDR *xdrs, const struct rejected_reply *rr);

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.
bool_t xdr_replymsg(XDR *xdrs, const struct rpc_msg *msg);

Used for describing RPC reply messages. It translates between the RPC reply message and its external representation. This reply could be either an acceptance, rejection or NULL.

ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  rpc(3N), xdr(3N), attributes(5)
NAME
rstat, havedisk – get performance data from remote kernel

PROTOCOL
/usr/include/rpcsvc/rstat

SYNOPSIS
cc [ flag ... ] file ... -lrpcsvc [ library ... ]
#include <rpc/rpc.h>
#include <rpcsvc/rstat.h>
enum clnt_stat rstat(char *host, struct statstime *statp);
havedisk(char *host);

DESCRIPTION
These routines require that the rpc.rstatd(1M) daemon be configured and available on the remote system indicated by host. The rstat() protocol is used to gather statistics from remote kernel. Statistics will be available on items such as paging, swapping, and cpu utilization.

rstat() fills in the statstime structure statp for host. statp 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).

havedisk() returns 1 if host has disk, 0 if it does not, and -1 if this cannot be determined.

The following XDR routines are available in librpcsvc:

xdr_statstime
xdr_statsvar

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
rpc.rstatd(1M), rup(1), rpc_clnt_calls(3N), attributes(5)

modified 30 Dec 1996

SunOS 5.6

3N-1351
rusers (3N)  Network Functions

NAME  rusers, rnusers – return information about users on remote machines

PROTOCOL  /usr/include/rpcsvc/rusers.x

SYNOPSIS  cc [ flag ... ] file ... -lrpcsvc [ library ... ]
#include <rpc/rpc.h>
#include <rpcsvc/rusers.h>
enum clnt_stat rusers(char *host, struct utmpidlearr *up);
int rnusers(char *host);

DESCRIPTION  These routines require that the rpc.rusersd(1M) daemon be configured and available on the remote system indicated by host. The rusers() protocol is used to retrieve information about users logged in on the remote system.

rusers() fills the utmpidlearr structure with data about host, and returns 0 if successful. up must point to an allocated utmpidlearr structure. If rusers() returns successful it will have allocated data structures within the up structure, which should be freed with xdr_free(3N) when you no longer need them:

xdr_free(xdr_utmpidlearr, up);

On error, the returned value can be interpreted as an enum clnt_stat and can be displayed with clnt_perror(3N) or clnt_spernno(3N).
See the header <rpcsvc/rusers.h> for a definition of struct utmpidlearr.

rnusers() returns the number of users logged on to host (-1 if it cannot determine that number).

The following XDR routines are available in librpcsvc:

xdr_utmpidlearr.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  rusers(1), rpc.rusersd(1M), rpc_clnt_calls(3N), xdr_free(3N), attributes(5)
NAME       rwall – write to specified remote machines

PROTOCOL   /usr/include/rpcsvc/rwall.x

SYNOPSIS   cc [ flag … ] file … -lrpcsvc [ library … ]
            #include <rpc/rpc.h>
            #include <rpcsvc/rwall.h>
            enum clnt_stat rwall(char *host, char *msg);

DESCRIPTION These routines require that the rpc.rwalld(1M) daemon be configured and available on
the remote system indicated by host.

rwall() executes wall(1M) on host. The rpc.rwalld process on host prints msg to all users
logged on to that system. rwall() returns RPC_SUCCESS if it was successful; otherwise a
enum clnt_stat is returned which can be displayed using clnt_perrno(3N).

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO       rpc.rwalld(1M), wall(1M), rpc_clnt_calls(3N), attributes(5)
NAME        rwlock, rwlock_init, rwlock_destroy, rw_rdlock, rw_wrlock, rw_tryrdlock, rw_trywrlock, rw_unlock – multiple readers, single writer locks

SYNOPSIS    cc [flag ...] file ... -lthread -lc [library ...]
#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);

DESCRIPTION Many threads can have simultaneous read-only access to data, while only one thread can have write access at any given time. Multiple read access with single write access is controlled by locks, which are generally used to protect data that is frequently searched. Readers/writer locks can synchronize threads in this process and other processes if they are allocated in writable memory and shared among cooperating processes (see mmap(2)), and are initialized for this purpose.

Additionally, readers/writer locks must be initialized prior to use. rwlock_init() The readers/writer lock pointed to by rwlp is initialized by rwlock_init(). A readers/writer lock is capable of having several types of behavior, which is specified by type. arg is currently not used, although a future type may define new behavior parameters via arg. type may be one of the following:

USYNC_PROCESS The readers/writer lock can synchronize threads in this process and other processes. The readers/writer lock should be initialized by only one process. arg is ignored. A readers/writer lock initialized with this type, must be allocated in memory shared between processes, i.e. either in Sys V shared memory (see shmodp(2)) or in memory mapped to a file (see mmap(2)). It is illegal to initialize the object this way and to not allocate it in such shared memory.

USYNC_THREAD The readers/writer lock can synchronize threads in this process, only. arg is ignored.

Additionally, readers/writer locks can be initialized by allocation in zeroed memory. A type of USYNC_THREAD is assumed in this case. Multiple threads must not simultaneously initialize the same readers/writer lock. And a readers/writer lock must not be re-initialized while in use by other threads.

The following are default readers/writer lock initialization (intra-process):

rwlock_t rwlp;
Threads Library

```
rwlock_init(&rwlp, NULL, NULL);
OR
rwlock_init(&rwlp, USYNC_THREAD, NULL);
OR
rwlock_t rwlp = DEFAULTRWLOCK;
```

The following is a customized readers/writer lock initialization (inter-process):
```
  rwlock_init(&rwlp, USYNC_PROCESS, NULL);
```

Any state associated with the readers/writer lock pointed to by `rwlp` are destroyed by `rwlock_destroy()` and the readers/writer lock storage space is not released.

`rw_rdlock()` gets a read lock on the readers/writer lock pointed to by `rwlp`. If the readers/writer lock is currently locked for writing, the calling thread blocks until the write lock is freed. Multiple threads may simultaneously hold a read lock on a readers/writer lock.

`rw_tryrdlock()` tries to get a read lock on the readers/writer lock pointed to by `rwlp`. If the readers/writer lock is locked for writing, it returns an error; otherwise, the read lock is acquired.

`rw_wrlock()` gets a write lock on the readers/writer lock pointed to by `rwlp`. If the readers/writer lock is currently locked for reading or writing, the calling thread blocks until all the read and write locks are freed. At any given time, only one thread may have a write lock on a readers/writer lock.

`rw_trywrlock()` tries to get a write lock on the readers/writer lock pointed to by `rwlp`. If the readers/writer lock is currently locked for reading or writing, it returns an error.

`rw_unlock()` unlocks a readers/writer lock pointed to by `rwlp`, if the readers/writer lock is locked and the calling thread holds the lock for either reading or writing. One of the other threads that is waiting for the readers/writer lock to be freed will be unblocked, provided there is other waiting threads. If the calling thread does not hold the lock for either reading or writing, no error status is returned, and the program’s behavior is unknown.

**RETURN VALUES**

Upon successful completion, 0 is returned; otherwise, a non-zero value indicates an error.

**ERRORS**

These functions fail and return the corresponding value if any of the following conditions are detected.

- **EINVAL** Invalid argument.
- **EFAULT** `rwlp` or `arg` point to an illegal address.
- **EBUSY** The readers/writer lock pointed to by `rwlp` was already locked.
ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO

mmap(2), attributes(5)

NOTES

These interfaces also available via:

```
#include <thread.h>
```

If multiple threads are waiting for a readers/writer lock, the acquisition order is random by default. However, some implementations may bias acquisition order to avoid depriving writers. The current implementation favors writers over readers.
NAME
scalb – load exponent of a radix-independent floating-point number

SYNOPSIS
#include <math.h>
double scalb(double x, double n);

DESCRIPTION
The scalb() function computes \( x \cdot r^n \), where \( r \) is the radix of the machine’s floating point arithmetic. When \( r \) is 2, scalb() is equivalent to ldexp(3C).

RETURN VALUES
Upon successful completion, the scalb() function returns \( x \cdot r^n \).
If the correct value would overflow, scalb() returns ±HUGE_VAL (according to the sign of \( x \)) and sets errno to ERANGE.
If the correct value would underflow to 0.0, scalb() returns 0 and sets errno to ERANGE.
The scalb() function returns \( x \) when \( x \) is ±Inf.
If \( x \) or \( n \) is NaN, then scalb() returns NaN.
For exceptional cases, matherr(3M) tabulates the values to be returned as dictated by Standards other than XPG4.

ERRORS
The scalb() function will fail if:
ERANGE The correct value would overflow or underflow.

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

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
ldexp(3C), matherr(3M), attributes(5)
scalbn (3M) Mathematical Library

NAME
scalbn – load exponent of a radix-independent floating-point number

SYNOPSIS
cc [ flag … ] file … -lm [ library … ]
#include <math.h>
double scalbn(double x, int n);

DESCRIPTION
The scalbn() function computes \( x \times r^n \), where \( r \) is the radix of the machine’s floating-point arithmetic.

RETURN VALUES
Upon successful completion, the scalbn() function returns \( x \times r^n \).
If the correct value would overflow, scalbn() returns ±HUGE_VAL (according to the sign of \( x \)).
The scalbn() function returns \( x \) when \( x \) is ±Inf.
If \( x \) is NaN, then scalbn() returns NaN.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
attributes(5)
NAME  scandir, alphasort – scan a directory

SYNOPSIS  
/usr/ucb/cc [ flag ... ] file ...
#include <sys/types.h>
#include <sys/dir.h>
int scandir(dirname, namelist, select, dcomp)
char *dirname;
struct direct *(+namelist[ ]);  
int (*select)(.), (*dcomp)( );
int alphasort(d1, d2)
struct direct **d1,**d2;

DESCRIPTION  The scandir() function reads the directory dirname 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.

The alphasort() function is a routine that sorts the array alphabetically.

RETURN VALUES  The scandir() function returns the number of entries in the array and a pointer to the array through the parameter namelist. The scandir() function returns −1 if the directory cannot be opened for reading or if malloc(3C) cannot allocate enough memory to hold all the data structures.

USAGE  The scandir() and alphasort() functions have explicit 64-bit equivalents. See interface64(5).

SEE ALSO  getdents(2), malloc(3C), qsort(3C), readdir(3B), readdir(3C), interface64(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.

modified 12 Sep 1996  SunOS 5.6  3B-1359
NAME    scanf, fscanf, sscanf – convert formatted input
SYNOPSIS  
#include <stdio.h>
int scanf(const char *format, ...);
int fscanf(FILE *strm, const char *format, ...);
int sscanf(const char *s, const char *format, ...);

DESCRIPTION  The scanf() function reads from the standard input stream, stdin.
The fscanf() function reads from the stream strm.
The sscanf() function reads from the character string s.
Each function reads characters, interprets them according to a format, and stores the
results in its arguments. Each expects, as arguments, a control string, format, 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.
The control string usually contains conversion specifications, which are used to direct
interpretation of input sequences. The control string may contain:

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 %d$igits$, an optional assignment suppression character *, a decimal
digit string that specifies an optional numerical maximum field width, an
optional letter l (ell), ll (ell ell), L, or h indicating the size of the receiving object,
and a conversion code:

% or digit, *, decimal digit string, h, or l, or ll, or L, conversion code
The following table defines which size indicators can be used with which conversion codes, and the size they indicate.

<table>
<thead>
<tr>
<th>Conversion Code</th>
<th>Size Indicator</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>d, i, n</td>
<td>none</td>
<td>signed int</td>
</tr>
<tr>
<td>h</td>
<td>signed short</td>
<td>signed short</td>
</tr>
<tr>
<td>l</td>
<td>signed long</td>
<td>signed long</td>
</tr>
<tr>
<td>ll</td>
<td>signed long long</td>
<td>signed long long</td>
</tr>
<tr>
<td>o, u, x</td>
<td>none</td>
<td>unsigned int</td>
</tr>
<tr>
<td>h</td>
<td>unsigned short</td>
<td>unsigned short</td>
</tr>
<tr>
<td>l</td>
<td>unsigned long</td>
<td>unsigned long</td>
</tr>
<tr>
<td>ll</td>
<td>unsigned long long</td>
<td>unsigned long long</td>
</tr>
<tr>
<td>e, f, g</td>
<td>none</td>
<td>float</td>
</tr>
<tr>
<td>l</td>
<td>double</td>
<td>double</td>
</tr>
<tr>
<td>L</td>
<td>long double</td>
<td>long double</td>
</tr>
</tbody>
</table>

The h, l, ll, 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 l and the character c, white space leading an input field is ignored.

Conversions can be applied to the nth 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 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.
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.

- **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 `strtof` 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 \(\text{printf}(3S)\) function. The corresponding argument should be a pointer to \(\text{void}\). 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 \(\text{cc}(1B)\)), 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 \(\text{LC_NUMERIC}\)). In the "C" locale, or in a locale where the decimal point character is not defined, the decimal point character defaults to a period (\(\cdot\)).

The \(\text{scanf()}\) 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.
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.

ERRORS
The scanf() and fscanf() functions will fail if data needs to be read and:

EOVERFLOW  The file is a regular file and an attempt was made to read at or beyond the offset maximum associated with the corresponding stream.

EXAMPLES
The call to the function scanf():

```c
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():

```c
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.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
<tr>
<td></td>
<td>Enabled</td>
</tr>
</tbody>
</table>

SEE ALSO  cc(1B), printf(3S), setlocale(3C), strtod(3C), strtol(3C), strtoul(3C), attributes(5)
NAME		schedctl_init, schedctl_lookup, schedctl_exit, schedctl_start, schedctl_stop — preemption control

SYNOPSIS	
tc [ flag ... ] file ... -lsched [ library ... ]
#include <schedctl.h>
schedctl_t *schedctl_init(void);
schedctl_t *schedctl_lookup(void);
void schedctl_exit(void);
void schedctl_start(schedctl_t *ptr);
void schedctl_stop(schedctl_t *ptr);

DESCRIPTION	
These functions provide limited control over the scheduling of a lightweight process (LWP). They allow a running LWP to give a hint to the kernel that preemptions of that LWP should be avoided. The most likely use for these functions is to block preemption while holding a spinlock. Improper use of this facility, including attempts to block preemption for sustained periods of time, may result in reduced performance.

schedctl_init() initializes preemption control for the calling LWP and returns a pointer used to refer to the data. If schedctl_init() is called more than once by the same LWP, the most recently returned pointer is the only valid one.

schedctl_lookup() returns the currently allocated preemption control data associated with the calling LWP that was previously returned by schedctl_init(). This can be useful in programs where it is difficult to maintain local state for each LWP.

schedctl_exit() removes the preemption control data associated with the calling LWP.

schedctl_start() is a macro that gives a hint to the kernel scheduler that preemption should be avoided on the current LWP. The pointer passed to the macro must be the same as the pointer returned by the call to schedctl_init() by the current LWP. The behavior of the program when other values are passed is undefined.

schedctl_stop() is a macro that removes the hint that was set by schedctl_start(). As with schedctl_start(), the pointer passed to the macro must be the same as the pointer returned by the call to schedctl_init() by the current LWP.

schedctl_start() and schedctl_stop() are intended to be used to bracket short critical sections, such as the time spent holding a spinlock. Other uses, including the failure to call schedctl_stop() soon after calling schedctl_start(), may result in poor performance.

RETURN VALUES	
schedctl_init() returns a pointer to a schedctl_t structure if the initialization was successful, or NULL otherwise. schedctl_lookup() returns a pointer to a schedctl_t structure if the data for that LWP was found, or NULL otherwise.

ERRORS	
None returned.

modified 10 May 1996 SunOS 5.6 3X-1365
SEE ALSO  priorctl(1), exec(2), fork(2), priorctl(2), thr_create(3T)

NOTES  Preemption control is intended for use by LWPs belonging to the time-sharing (TS) and interactive (IA) scheduling classes. If used by LWPs in other scheduling classes, such as real-time (RT), no errors will be returned but schedctl_start() and schedctl_stop() will not have any effect.

Use of preemption control by unbound threads in multithreaded applications (see thr_create(3T)) is not supported and will result in undefined behavior.

The data used for preemption control is not copied in the child of a fork(2). Thus, if a process containing LWPs using preemption control calls fork, and the child does not immediately call exec(2), each LWP in the child must call schedctl_init() again prior to any future uses of schedctl_start() and schedctl_stop(). Failure to do so will result in undefined behavior.
NAME sched_get_priority_max, sched_get_priority_min, sched_rr_get_interval – get scheduling parameter limits

SYNOPSIS 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 */
};

DESCRIPTION sched_get_priority_max() and sched_get_priority_min() return the appropriate maximum or minimum values, respectively, for the scheduling policy specified by policy.

sched_rr_get_interval() updates the timespec structure referenced by interval to contain the current execution time limit (i.e., time quantum) for the process specified by pid 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 pid is zero, the current execution time limit for the calling process is stored in interval.

The value of policy must be one of the scheduling policy values defined in <sched.h>: SCHED_FIFO, SCHED_RR, or SCHED_OTHER.

RETURN VALUES If successful, sched_get_priority_max() or sched_get_priority_min() returns the appropriate maximum or minimum values, respectively.

If successful, sched_rr_get_interval() returns 0.

If unsuccessful, these functions return -1, and set errno to indicate the error condition.

ERRORS EINVAL The value of policy does not represent a defined scheduling policy.
ENOSYS sched_get_priority_max(), sched_get_priority_min(), and sched_rr_get_interval() are not supported by this implementation.
ESRCH No process can be found corresponding to that specified by pid.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO sched_setparam(3R), sched_setscheduler(3R), attributes(5)

modified 30 Dec 1996 SunOS 5.6 3R-1367
NAME sched_setparam, sched_getparam – set/get scheduling parameters

SYNOPSIS 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 */
    ...
}

DESCRIPTION sched_setparam() sets the scheduling parameters of the process specified by pid to the values specified by the sched_param structure referenced by param.

sched_getparam() stores the scheduling parameters of a process, specified by pid, in the sched_param structure pointed to by param.

If the target process has as its scheduling policy, SCHED_FIFO or SCHED_RR:

If pid is zero, the scheduling parameters are set/stored for the calling process. Otherwise, if a process specified by pid exists and if the calling process has permission, the scheduling parameters are set/stored for the process whose process ID is equal to pid. 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).

The target process, pid, whether it is running or not running, resumes execution after all other runnable processes of equal or greater priority have been scheduled to run.

If the priority of the process, pid, is set higher than that of the lowest priority running process, and if process pid is ready to run, then process pid 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.

The value of param->sched_priority must be an integer within the inclusive priority range for the current scheduling policy of the process specified by pid. Higher numerical values for the priority represent higher priorities.

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`.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO `intro(2)`, `exec(2)`, `sched_setscheduler(3R)`, `attributes(5)`
NAME sched_setscheduler, sched_getscheduler – set/get scheduling policy and scheduling parameters

SYNOPSIS 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 */
    ... 
}

DESCRIPTION sched_setscheduler() sets the scheduling policy and scheduling parameters of the process specified by pid to policy and the parameters specified in the sched_param structure pointed to by param, respectively. The value of param->sched_priority must be any integer within the inclusive priority range for the scheduling policy specified by policy.

The possible values for the policy parameter are defined in the header file <sched.h>:
SCHED_FIFO, SCHED_RR, or SCHED_OTHER.

If pid is zero, the scheduling policy and scheduling parameters are set for the calling process. Otherwise, if a process specified by pid 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 pid. 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 superuser. See intro(2).

To change the policy 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.

sched_getscheduler() returns the scheduling policy of the process specified by pid. If pid is zero, the scheduling policy is returned for the calling process. Otherwise, if a process specified by pid exists and if the calling process has permission, the scheduling policy is returned for the process whose process ID is equal to pid.

RETURN VALUES If successful, sched_setscheduler() returns the former scheduling policy of the specified process (pid), which will be one of the following values:

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.

ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO

priocntl(1), intro(2), exec(2), priocntl(2), sched_get_priority_max(3R), sched_setparam(3R), attributes(5)

BUGS

In Solaris 2.5, these functions always return -1 and set 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  cc [ flag ...] file ... -lposix4 [ library ... ]
#include <sched.h>
int sched_yield(void);

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.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  attributes(5)
NAME scr_dump, scr_init, scr_restore, scr_set – write screen contents to/from a file

SYNOPSIS

#include <curses.h>

int scr_dump(const char *filename);
int scr_init(const char *filename);
int scr_restore(const char *filename);
int scr_set(const char *filename);

ARGUMENTS

filename Is a pointer to the file in which screen contents are written.

DESCRIPTION

These functions perform input/output functions on a screen basis.

The scr_dump() function writes the contents of the virtual screen, curscr, to filename.

The scr_restore() function reads the contents of filename from curscr (which must have been written with scr_dump()). The next refresh operation restores the screen to the way it looks in filename.

The scr_init() function reads the contents of filename and uses those contents to initialize the X/Open Curses data structures to what is actually on screen. The next refresh operation bases its updates on this data, unless the terminal has been written to since filename was saved or the terminfo capabilities rmcup and nrrmc are defined for the current terminal.

The scr_set() function combines scr_restore() and scr_init(). It informs the program that the contents of the file filename are what is currently on the screen and that the program wants those contents on the screen.

RETURN VALUES

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

ERRORS

None.

SEE ALSO
delscreen(3XC), doupdate(3XC), endwin(3XC), getwin(3XC)
**NAME**  
`scrl`, `scroll`, `wscrl` – scroll a window

**SYNOPSIS**  
```
#include <curses.h>

int scrl(int n);
int scroll(WINDOW *win);
int wscrl(WINDOW *win, int n);
```

**ARGUMENTS**  
- `n` number and direction of lines to scroll  
- `win` pointer to the window in which to scroll

**DESCRIPTION**  
The `scroll()` function scrolls the window `win` up one line. The current cursor position is not changed.

The `scrl()` and `wscrl()` functions scroll the window `stdscr` or `win` up or down `n` lines, where `n` is a positive (scroll up) or negative (scroll down) integer.

The `scrollok(3XC)` function must be enabled for these functions to work.

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

**ERRORS**  
None.

**SEE ALSO**  
`clearok(3XC)`
**NAME**

secure_rpc, authdes_getucred, authdes_seccreate, getnetname, host2netname, key_decryptsession, key_encryptsession, key_gendes, key_setsecret, key_secretkey_is_set, netname2host, netname2user, user2netname – library routines for secure remote procedure calls

**DESCRIPTION**

RPC library routines allow C programs to make procedure calls on other machines across the network.

RPC supports various authentication flavors. Among them are:

- **AUTH_NONE** (none) no authentication.
- **AUTH_SYS** Traditional UNIX-style authentication.
- **AUTH_DES** DES encryption-based authentication.
- **AUTH_KERB** Kerberos encryption-based authentication.

The `authdes_getucred()` and `authdes_seccreate()` routines implement the **AUTH_DES** authentication flavor. The keysender daemon `keyserv` (see `keyserv(1M)`) must be running for the **AUTH_DES** authentication system to work, and `keylogin(1)` must have been run. Only the **AUTH_DES** style of authentication is discussed here. For information about the **AUTH_NONE** and **AUTH_SYS** styles of authentication, refer to `rpc_clnt_auth(3N)`. For information about the **AUTH_KERB** style of authentication, refer to `kerberos_rpc(3N)`.

The routines documented on this page are MT-Safe. See the pages of the other authentication styles for their MT-level.

**Routines**

See `rpc(3N)` for the definition of the AUTH data structure.

```
#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);
```

`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.

- *uidp* is set to the user’s numerical ID associated with *adc*. *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* (see `netname2user()`, below).

Warning: `authdes_getucred()` will fail if the authdes_cred structure was created with the netname of a host. In such a case, `netname2host()` should be used on the host netname in the authdes_cred structure to get the host name.
AUTH *authdes_seccreate(const char *name, const 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. If domain is NULL and host is a NIS name like “host1.ssi.sun.com,” host2netname() uses the domain “ssi.sun.com” rather than the default domain name of the machine.

int key_decryptsession(const char *remotename, des_block *deskey);

key_decryptsession() is an interface to the keyserver daemon, which is associated with RPC’s secure authentication system (AUTH_DES authentication).
User programs rarely need to call it, or its associated routines
\texttt{key\_encryptsession()}, \texttt{key\_gendes()}, and \texttt{key\_setsecret()}.  

\texttt{key\_decryptsession()} takes a server netname \texttt{remotename} and a DES key \texttt{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
\texttt{key\_encryptsession()}.  

\begin{verbatim}
int key\_encryptsession(const char *remotename, des\_block *deskey);

des\_block *deskey;

key\_encryptsession() is a keyserver interface routine. It takes a server netname
\texttt{remotename} and a DES key \texttt{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 \texttt{key\_decryptsession()}. This routine returns 0 if it succeeds, \texttt{-1}
if it fails.
\end{verbatim}

\begin{verbatim}
int key\_gendes(des\_block *deskey);

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, \texttt{-1}
if it fails.
\end{verbatim}

\begin{verbatim}
int key\_setsecret(const char *key);

des\_block *deskey;

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, \texttt{-1}
if it fails.
\end{verbatim}

\begin{verbatim}
int key\_secretkey\_is\_set(void);

des\_block *deskey;

key\_secretkey\_is\_set() is a keyserver interface routine that may be used to deter-
mine 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.
\end{verbatim}

\begin{verbatim}
int netname2host(const char *name, char *host, const int hostlen);

char *name, *host, const int hostlen;

Convert from an operating system independent netname \texttt{name} to a domain-
specific hostname \texttt{host}. \texttt{hostlen} is the maximum size of \texttt{host}. Returns 1 if it
succeeds, and 0 if it fails. Inverse of \texttt{host2netname()}.  
\end{verbatim}

\begin{verbatim}
int netname2user(const char *name, uid\_t *uidp, gid\_t *gidp,
int *gidlenp, gid\_t gidlist[NGRPS]);

const char *name, uid\_t *uidp, gid\_t *gidp,
int *gidlenp, gid\_t gidlist[NGRPS];

Convert from an operating system independent netname to a domain-specific
user ID. Returns 1 if it succeeds, and 0 if it fails. Inverse of \texttt{user2netname()}.  
*\texttt{uidp} is set to the user’s numerical ID associated with \texttt{name}. *\texttt{gidp} is set to the
numerical ID of the user’s group. \texttt{gidlist} contains the numerical IDs of the other
groups to which the user belongs. *\texttt{gidlenp} is set to the number of valid group ID
entries in \texttt{gidlist}.
int user2netname(char name[MAXNETNAMELEN+1], const uid_t uid,
              const char *domain);

Convert from a domain-specific username to an operating system independent
netname. Returns 1 if it succeeds, and 0 if it fails. Inverse of netname2user().

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  chkey(1), keylogin(1), keyserv(1M), newkey(1M), kerberos_rpc(3N), rpc(3N), 
          rpc_clnt_auth(3N), attributes(5)
NAME  seekdir – set position of directory stream

SYNOPSIS  
#include <sys/types.h>
#include <dirent.h>

void seekdir(DIR *dirp, long int loc);

DESCRIPTION  
The seekdir() function sets the position of the next readdir(3C) operation on the directory stream specified by dirp to the position specified by loc. The value of loc should have been returned from an earlier call to telldir(3C). The new position reverts to the one associated with the directory stream when telldir() was performed.

If the value of loc was not obtained from an earlier call to telldir() or if a call to rewinddir(3C) occurred between the call to telldir() and the call to seekdir(), the results of subsequent calls to readdir() are unspecified.

RETURN VALUES  
The seekdir() function returns no value.

ERRORS  
No errors are defined.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  
opendir(3C), readdir(3C), rewinddir(3C), telldir(3C), attributes(5)
NAME

select, FD_SET, FD_CLR, FD_ISSET, FD_ZERO – synchronous I/O multiplexing

SYNOPSIS

#include <sys/time.h>

int select(int nfds, fd_set *readfds, fd_set *writefds, fd_set *errorfds,
    struct timeval *timeout);

void FD_SET(int fd, fd_set *fdset);

void FD_CLR(int fd, fd_set *fdset);

int FD_ISSET(int fd, fd_set *fdset);

void FD_ZERO(fd_set *fdset);

DESCRIPTION

The `select()` function indicates which of the specified file descriptors is ready for reading, ready for writing, or has an error condition pending. If the specified condition is false for all of the specified file descriptors, `select()` blocks, up to the specified timeout interval, until the specified condition is true for at least one of the specified file descriptors.

The `select()` function supports regular files, terminal and pseudo-terminal devices, STREAMS-based files, FIFOs and pipes. The behavior of `select()` on file descriptors that refer to other types of file is unspecified.

The `nfds` argument specifies the range of file descriptors to be tested. The `select()` function tests file descriptors in the range of 0 to `nfds`–1.

If the `readfs` argument is not a null pointer, it points to an object of type `fd_set` that on input specifies the file descriptors to be checked for being ready to read, and on output indicates which file descriptors are ready to read.

If the `writefs` argument is not a null pointer, it points to an object of type `fd_set` that on input specifies the file descriptors to be checked for being ready to write, and on output indicates which file descriptors are ready to write.

If the `errorfs` argument is not a null pointer, it points to an object of type `fd_set` that on input specifies the file descriptors to be checked for error conditions pending, and on output indicates which file descriptors have error conditions pending.

On successful completion, the objects pointed to by the `readfs`, `writefs`, and `errorfs` arguments are modified to indicate which file descriptors are ready for reading, ready for writing, or have an error condition pending, respectively. For each file descriptor less than `nfds`, the corresponding bit will be set on successful completion if it was set on input and the associated condition is true for that file descriptor.

If the `timeout` argument is not a null pointer, it points to an object of type `struct timeval` that specifies a maximum interval to wait for the selection to complete. If the `timeout` argument points to an object of type `struct timeval` whose members are 0, `select()` does not block. If the `timeout` argument is a null pointer, `select()` blocks until an event causes one of the masks to be returned with a valid (non-zero) value. If the time limit expires before any event occurs that would cause one of the masks to be set to a non-zero value, `select()` completes successfully and returns 0.
If the `readfs`, `writefs`, and `errorfds` arguments are all null pointers and the `timeout` argument is not a null pointer, `select()` blocks for the time specified, or until interrupted by a signal. If the `readfs`, `writefs`, and `errorfds` arguments are all null pointers and the `timeout` argument is a null pointer, `select()` blocks until interrupted by a signal.

File descriptors associated with regular files always select true for ready to read, ready to write, and error conditions.

On failure, the objects pointed to by the `readfs`, `writefs`, and `errorfds` arguments are not modified. If the timeout interval expires without the specified condition being true for any of the specified file descriptors, the objects pointed to by the `readfs`, `writefs`, and `errorfds` arguments have all bits set to 0.

A file descriptor for a socket that is listening for connections will indicate that it is ready for reading, when connections are available. A file descriptor for a socket that is connecting asynchronously will indicate that it is ready for writing, when a connection has been established.

Selecting true for reading on a socket descriptor upon which a `listen(3N)` call has been performed indicates that a subsequent `accept(3N)` call on that descriptor will not block.

File descriptor masks of type `fd_set` can be initialized and tested with the macros `FD_CLR()`, `FD_ISSET()`, `FD_SET()`, and `FD_ZERO()`.

The behavior of these macros is undefined if the `fd` argument is less than 0 or greater than or equal to `FD_SETSIZE`.

**RETURN VALUES**

The `FD_CLR()`, `FD_SET()`, and `FD_ZERO()` macros return no value. The `FD_ISSET()` macro returns a non-zero value if the bit for the file descriptor `fd` is set in the file descriptor set pointed to by `fdset`, and 0 otherwise.

On successful completion, `select()` returns the total number of bits set in the bit masks. Otherwise, −1 is returned, and `errno` is set to indicate the error.

**ERRORS**

Under the following conditions, `select()` fails and sets `errno` to:

- **EBADF** One or more of the file descriptor sets specified a file descriptor that is not a valid open file descriptor.
- **EINTR** The `select()` function was interrupted before any of the selected events occurred and before the timeout interval expired.
  
  If `SA_RESTART` has been set for the interrupting signal, it is implementation-dependent whether `select()` restarts or returns with `EINTR`.
- **EINVAL** An invalid timeout interval was specified.
EINVAL The **nfds** argument is less than 0, or greater than or equal to **FD_SETSIZE**.

EINVAL One of the specified file descriptors refers to a STREAM or multiplexer that is linked (directly or indirectly) downstream from a multiplexer.

EINVAL A component of the pointed-to time limit is outside the acceptable range: 
\( t_{\text{sec}} \) must be between 0 and \( 10^8 \), inclusive. \( t_{\text{usec}} \) must be greater than or equal to 0, and less than \( 10^6 \).

**USAGE** The **poll**(2) function is preferred over this function. It must be used when the number of file descriptors exceeds **FD_SETSIZE**.

The use of a timeout does not affect any pending timers set up by **alarm**(2), **ualarm**(3C) or **setitimer**(2).

On successful completion, the object pointed to by the **timeout** argument may be modified.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**SEE ALSO** **alarm**(2), **fcntl**(2), **poll**(2), **read**(2), **setitimer**(2), **write**(2), **accept**(3N), **listen**(3N),
**ualarm**(3C), **attributes**(5)

**NOTES** The default value for **FD_SETSIZE** (currently 1024) is larger than the default limit on the number of open files. It is not possible to increase the size of the **fd_set** data type when used with **select**().
NAME

semaphore, sema_init, sema_destroy, sema_wait, sema_trywait, sema_post – semaphores

SYNOPSIS

cc [ flag ... ] file ... -lt thread -lc [ library ... ]
#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);

DESCRIPTION

A semaphore is a non-negative integer count and is generally used to coordinate access to resources. The initial semaphore count is set to the number of free resources, then threads slowly increment and decrement the count as resources are added and removed. If the semaphore count drops to zero, which means no available resources, threads attempting to decrement the semaphore will block until the count is greater than zero.

Semaphores can synchronize threads in this process and other processes if they are allocated in writable memory and shared among the cooperating processes (see `mmap(2)`), and have been initialized for this purpose.

Semaphores must be initialized before use; semaphores pointed to by `sp` to `count` are initialized by `sema_init()`. `type` can assign several different types of behavior to a semaphore. No current type uses `arg` although it may be used in the future.

`type` may be one of the following:

USYNC_PROCESS The semaphore can synchronize threads in this process and other processes. Initializing the semaphore should be done by only one process. A semaphore initialized with this type must be allocated in memory shared between processes, i.e. either in Sys V shard memory (see `shmop(2)`), or in memory mapped to a file (see `mmap(2)`). It is illegal to initialize the object this way and to not allocate it in such shared memory. `arg` is ignored.

USYNC_THREAD The semaphore can synchronize threads only in this process. `arg` is ignored.

A semaphore must not be simultaneously initialized by multiple threads, nor re-initialized while in use by other threads.

Default semaphore initialization (intra-process):

```c
sema_t sp;

sema_init(&sp, NULL, NULL);
OR
sema_init(&sp, USYNC_THREAD, NULL);
```

modified 8 May 1997

SunOS 5.6

3T-1383
OR
sema_t sp = DEFAULTSEMA;

Customized semaphore initialization (inter-process):

sema_init(&sp, USYNC_PROCESS, NULL);
sema_destroy() destroys any state related to the semaphore pointed to by sp. The semaphore storage space is not released.
sema_wait() blocks the calling thread until the semaphore count pointed to by sp is greater than zero, and then it atomically decrements the count.
sema_trywait() atomically decrements the semaphore count pointed to by sp, if the count is greater than zero; otherwise, it returns an error.
sema_post() atomically increments the semaphore count pointed to by sp. If there are any threads blocked on the semaphore, one will be unblocked.

The semaphore functionality described on this man page is for the Solaris threads implementation. For the POSIX-compliant semaphore interface documentation, see sem_open(3R), sem_init(3R), sem_wait(3R), sem_post(3R), sem_getvalue(3R), sem_unlink(3R), sem_close(3R), sem_destroy(3R)).

RETURN VALUES
Upon successful completion, 0 is returned; otherwise, a non-zero value indicates an error.

ERRORS
These functions fail and return the corresponding value if any of the following conditions are detected:
EINVAL Invalid argument.
EFAULT sp or arg points to an illegal address.

sema_wait() fails and returns the corresponding value if any of the following conditions are detected:
EINTR The wait was interrupted by a signal or fork().
EBUSY The semaphore pointed to by sp has a zero count.

EXAMPLES
The customer waiting-line in a bank is analogous to the synchronization scheme of a semaphore using sema_wait() and sema_trywait():

/* cc [ flag ... ] file ... -lthread [ library ... ] */
#include <errno.h>
define TELLERS 10
sema_t tellers; /* semaphore */
int banking_hours(), deposit_withdrawal;
void *customer(), do_business(), skip_banking_today();
...
sem_init(&tellers, TELLERS, USYNC_THREAD, NULL);
/* 10 tellers available */
while(banking_hours())
    pthread_create(NULL, NULL, customer, deposit_withdrawal);
...

void *
customer(int deposit_withdrawal)
{
    int this_customer, in_a_hurry = 50;
    this_customer = rand() % 100;
    if (this_customer == in_a_hurry) {
        if (sema_trywait(&tellers) != 0)
            if (errno == EAGAIN) { /* no teller available */
                skip_banking_today(this_customer);
                return;
            } /* else go immediately to available teller & decrement tellers */
    } else
        sema_wait(&tellers); /* wait for next teller, then proceed, and decrement tellers */
    do_business(deposit_withdrawal);
    sema_post(&tellers); /* increment tellers; this_customer's teller is now available */
}

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO mmap(2), sem_open(3R), sem_init(3R), sem_wait(3R), sem_post(3R), sem_getvalue(3R), sem_unlink(3R), sem_close(3R), sem_destroy(3R), attributes(5), standards(5)

NOTES These interfaces are also available via:
#include <thread.h>
If multiple threads are waiting for a semaphore, by default, there is no defined order of unblocking.

modified 8 May 1997 SunOS 5.6 3T-1385
NAME
sem_close – close a named semaphore

SYNOPSIS
cc [ flag ...] file ... -lposix4 [ library ...]
#include <semaphore.h>
int sem_close(sem_t *sem);

DESCRIPTION
The sem_close() function is used to indicate that the calling process is finished using the named semaphore sem. It 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 name 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.

The sem_close() function should not be called for an unnamed semaphore initialized by sem_init(3R).

RETURN VALUES
If successful, sem_close() returns 0, otherwise it returns −1 and sets errno to indicate the error condition.

ERRORS
EINVAL
The sem argument is not a valid semaphore descriptor.
ENOSYS
The sem_close() function is not supported by this implementation.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
sem_init(3R), sem_open(3R), sem_unlink(3R), attributes(5)
NAME  sem_destroy – destroy an unnamed semaphore

SYNOPSIS  cc [ flag ... ] file ... -lposix4 [ library ... ]
#include <semaphore.h>
int sem_destroy(sem_t *sem);

DESCRIPTION  The sem_destroy() function is used to destroy the unnamed semaphore, sem, 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  The sem argument is not a valid semaphore.
ENOSYS  The sem_destroy() function is not supported.
EBUSY  Other processes (or LWPs or threads) are currently blocked on the semaphore.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  sem_init(3R), sem_open(3R), attributes(5)
NAME
sem_getvalue – get the value of a semaphore

SYNOPSIS
cc [ flag ...] file ... -lposix4 [ library ...]
#include <semaphore.h>
int sem_getvalue(sem_t *sem, int *sval);

DESCRIPTION
The sem_getvalue() function updates the location referenced by sval to have the value of
the semaphore referenced by sem without affecting the state of the semaphore. The
updated value represents an actual semaphore value that occurred at some unspecified
time during the call to sem_getvalue(), but may not be the actual value of the semaphore
when sem_getvalue() is returned to the caller.
The value set in sval may be 0 or positive. If sval is 0, there may be other processes (or
LWPs or threads) waiting for the semaphore; if sval is positive, no one is waiting.

RETURN VALUES
If successful, sem_getvalue() returns 0, otherwise, it returns −1, and sets errno to indicate
the error condition.

ERRORS
EINVAL
The sem argument does not refer to a valid semaphore.
ENOSYS
The sem_getvalue() function is not supported.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
sem_post(3R), sem_wait(3R), attributes(5)
NAME  sem_init – initialize an unnamed semaphore

SYNOPSIS  cc [ flag ... ] file ... -lpthread [ library ... ]
#include <semaphore.h>
int sem_init(sem_t *sem, int pshared, unsigned int value);

DESCRIPTION  The sem_init() function is used to initialize the unnamed semaphore referred to by sem to value. 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.

If pshared is non-zero, then the semaphore is sharable between processes. If the semaphore is not being shared between processes, the application should set pshared to 0.

RETURN VALUES  If successful, sem_init() returns 0 and initializes the semaphore in sem; otherwise it returns -1 and sets errno to indicate the error condition.

ERRORS  EINVAL  The value argument exceeds SEM_VALUE_MAX.
ENOSPC  A resource required to initialize the semaphore has been exhausted, or the resources have reached the limit on semaphores (SEM_NSEMS_MAX).
ENOSYS  The sem_init() function is not supported.
EPERM  The calling process lacks the appropriate privileges to initialize the semaphore.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  sem_destroy(3R), sem_post(3R), sem_wait(3R), attributes(5)
NAME
sem_open – initialize/open a named semaphore

SYNOPSIS
c
cc [flag . . . ] file . . . -lposix4 [library . . . ]
#include <semaphore.h>
sem_t *sem_open(const char *name, int oflag,
                /* unsigned long mode, unsigned int value */ ...);

DESCRIPTION
The sem_open() function establishes a connection to a semaphore, name, 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.

The name argument points to a string naming a semaphore object. The name 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 name, 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. The first character of name must be a slash
( /) character and the remaining characters of name cannot include any slash characters.

For maximum portability, name should include no more than 14 characters, but this limit
is not enforced.

The oflag argument determines whether the semaphore is created or merely accessed by
the call to sem_open(). The three valid values for oflag are 0, O_CREAT, or the bitwise
inclusive OR of O_CREAT and O_EXCL. Setting the oflag 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 name 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 name, and nobits set in oflag.

Using the O_CREAT flag requires a third and a fourth argument: mode and value. The
semaphore is created with an initial count of value. value must be less than or equal to
SEM_VALUE_MAX. The semaphore’s user ID acquires the effective user ID of the pro-
cess; 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 mode,
modified by clearing all bits set in the file creation mask of the process (see umask(2)).

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    The sem_open() function was interrupted by a signal.
EINV AL  The name argument is not a valid name, or O_CREAT was set in oflag and value is greater than SEM_VALUE_MAX.

EMFILE The number of open semaphore descriptors in this process exceeds SEM_NSEMS_MAX.

ENAMETOOLONG The number of open file descriptors in this process exceeds OPEN_MAX.

ENFILE The string-length of name exceeds PATH_MAX, or a pathname component is longer than NAME_MAX while _POSIX_NO_TRUNC is in effect.

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 The sem_open() function is not supported.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO exec(2), exit(2), umask(2), sem_close(3R), sem_post(3R), sem_unlink(3R), sem_wait(3R), sysconf(3C), attributes(5)

modified 30 Dec 1996

SunOS 5.6

3R-1391
NAME  sem_post – increment the count of a semaphore

SYNOPSIS  cc [ flag ... ] file ... -lposix4 [ library ... ]
#include <semaphore.h>

int sem_post(sem_t *sem); If prior to the call to sem_post() the value of sem 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.

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.

The sem_post() function is reentrant with respect to signals (ASYNC-SAFE), and may be invoked from a signal-catching function. The semaphore functionality described on this man page is for the POSIX (see standards(5)) threads implementation. For the documentation of the Solaris threads interface, see semaphore(3T).

RETURN VALUES  If successful, sem_post() returns 0; otherwise it returns −1, and sets errno to indicate the error condition.

ERRORS  EINVAL  sem does not refer to a valid semaphore.
ENOSYS  sem_post() is not supported by this implementation.

EXAMPLES  (see sem_wait(3R))

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Async-Signal-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  sched_setscheduler(3R), sem_wait(3R), semaphore(3T), attributes(5), standards(5)

NOTES  The sem_wait(3R) and sem_trywait(3R) functions decrement the semaphore upon their successful return.

BUGS  In Solaris 2.5, 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  cc [ flag ... ] file ... -lposix4 [ library ... ]
#include <semaphore.h>
int sem_unlink(const char *name);

DESCRIPTION  sem_unlink( ) removes the semaphore named by the string name. If the semaphore, name, is currently referenced by other processes, then sem_unlink( ) has no effect on the state of the semaphore. If one or more processes have the semaphore open when sem_unlink( ) is called, destruction of the semaphore is postponed until all references to the semaphore have been destroyed by calls to sem_close(3R), exit(2), or 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.

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  EACCES  Permission is denied to unlink the named semaphore.
ENAMETOOLONG  The string-length of name exceeds PATH_MAX, or a pathname component is longer than NAME_MAX while _POSIX_NO_TRUNC is in effect.
ENOENT  The named semaphore does not exist.
ENOSYS  sem_unlink( ) is not supported by this implementation.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  exec(2), exit(2), sem_close(3R), sem_open(3R), attributes(5)
NAME
sem_wait, sem_trywait – acquire or wait for a semaphore

SYNOPSIS
cc [ flag . . . ] file . . . −lposix4 [ library . . . ]
#include <semaphore.h>
int sem_wait(sem_t *sem);
int sem_trywait(sem_t *sem);

DESCRIPTION
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.

If at the time of a call to either sem_wait() or sem_trywait(), the value of sem is positive,
these functions decrement the value of the semaphore, return immediately, and allow
the calling process to continue.

If the semaphore’s value is 0:

sem_wait() blocks, awaiting the semaphore to be released by another process (or
LWP or thread).

sem_trywait() fails, returning immediately.

RETURN VALUES
If at the time of a call to either sem_wait() or sem_trywait(), the value of sem is positive,
these functions return 0 on success. If the call was unsuccessfull, the state of the sema-
phore is unchanged, the calling function returns −1, and sets errno to indicate the error
condition.

ERRORS
EAGAIN The value of sem was 0 when sem_trywait() was called.
EINVAL sem does not refer to a valid semaphore.
EINTR sem_wait() was interrupted by a signal.
ENOSYS sem_wait() and sem_trywait() are not supported by this implementation.
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.

EXAMPLES
The customer waiting-line in a bank may be analogous to the synchronization scheme of
a semaphore utilizing sem_wait() and sem_trywait():
/
#include <errno.h>
define TELLERS 10
sem_t bank_line; /* semaphore */
int banking_hours(), deposit_withdrawal;
void *customer0, do_business(), skip_banking_today();
thread_t tid;
/

3R-1394 SunOS 5.6 modified 30 Dec 1996
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 */
}

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

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

SEE ALSO sem_post(3R), attributes(5)

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.
NAME send, sendto, sendmsg – send a message from a socket

SYNOPSIS

cc [ flag ...] file ... -lssocket -lnsl [ library ...]

#include <sys/types.h>
#include <sys/socket.h>

int send(int s, const char *msg, int len, int flags);
int sendto(int s, const char *msg, int len, int flags, const struct sockaddr *to, int tolen);
int sendmsg(int s, const struct msghdr *msg, int flags);

DESCRIPTION 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 connected state, while sendto() and sendmsg() may be used at any time. s is a socket created with socket(3N).

The address of the target is given by to with tolen specifying its size. The length of the message is given by len. 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.

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.

The flags parameter is formed from the bitwise OR of zero or more of the following:

MSG_OOB Send “out-of-band” data on sockets that support this notion. The underlying protocol must also support “out-of-band” data. Only SOCK_STREAM sockets created in the AF_INET address family support out-of-band data.

MSG_DONTROUTE The SO_DONTROUTE option is turned on for the duration of the operation. It is used only by diagnostic or routing programs. See recv(3N) for a description of the msghdr structure.

RETURN VALUES These calls return the number of bytes sent, or −1 if an error occurred.

ERRORS The calls fail if:

EBADF s is an invalid file descriptor.
EINTR The operation was interrupted by delivery of a signal before any data could be buffered to be sent.
EINVAL tolen is not the size of a valid address for the specified address family.
EMSGSIZE The socket requires that message be sent atomically, and the
message was too long.

ENOMEM  There was insufficient memory available to complete the operation.
ENOSR   There were insufficient STREAMS resources available for the operation to complete.
ENOTSOCK s is not a socket.
EWOULDBLOCK The socket is marked non-blocking and the requested operation would block.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO fcntl(2), poll(2), write(2), connect(3N), getsockopt(3N), recv(3N), select(3C), socket(3N), attributes(5), socket(5)
NAME
send – send a message on a socket

SYNOPSIS
cc [ flag ...] file ... -lxnet [ library ...]
#include <sys/socket.h>
ssize_t send(int socket, const void *buffer, size_t length, int flags);

DESCRIPTION
The send() function initiates transmission of a message from the specified socket to its
peer. The send() function sends a message only when the socket is connected. This func-
tion takes the following arguments:
socket      Specifies the socket file descriptor.
buffer      Points to the buffer containing the message to send.
length      Specifies the length of the message in bytes.
flags       Specifies the type of message transmission. Values of this argument are
formed by logically OR’ing zero or more of the following flags:
MSG_EOR     Terminates a record (if supported by the protocol)
MSG_OOB     Sends out-of-band data on sockets that support out-of-
            band communications. The significance and semantics
            of out-of-band data are protocol-specific.

The length of the message to be sent is specified by the length argument. If the message is
too long to pass through the underlying protocol, send() fails and no data is transmitted.
Successful completion of a call to send() does not guarantee delivery of the message. A
return value of −1 indicates only locally-detected errors.
If space is not available at the sending socket to hold the message to be transmitted and
the socket file descriptor does not have O_NONBLOCK set, send() blocks until space is
available. If space is not available at the sending socket to hold the message to be
transmitted and the socket file descriptor does have O_NONBLOCK set, send() will fail.
The select(3C) and poll(2) functions can be used to determine when it is possible to send
more data.

RETURN VALUES
Upon successful completion, send() returns the number of bytes sent. Otherwise, −1 is
returned and errno is set to indicate the error.

ERRORS
The send() function will fail if:
EBADF       The socket argument is not a valid file descriptor.
ECONNRESET  A connection was forcibly closed by a peer.
EDESTADDRREQ The socket is not connection-mode and no peer address is set.
EINTR       A signal interrupted send() before any data was transmitted.
EMSGSIZE    The message is too large to be sent all at once, as the socket requires.
ENOTCONN    The socket is not connected or otherwise has not had the peer
ENOTSOCK The socket argument does not refer to a socket.
EOPNOTSUPP The socket argument is associated with a socket that does not support
one or more of the values set in flags.
EPIPE The socket is shut down for writing, or the socket is connection-mode
and the peer is closed or shut down for reading. In the latter case, and if
the socket is of type SOCK_STREAM, the SIGPIPE signal is generated to
the calling process.
EAGAIN The socket’s file descriptor is marked O_NONBLOCK and the requested
operation would block. The send() function may fail if:
ENETDOWN The local interface used to reach the destination is down.
ENETUNREACH No route to the network is present.
ENOBUFS Insufficient resources were available in the system to perform the opera-
tion.
ENOSR There were insufficient STREAMS resources available for the operation to
complete.
EIO An I/O error occurred while reading from or writing to the file system.

USAGE The send() function is identical to sendto(3XN) with a null pointer dest_len argument,
and to write(2) if no flags are used.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO poll(2), connect(3XN), getsockopt(3XN), recv(3XN), recvfrom(3XN), recvmsg(3XN),
select(3C), sendmsg(3XN), sendto(3XN), setsockopt(3XN), shutdown(3XN),
socket(3XN), attributes(5), socket(5)
NAME
sendmsg – send a message on a socket using a message structure

SYNOPSIS
cc [ flag ...] file ... -lxnet [ library ...]
#include <sys/socket.h>
ssize_t sendmsg(int socket, const struct msghdr *message, int flags);

DESCRIPTION
The sendmsg() function sends a message through a connection-mode or connectionless-mode socket. If the socket is connectionless-mode, the message will be sent to the address specified by msghdr. If the socket is connection-mode, the destination address in msghdr is ignored.

The function takes the following arguments:
socket
Specifies the socket file descriptor.
message
Points to a msghdr structure, containing both the destination address and the buffers for the outgoing message. The length and format of the address depend on the address family of the socket. The msg_flags member is ignored.
flags
Specifies the type of message transmission. Values of this argument are formed by logically OR’ing zero or more of the following flags:
MSG_EOR Terminates a record (if supported by the protocol)
MSG_OOB Sends out-of-band data on sockets that support out-of-bound data. The significance and semantics of out-of-bound data are protocol-specific.

Successful completion of a call to sendmsg() does not guarantee delivery of the message. A return value of −1 indicates only locally-detected errors.

If space is not available at the sending socket to hold the message to be transmitted and the socket file descriptor does not have O_NONBLOCK set, sendmsg() function blocks until space is available. If space is not available at the sending socket to hold the message to be transmitted and the socket file descriptor does have O_NONBLOCK set, sendmsg() function will fail.

If the socket protocol supports broadcast and the specified address is a broadcast address for the socket protocol, sendmsg() will fail if the SO_BROADCAST option is not set for the socket.

RETURN VALUES
Upon successful completion, sendmsg() function returns the number of bytes sent. Otherwise, −1 is returned and errno is set to indicate the error.

ERRORS
The sendmsg() function will fail if:
EAFNOSUPPORT Addresses in the specified address family cannot be used with this socket.
EBADF The socket argument is not a valid file descriptor.
The sendmsg() function may fail if:

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EACCESS</td>
<td>Search permission is denied for a component of the path prefix; or write access to the named socket is denied.</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>ELOOP</td>
<td>Too many symbolic links were encountered in translating the pathname in the socket address.</td>
</tr>
<tr>
<td>ENAMETOOLONG</td>
<td>A component of a pathname exceeded NAME_MAX characters, or an entire pathname exceeded PATH_MAX characters.</td>
</tr>
<tr>
<td>ENOENT</td>
<td>A component of the pathname does not name an existing file or the pathname is an empty string.</td>
</tr>
<tr>
<td>ENOTDIR</td>
<td>A component of the path prefix of the pathname in the socket address is not a directory.</td>
</tr>
<tr>
<td>EDEADCASE</td>
<td>The socket is not connection-mode and does not have its peer address set, and no destination address was specified.</td>
</tr>
<tr>
<td>EHOSTUNREACH</td>
<td>The destination host cannot be reached (probably because the host is down or a remote router cannot reach it).</td>
</tr>
<tr>
<td>EINVAL</td>
<td>The msg_iovlen member of the msghdr structure pointed to by msg is less than or equal to 0, or is greater than IOV_MAX.</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>EISCONN</td>
<td>A destination address was specified and the socket is connection-mode and is already connected.</td>
</tr>
</tbody>
</table>
ENETDOWN  The local interface used to reach the destination is down.
ENETUNREACH  No route to the network is present.
ENOBUFFS  Insufficient resources were available in the system to perform the operation.
ENOMEM  Insufficient memory was available to fulfill the request.
ENOSR  There were insufficient STREAMS resources available for the operation to complete.

If the address family of the socket is AF_UNIX, then sendmsg() may fail if:
ENAMETOOLONG  Pathname resolution of a symbolic link produced an intermediate result whose length exceeds PATH_MAX.

**USAGE**  The select(3C) and poll(2) functions can be used to determine when it is possible to send more data.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**SEE ALSO**  poll(2), getsockopt(3XN), recv(3XN), recvfrom(3XN), recvmsg(3XN), select(3C), send(3XN), sendto(3XN), setsockopt(3XN), shutdown(3XN), socket(3XN), attributes(5), socket(5)
NAME
sendto – send a message on a socket

SYNOPSIS
cc [ flag ...] file ... -lxnet [ library ...]
#include <sys/socket.h>
ssize_t sendto(int socket, const void *message, size_t length, int flags,
    const struct sockaddr *dest_addr, size_t dest_len);

DESCRIPTION
The sendto() function sends a message through a connection-mode or connectionless-mode socket. If the socket is connectionless-mode, the message will be sent to the address specified by dest_addr. If the socket is connection-mode, dest_addr is ignored.

The function takes the following arguments:

- **socket**: Specifies the socket file descriptor.
- **message**: Points to a buffer containing the message to be sent.
- **length**: Specifies the size of the message in bytes.
- **flags**: Specifies the type of message transmission. Values of this argument are formed by logically OR'ing zero or more of the following flags:
  - **MSG_EOR**: Terminates a record (if supported by the protocol)
  - **MSG_OOB**: Sends out-of-band data on sockets that support out-of-band data. The significance and semantics of out-of-band data are protocol-specific.
- **dest_addr**: Points to a sockaddr structure containing the destination address. The length and format of the address depend on the address family of the socket.
- **dest_len**: Specifies the length of the sockaddr structure pointed to by the dest_addr argument.

If the socket protocol supports broadcast and the specified address is a broadcast address for the socket protocol, sendto() will fail if the SO_BROADCAST option is not set for the socket.

The dest_addr argument specifies the address of the target. The length argument specifies the length of the message.

Successful completion of a call to sendto() does not guarantee delivery of the message. A return value of -1 indicates only locally-detected errors.

If space is not available at the sending socket to hold the message to be transmitted and the socket file descriptor does not have O_NONBLOCK set, sendto() blocks until space is available. If space is not available at the sending socket to hold the message to be transmitted and the socket file descriptor does have O_NONBLOCK set, sendto() will fail.

RETURN VALUES
Upon successful completion, sendto() returns the number of bytes sent. Otherwise, -1 is returned and errno is set to indicate the error.

modified 16 May 1997 SunOS 5.6 3XN-1403
The `sendto()` function will fail if:

**EAFNOSUPPORT**
Addresses in the specified address family cannot be used with this socket.

**EBADF**
The socket argument is not a valid file descriptor.

**ECONNRESET**
A connection was forcibly closed by a peer.

**EINVAL**
The `dest_len` argument is not a valid length for the address family.

**EIO**
An I/O error occurred while reading from or writing to the file system.

**ENAMETOOLONG**
A component of a pathname exceeded NAME_MAX characters, or an entire pathname exceeded PATH_MAX characters.

**ENOENT**
A component of the pathname does not name an existing file or the pathname is an empty string.

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

**ENOTCONN**
The socket is connection-mode but is not connected.

**ENOTSOCK**
The socket argument does not refer to a socket.

**EOPNOTSUPP**
The socket argument is associated with a socket that does not support one or more of the values set in `flags`.

**EPIPE**
The socket is shut down for writing, or the socket is connection-mode and the peer is closed or shut down for reading. In the latter case, and if the socket is of type `SOCK_STREAM`, the SIGPIPE signal is generated to the calling process.

**EAGAIN**
The socket’s file descriptor is marked O_NONBLOCK and the requested operation would block.

If the address family of the socket is `AF_UNIX`, then `sendto()` will fail if:

**EACCES**
Search permission is denied for a component of the path prefix; or write access to the named socket is denied.

**ELOOP**
Too many symbolic links were encountered in translating the pathname in the socket address.

**ENOENT**
A component of the pathname does not name an existing file or the pathname is an empty string.

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

**EDESTADDRREQ**
The socket is not connection-mode and does not have its peer address set, and no destination address was specified.

**EHOSTUNREACH**
The destination host cannot be reached (probably because the host is down or a remote router cannot reach it).

**EINVAL**
The `dest_len` argument is not a valid length for the address family.
EIO
An I/O error occurred while reading from or writing to the file system.

EISCONN
A destination address was specified and the socket is connection-mode and is already connected.

ENETDOWN
The local interface used to reach the destination is down.

ENETUNREACH
No route to the network is present.

ENOBUS
Insufficient resources were available in the system to perform the operation.

ENOMEM
Insufficient memory was available to fulfill the request.

ENOSR
There were insufficient STREAMS resources available for the operation to complete.

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

If the address family of the socket is AF_UNIX, then sendto() may fail if:

USAGE
The select(3C) and poll(2) functions can be used to determine when it is possible to send more data.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
poll(2), getsockopt(3XN), recv(3XN), recvfrom(3XN), recvmsg(3XN), select(3C), send(3XN), sendmsg(3XN), setsockopt(3XN), shutdown(3XN), socket(3XN), attributes(5), socket(5)
NAME
setbuf, setvbuf – assign buffering to a stream

SYNOPSIS
#include <stdio.h>
void setbuf(FILE *stream, char *buf);
int setvbuf(FILE *stream, char *buf, int type, size_t size);

DESCRIPTION
setbuf() may be used after a stream (see intro(3)) has been opened but before it is read or written. It causes the array pointed to by buf to be used instead of an automatically allocated buffer. If buf is the NULL pointer input/output will be completely unbuffered. The constant BUFSIZ, defined in the <stdio.h> header, indicates how large the array pointed to by buf should be.

cchar buf[BUFSIZ];

setvbuf() may be used after a stream has been opened but before it is read or written. type determines how stream will be buffered. Legal values for type (defined in <stdio.h>) are:

_IOFBF causes input/output to be fully buffered.
_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.
_IONBF causes input/output to be completely unbuffered.

If buf is not the NULL pointer, the array it points to will be used for buffering, instead of an automatically allocated buffer. size specifies the size of the buffer to be used. If input/output is unbuffered, buf and size are ignored.

For a further discussion of buffering, see stdio(3).

RETURN VALUES
If an illegal value for type is provided, setvbuf() returns a non-zero value. Otherwise, it returns zero.

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
fopen(3S), getc(3S), malloc(3C), putc(3S), stdio(3S), attributes(5)

NOTES
A common source of error is allocating buffer space as an “automatic” variable in a code block, and then failing to close the stream in the same block.

When using setbuf(), buf should always be sized using BUFSIZ. If the array pointed to by buf is larger than BUFSIZ, a portion of buf will not be used. If buf is smaller than BUFSIZ, other memory may be unexpectedly overwritten.
Parts of buf will be used for internal bookkeeping of the stream and, therefore, buf will contain less than size bytes when full. It is recommended that stdio(3S) be used to handle buffer allocation when using setvbuf().
NAME
setbuffer, setlinebuf – assign buffering to a stream

SYNOPSIS
#include <stdio.h>

void setbuffer(FILE *iop, char *abuf, size_t asize);
int setlinebuf(FILE *iop);

DESCRIPTION
The setbuffer() and setlinebuf() functions assign buffering to a stream. The three types
of buffering available are unbuffered, block buffered, and line buffered. When an output
stream is unbuffered, information appears on the destination file or terminal as soon as
written; when it is block buffered, many characters are saved and written as a block;
when it is line buffered, characters are saved until either a NEWLINE is encountered or
input is read from stdin. The fflush(3S) function may be used to force the block out
early. Normally all files are block buffered. A buffer is obtained from malloc(3C) upon
the first getc(3S) or putc(3S) performed on the file. If the standard stream stdout refers to
a terminal, it is line buffered. The standard stream stderr is unbuffered by default.

The setbuffer() function can be used after a stream iop has been opened but before it is
read or written. It uses the character array abuf whose size is determined by the asize
argument instead of an automatically allocated buffer. If abuf is the null pointer,
input/output will be completely unbuffered. A manifest constant BUFSIZ, defined in the
<stdio.h> header, tells how large an array is needed:

    char buf[BUFSIZ];

The setlinebuf() function is used to change the buffering on a stream from block buf-
fered or unbuffered to line buffered. Unlike setbuffer(), it can be used at any time that
the stream iop is active.

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 unbuf-
fered by using freopen(3S) followed by setbuf(3S) with a buffer argument of NULL.

RETURN VALUES
The setlinebuf() function returns no useful value.

SEE ALSO
malloc(3C), fclose(3S), fopen(3S), fread(3S), getc(3S), printf(3S), putc(3S), puts(3S),
setbuf(3S), setvbuf(3S)

NOTES
A common source of error is allocating buffer space as an “automatic” variable in a code
block, and then failing to close the stream in the same block.
NAME
setcat – define default catalog

SYNOPSIS
#include <pfmt.h>
char *setcat(const char *catalog);

DESCRIPTION
The routine setcat() defines the default message catalog to be used by subsequent calls to pfmt(), pfmt() or gettext() which do not explicitely specify a message catalog.
catalog 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.
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.
If no default catalog is specified, or if catalog is an invalid catalog name, Subsequent calls to gettext(), pfmt() or Ifmt() that do not explicitely specify a catalog name will use Message not found!!\n as default string.

RETURN VALUE
Upon success, setcat() returns a pointer to the catalog name. Upon failure, setcat() returns a NULL pointer.

EXAMPLE
setcat("test");
gettext(":10", "hello world\n")

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-safe</td>
</tr>
</tbody>
</table>

SEE ALSO
gettext(3C), Ifmt(3C), pfmt(3C), setlocale(3C), attributes(5), environ(5)
NAME
setcchar – set a cchar_t type character from a wide character and rendition

SYNOPSIS
#include <curses.h>
int setcchar(cchar_t ∗wcval, const wchar_t ∗wch,
const attr_t attrs, short color_pair,
const void ∗opts);

ARGUMENTS
wcval Is a pointer to a location where a cchar_t character (and its rendition) can be stored.
wch Is a pointer to a wide character.
attrs Is the set of attributes to apply to wch in creating wcval.
color_pair Is the color pair to apply to wch in creating wcval.
opts Is reserved for future use. Currently, this must be a null pointer.

DESCRIPTION
The setcchar() function takes the wide character pointed to by wch, combines it with the attributes indicated by attrs and the color pair indicated by color_pair and stores the result in the object pointed to by wcval.

RETURN VALUES
On success, the setcchar() function returns OK. Otherwise, it returns ERR.

ERRORS
None.

SEE ALSO
attroff(3XC), can_change_color(3XC), getchar(3XC)
NAME

setjmp, longjmp, _setjmp, _longjmp – non-local goto

SYNOPSIS

/usr/ucb/cc [ flag ... ] file ...
#include <setjmp.h>
int setjmp(env)
jmp_buf env;
void longjmp(env, val)
jmp_buf env;
int val;
int _setjmp(env)
jmp_buf env;
void _longjmp(env, val)
jmp_buf env;
int val;

DESCRIPTION

setjmp() and longjmp() are useful for dealing with errors and interrupts encountered in
a low-level subroutine of a program.

setjmp() saves its stack environment in env 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).

longjmp() restores the environment saved by the last call of setjmp(), and then returns
in such a way that execution continues as if the call of setjmp() had just returned the
value val to the function that invoked setjmp(); however, if val were zero, execution
would continue as if the call of setjmp() had returned one. This ensures that a “return”
from setjmp() caused by a call to longjmp() can be distinguished from a regular return
from setjmp(). The calling function must not itself have returned in the interim, other-
wise 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, vari-
ables 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 pro-
grammers trying to write machine-independent C routines.

setjmp() and longjmp() save and restore the signal mask while _setjmp() and
_longjmp() manipulate only the C stack and registers.

None of these functions save or restore any floating-point status or control registers.
EXAMPLES

The following example uses both `setjmp()` and `longjmp()` to return the flow of control to the appropriate instruction block:

```c
#include <stdio.h>
#include <setjmp.h>
#include <signal.h>
#include <unistd.h>
jmp_buf env; static void signal_handler();

main() {
    int returned_from_longjump, processing = 1;
    unsigned int time_interval = 4;
    if ((returned_from_longjump = setjmp(env)) != 0)
        switch (returned_from_longjump) {
            case SIGINT:
                printf("longjumped from interrupt %d\n",SIGINT);
                break;
            case SIGALRM:
                printf("longjumped from alarm %d\n",SIGALRM);
                break;
        }
    (void) signal(SIGINT, signal_handler);
    (void) signal(SIGALRM, signal_handler);
    alarm(time_interval);
    while (processing) {
        printf(" waiting for you to INTERRUPT (cntrl-C) ...
");
        sleep(1);
    } /* 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);
    }
}
```

SunOS 5.6 modified 7 Apr 1993
When this example is compiled and executed, and the user sends an interrupt signal, the output will be:

```
longjumped from interrupt
```

Additionally, every 4 seconds the alarm will expire, signalling this process, and the output will be:

```
longjumped from alarm
```

**SEE ALSO**
c(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  

```c
#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);
```

DESCRIPTION  These functions are useful for dealing with errors and interrupts encountered in a low-level subroutine of a program.

**setjmp()** saves its stack environment in `env` for later use by **longjmp()**.

**sigsetjmp()** saves the calling process’s registers and stack environment (see `sigaltstack(2)`) in `env` for later use by **siglongjmp()**. If `savemask` is non-zero, the calling process’s signal mask (see `sigprocmask(2)`) and scheduling parameters (see `priocntl(2)`) are also saved.

**longjmp()** restores the environment saved by the last call of **setjmp()** with the corresponding `env` argument. After **longjmp()** is completed, program execution continues as if the corresponding call of **setjmp()** had just returned the value `val`. 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).

**siglongjmp()** restores the environment saved by the last call of **sigsetjmp()** with the corresponding `env` argument. After **siglongjmp()** is completed, program execution continues as if the corresponding call of **sigsetjmp()** had just returned the value `val`. **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.

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.

The function **siglongjmp()** restores the saved signal mask if and only if the `env` argument was initialized by a call to the **sigsetjmp()** function with a non-zero `savemask` argument.

The values of register and automatic variables are undefined. Register or automatic variables whose value must be relied upon must be declared as **volatile**.

3C-1414  SunOS 5.6  modified 29 Dec 1996
EXAMPLES

The following example uses both `setjmp()` and `longjmp()` to return the flow of control to the appropriate instruction block:

```c
#include <stdio.h>
#include <setjmp.h>
#include <signal.h>
#include <unistd.h>
jmp_buf env; static void signal_handler();

main() {
    int returned_from_longjump, processing = 1;
    unsigned int time_interval = 4;
    if ((returned_from_longjump = setjmp(env)) != 0)
        switch (returned_from_longjump) {
            case SIGINT:
                printf("longjumped from interrupt \%d\n",SIGINT);
                break;
            case SIGALRM:
                printf("longjumped from alarm \%d\n",SIGALRM);
                break;
        }
    (void) signal(SIGINT, signal_handler);
    (void) signal(SIGALRM, signal_handler);
    alarm(time_interval);
    while (processing) {
        printf(" waiting for you to INTERRUPT (cntrl-C) ...\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);
    }
}
```

modified 29 Dec 1996

SunOS 5.6

3C-1415
When this example is compiled and executed, and the user sends an interrupt signal, the output will be:

```
longjumped from interrupt
```

Additionally, every 4 seconds the alarm will expire, signalling this process, and the output will be:

```
longjumped from alarm
```

## 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.

## ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

## SEE ALSO

`getcontext(2)`, `priocntl(2)`, `sigaction(2)`, `sigaltstack(2)`, `sigprocmask(2)`, `signal(3C)`, `attributes(5)`

## WARNINGS

If `longjmp()` or `siglongjmp()` are called even though `env` was never primed by a call to `setjmp()` or `sigsetjmp()`, or when the last such call was in a function that has since returned, absolute chaos is guaranteed.
NAME

setkey – set encoding key

SYNOPSIS

#include <stdlib.h>

void setkey(const char *key);

DESCRIPTION

The setkey() function provides (rather primitive) access to the hashing algorithm employed by the crypt(3C) function. The argument of setkey() is an array of length 64 bytes containing only the bytes with numerical value of 0 and 1. If this string is divided into groups of 8, the low-order bit in each group is ignored; this gives a 56-bit key which is used by the algorithm. This is the key that will be used with the algorithm to encode a string block passed to encrypt(3C).

RETURN VALUES

No values are returned.

ERRORS

The setkey() function will fail if:

ENOSYS The functionality is not supported on this implementation.

USAGE

In some environments, decoding may not be implemented. This is related to U.S. Government restrictions on encryption and decryption routines: the DES decryption algorithm cannot be exported outside the U.S.A. Historical practice has been to ship a different version of the encryption library without the decryption feature in the routines supplied. Thus the exported version of encrypt() does encoding but not decoding. Because setkey() does not return a value, applications wishing to check for errors should set errno to 0, call setkey(), then test errno and, if it is non-zero, assume an error has occurred.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO

crypt(3C), encrypt(3C), attributes(5)
NAME

setlabel – define the label for pfmt() and lfmt()

DESCRIPTION

The routine setlabel() defines the label for messages produced in standard format by subsequent calls to pfmt() and lfmt().

label is a character string no more than 25 characters in length.

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.

RETURN VALUE

setlabel() returns 0 in case of success, non-zero otherwise.

EXAMPLE

The following code (without previous call to setlabel()):

    pfmt(stderr, MM_ERROR, "test:2:Cannot open file\n");
    setlabel("UX:test");
    pfmt(stderr, MM_ERROR, "test:2:Cannot open file\n");

will produce the following output:

ERROR: Cannot open file
UX:test: ERROR: Cannot open file

USAGE

The label should be set once at the beginning of a utility and remain constant.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-safe</td>
</tr>
</tbody>
</table>

SEE ALSO

gopt(3C), lfmt(3C), pfmt(3C), attributes(5)
NAME
setlocale – modify and query a program’s locale

SYNOPSIS
#include <locale.h>

char *setlocale(int category, const char *locale);

DESCRIPTION
setlocale() selects the appropriate piece of the program’s locale as specified by the
category and locale arguments. The category 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.

LC_CTYPE affects the behavior of character handling functions such as isdigit(3C) and
tolower(3C), and multibyte character functions such as mbtowc(3C) and wctomb(3C).

LC_NUMERIC affects the decimal point character and thousands separator character for
the formatted input/output functions and string conversion functions.

LC_TIME affects the date and time format as delivered by asctime(3C), ctime(3C),
getdate(3C), strftime(3C), and strptime(3C).

LC_COLLATE affects the sort order produced by collating functions such as strcoll (3C)
and strxfrm(3C).

LC_MONETARY affects the monetary formatted information returned by localeconv(3C).

LC_MESSAGES affects the behavior of messaging functions such as dgettext(3C),
gettext(3C), and gettxt(3C).

A value of "C" for locale specifies the traditional UNIX system behavior. At program
startup, the equivalent of

setlocale(LC_ALL, "C")

is executed. This has the effect of initializing each category to the locale described by the
environment "C".

A value of "" for locale specifies that the locale should be taken from environment vari-
able. The order in which the environment variables are checked for the various
categories is given below:

<table>
<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>

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.

modified 20 Dec 1996

SunOS 5.6

3C-1419
For category LC_ALL, the behavior is slightly different. If a pointer to a string is given for locale and LC_ALL is given for category, setlocale() attempts to set the locale for all the categories to locale. The locale may be a simple locale, consisting of a single locale, or a composite locale. If the locales for all the categories are the same after all the attempted locale changes, setlocale() will return a pointer to the common simple locale. If there is a mixture of locales among the categories, setlocale() will return a composite locale.

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
/usr/lib/locale/locale database directory for locale

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe with exceptions</td>
</tr>
<tr>
<td>CSI</td>
<td>Enabled</td>
</tr>
</tbody>
</table>

SEE ALSO
locale(1), ctype(3C), getdate(3C), gettext(3C), gettxt(3C), isdigit(3C), localeconv(3C), mbtowc(3C), strcoll(3C), strftime(3C), strptime(3C), strxfrm(3C), tolower(3C), wctomb(3C), libc(4), attributes(5), environ(5), locale(5)

NOTES
To change locale in a 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.

Internationalization functions by setlocale() are supported only when the dynamic linking version of libc has been linked with the application. If the static linking version of libc has been linked with the application, setlocale() can handle only C and POSIX locales.
NAME

setsockopt – set the socket options

SYNOPSIS

cc [ flag ... ] file ... -lxnet [ library ... ]
#include <sys/socket.h>

int setsockopt(int socket, int level, int option_name, const void *option_value,
               size_t option_len);

DESCRIPTION

The setsockopt() function sets the option specified by the option_name argument, at the
protocol level specified by the level argument, to the value pointed to by the option_value
argument for the socket associated with the file descriptor specified by the socket argu-
ment.

The level argument specifies the protocol level at which the option resides. To set options
at the socket level, specify the level argument as SOL_SOCKET. To set options at other
levels, supply the appropriate protocol number for the protocol controlling the option.
For example, to indicate that an option will be interpreted by the TCP (Transport Control
Protocol), set level to the protocol number of TCP, as defined in the <netinet/in.h> header,
or as determined by using getprotobyname(3XN).

The option_name argument specifies a single option to set. The option_name argument and
any specified options are passed uninterpreted to the appropriate protocol module for
interpretations. The <sys/socket.h> header defines the socket level options. The socket
level options can be enabled or disabled. The options are as follows:

SO_DEBUG       Turns on recording of debugging information. This option enables or
disables debugging in the underlying protocol modules. This option
takes an int value.

SO_BROADCAST    Permits sending of broadcast messages, if this is supported by the proto-
col. This option takes an int value.

SO_REUSEADDR   Specifies that the rules used in validating addresses supplied to
bind(3XN) should allow reuse of local addresses, if this is supported by
the protocol. This option takes an int value.

SO_KEEPALIVE   Keeps connections active by enabling the periodic transmission of mes-
sages, if this is supported by the protocol. This option takes an int
value.

If the connected socket fails to respond to these messages, the connec-
tion is broken and processes writing to that socket are notified with a
SIGPIPE signal.

SO_LINGER      Lingers on a close(2) if data is present. This option controls the action
taken when unsent messages queue on a socket and close(2) is per-
formed. If SO_LINGER is set, the system blocks the process during
close(2) until it can transmit the data or until the time expires. If
SO_LINGER is not specified, and close(2) is issued, the system handles
the call in a way that allows the process to continue as quickly as possible. This option takes a linger structure, as defined in the `<sys/socket.h>` header, to specify the state of the option and linger interval.

SO_OOBINLINE Leaves received out-of-band data (data marked urgent) in line. This option takes an int value.

SO_SNDBUF Sets send buffer size. This option takes an int value.

SO_RCVBUF Sets receive buffer size. This option takes an int value.

For boolean options, 0 indicates that the option is disabled and 1 indicates that the option is enabled.

Options at other protocol levels vary in format and name.

**RETURN VALUES**

Upon successful completion, `setsockopt()` returns 0. Otherwise, −1 is returned and `errno` is set to indicate the error.

**ERRORS**

The `setsockopt()` function will fail if:

- **EBADF** The `socket` argument is not a valid file descriptor.
- **EINVAL** The specified option is invalid at the specified socket level or the socket has been shut down.
- **ENOPROTOOPT** The option is not supported by the protocol.
- **ENOTSOCK** The `socket` argument does not refer to a socket.

The `setsockopt()` function may fail if:

- **ENOMEM** There was insufficient memory available for the operation to complete.
- **ENOBUFFS** Insufficient resources are available in the system to complete the call.
- **ENOSR** There were insufficient STREAMS resources available for the operation to complete.

**USAGE**

The `setsockopt()` function provides an application program with the means to control socket behaviour. An application program can use `setsockopt()` to allocate buffer space, control timeouts, or permit socket data broadcasts. The `<sys/socket.h>` header defines the socket-level options available to `setsockopt()`.

Options may exist at multiple protocol levels. The SO_ options are always present at the uppermost socket level.

**ATTRIBUTES**

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
SEE ALSO  bind(3XN), endprotoent(3XN), getsockopt(3XN), socket(3XN), attributes(5), socket(5)
<table>
<thead>
<tr>
<th>NAME</th>
<th>set_term – switch between terminals</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYNOPSIS</td>
<td>#include &lt;curses.h&gt;</td>
</tr>
<tr>
<td></td>
<td>SCREEN *set_term (SCREEN *new);</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>new Is the new terminal to which the set_term() function will switch.</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>The set_term() function switches to the terminal specified by new and returns a screen reference to the previous terminal. Calls to subsequent X/Open Curses functions affect the new terminal.</td>
</tr>
<tr>
<td>RETURN VALUES</td>
<td>On success, the set_term() function returns a pointer to the previous screen. Otherwise, it returns a null pointer.</td>
</tr>
<tr>
<td>ERRORS</td>
<td>None.</td>
</tr>
</tbody>
</table>
NAME    shm_open – open a shared memory object

SYNOPSIS cc [ flag ... ] file ... -lposix4 [ library ... ]
#include <sys/mman.h>
int shm_open(const char *name, int oflag, mode_t mode);

DESCRIPTION shm_open() either opens a file descriptor for the shared memory object with the name referenced by name. 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.

name points to a string naming a shared memory object. The name argument should conform to the construction rules for a pathname. If a process makes multiple successful calls to shm_open(), with the same value for name, the same semaphore address will be returned for each successful call, provided that there have been no calls to sem_unlink(3R) for this semaphore. The first character of name must be a slash (/) character and the remaining characters of name cannot include any slash characters. For maximum portability, name should include no more than 14 characters, but this limit is not enforced.

The file status flags and file access modes of the open file descriptor are set according to the value of oflag: 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 oflag):

O_RDONLY  Open for read access only.
O_RDWR    Open for read or write access.

Any combination of the remaining flags may be bitwise inclusive OR-ed with the value of oflag:

O_CREAT   If name 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 mode, modified by clearing all bits set in the file mode creation mask of the process (see umask(2)).

mode 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.

If the shared memory object does exist, this flag will have no effect, except as specified under O_EXCL below.

O_EXCL    If both OEXCL and O_CREAT are set, shm_open() fails if the shared memory object, name, exists. The check for the existence of the shared memory object and the creation of the object if it does not exist is

modified 30 Dec 1996  SunOS 5.6  3R-1425
atomic with respect to other processes executing `shm_open()` naming the same shared memory object with `O_EXCL` 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 `oflag` 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.
- **EINVAL** `name` is an invalid file description.
- **EMFILE** The number of open file descriptors in this process exceeds `OPEN_MAX`.
- **ENAMETOOLONG** The length of the `name` string exceeds `PATH_MAX`, or a pathname component is longer than `NAME_MAX` while `_POSIX_NO_TRUNC` is in effect.
- **ENOENT** `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`

**ATTRIBUTES**

See `attributes(5)` for descriptions of the following attributes:

```
<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
```

**SEE ALSO**

`close(2), dup(2), exec(2), fcntl(2), mmap(2), umask(2), shm_unlink(3R), sysconf(3C), attributes(5), 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.
NAME  shm_unlink – remove a shared memory object

SYNOPSIS  cc [ flag ...] file ... -lposix4 [ library ... ]
           int shm_unlink(const char *name);

DESCRIPTION  shm_unlink() removes the name of the shared memory object named by the string
             pointed to by name. 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 remo-
             val of the memory object contents will be postponed until all open and mapped refer-
             ences 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 name string exceeds PATH_MAX, or a pathname com-
                          ponent 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.

ATTRIBUTES  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  close(2), mmap(2), mlock(3C), shm_open(3R), attributes(5)
NAME  
shutdown – shut down part of a full-duplex connection

SYNOPSIS  
cc [ flag ... ] file ... -lsocket -lssl [ library ... ]  
int shutdown(int s, int how);

DESCRIPTION  
The shutdown( ) call shuts down all or part of a full-duplex connection on the socket associated with s. If how is 0, then further receives will be disallowed. If how is 1, then further sends will be disallowed. If how is 2, then further sends and receives will be disallowed.

RETURN VALUES  
A 0 is returned if the call succeeds, −1 if it fails.

ERRORS  
The call succeeds unless:
EBADF  
s is not a valid file descriptor.
ENOMEM  
There was insufficient user memory available for the operation to complete.
ENOSR  
There were insufficient STREAMS resources available for the operation to complete.
ENOTCONN  
The specified socket is not connected.
ENOTSOCK  
s is not a socket.

ATTRIBUTES  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  
connect(3N), socket(3N), attributes(5), socket(5)

NOTES  
The how values should be defined constants.

modified 16 May 1997  
SunOS 5.6  
3N-1429
NAME
shutdown – shut down socket send and receive operations

SYNOPSIS
cc [ flag ...] file ... -lxnet [ library ...]
#include <sys/socket.h>
int shutdown(int socket, int how);

DESCRIPTION
The shutdown() function disables subsequent send and/or receive operations on a
socket, depending on the value of the how argument. This function takes the following
arguments:
socket Specifies the file descriptor of the socket.
how Specifies the type of shutdown. The values are as follows:
SHUT_RD Disables further receive operations.
SHUT_WR Disables further send operations.
SHUT_RDWR Disables further send and receive operations.

RETURN VALUES
Upon successful completion, shutdown() returns 0. Otherwise, −1 is returned and errno
is set to indicate the error.

ERRORS
The shutdown() function will fail if:
EBADF The socket argument is not a valid file descriptor.
ENOTCONN The socket is not connected.
ENOTSOCK The socket argument does not refer to a socket.
EINVAL The how argument is invalid.

The shutdown() function may fail if:
ENOBUS Insufficient resources were available in the system to perform the opera-
tion.
ENOSR There were insufficient STREAMS resources available for the operation to
complete.

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
read(2), write(2), getsockopt(3XN), recv(3XN), recvfrom(3XN), recvmsg(3XN),
select(3C), send(3XN), sendto(3XN), setsockopt(3XN), socket(3XN), attributes(5),
socket(5)
NAME
    sigblock, sigmask, sigpause, sigsetmask – block signals

SYNOPSIS
    /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;

DESCRIPTION
    sigblock, sigmask, sigpause, sigsetmask – block signals
    sigblock() adds the signals specified in mask to the set of signals currently being blocked
    from delivery. Signals are blocked if the appropriate bit in mask is a 1; the macro sigmask
    is provided to construct the mask for a given signum. sigblock() returns the previous
    mask. The previous mask may be restored using sigsetmask().
    sigpause() assigns mask to the set of masked signals and then waits for a signal to arrive;
    on return the set of masked signals is restored. mask is usually 0 to indicate that no sig-
    nals are now to be blocked. sigpause() always terminates by being interrupted, returning
    −1 and setting errno to EINTR.
    sigsetmask() sets the current signal mask (those signals that are blocked from delivery).
    Signals are blocked if the corresponding bit in mask is a 1; the macro
    sigmask is provided to construct the mask for a given signum.
    In normal usage, a signal is blocked using sigblock(). To begin a critical section, vari-
    ables modified on the occurrence of the signal are examined to determine that there is no
    work to be done, and the process pauses awaiting work by using sigpause() with the
    mask returned by sigblock().
    It is not possible to block SIGKILL, SIGSTOP, or SIGCONT, this restriction is silently
    imposed by the system.

RETURN VALUES
    sigblock() and sigsetmask() return the previous set of masked signals. sigpause() returns −1 and sets errno to EINTR.

SEE ALSO
    kill(2), sigaction(2), signal(3B), sigvec(3B)

NOTES
    Use of these interfaces should be restricted to only applications written on BSD plat-
    forms. Use of these interfaces with any of the system libraries or in multi-thread applica-
    tions is unsupported.

modified 19 Feb 1993

SunOS 5.6

3B-1431
NAME

sigfpe – signal handling for specific SIGFPE codes

SYNOPSIS

```
#include <floatingpoint.h>
#include <siginfo.h>

sigfpe_handler_type sigfpe(sigfpe_code_type code, sigfpe_handler_type hdl);
```

DESCRIPTION

This function allows signal handling to be specified for particular SIGFPE codes. A call to `sigfpe()` defines a new handler `hdl` for a particular SIGFPE code 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)`.

`code` is usually one of the five IEEE 754-related SIGFPE codes:

- **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

Three steps are required to intercept an IEEE 754-related SIGFPE code with `sigfpe()`:

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.

`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.

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.
A user-specified signal handler might look like this:

```c
#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:

```c
#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_overlow and fp_invalid handlers; set the new
    * fp_overlow 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_overlow and fp_invalid handlers
    */
```
```c
  sigfpe(FPE_FLTOVF, old_handler1);
sigfpe(FPE_FLTINV, old_handler2);
}
```

**FILES**

/usr/include/floatingpoint.h
/usr/include/siginfo.h

**ATTRIBUTES**

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

**SEE ALSO**

sigaction(2), abort(3C), signal(3C), attributes(5), floatingpoint(5)

**DIAGNOSTICS**

sigfpe() returns BADSIG if `code` is not zero or a defined SIGFPE code.
NAME  siginterrupt – allow signals to interrupt functions

SYNOPSIS  
/usr/ucb/cc [flag ...] file ...

int siginterrupt(sig, flag)

DESCRIPTION  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.

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.

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.

NOTES  Use of these interfaces should be restricted to only applications written on BSD platforms. Use of these interfaces with any of the system libraries or in multi-threaded applications is unsupported.

This library routine uses an extension of the sigvec(3B) function that is not available in 4.2 BSD, hence it should not be used if backward compatibility is needed.

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  sig is not a valid signal.

SEE ALSO  sigblock(3B), sigvec(3B), signal(3C)
NAME

signal – simplified software signal facilities

SYNOPSIS

/usr/ucb/cc [ flag ... ] file ...
#include <signal.h>
void (*signal(sig,func))()
int sig;
void (*func)();

DESCRIPTION

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.

A signal is generated by some abnormal event, initiated by a user at a terminal (quit, interrupt, stop), by a program error (bus error, etc.), by request of another program (kill), or when a process is stopped because it wishes to access its control terminal while in the background (see termio(7I)). Signals are optionally generated when a process resumes after being stopped, when the status of child processes changes, or when input is ready at the control terminal. Most signals cause termination of the receiving process if no action is taken; some signals instead cause the process receiving them to be stopped, or are simply discarded if the process has not requested otherwise. Except for the SIGKILL and SIGSTOP signals, the signal() call allows signals either to be ignored or to interrupt to a specified location. See sigvec(3B) for a complete list of the signals.

If func is SIG_DFL, the default action for signal sig 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 func 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 func is called.

A return from the function unblocks the handled signal and continues the process at the point it was interrupted.

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).

The value of signal() is the previous (or initial) value of func 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.

RETURN VALUES

The previous action is returned on a successful call. Otherwise, −1 is returned and errno is set to indicate the error.

ERRORS

signal() will fail and no action will take place if the following occurs:

EINVAL sig is not a valid signal number, or is SIGKILL or SIGSTOP.
SEE ALSO  kill(1), exec(2), fcntl(2), fork(2), getitimer(2), getrlimit(2), kill(2), ptrace(2), read(2),
sigaction(2), wait(2), write(2), abort(3C), setjmp(3B), sigblock(3B), sigstack(3B),
sigvec(3B), wait(3B), setjmp(3C), signal(3C), signal(5), termio(7I)

NOTES  Use of these interfaces should be restricted to only applications written on BSD platforms. Use of these interfaces with any of the system libraries or in multi-thread applications is unsupported.

The handler routine, func, can be declared:

```c
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
```
#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);
```

### DESCRIPTION
These functions provide simplified signal management for application processes. See `signal(5)` for an explanation of general signal concepts.

`signal()` and `sigset()` are used to modify signal dispositions. `sig` specifies the signal, which may be any signal except `SIGKILL` and `SIGSTOP`. `disp` specifies the signal’s disposition, which may be `SIG_DFL`, `SIG_IGN`, or the address of a signal handler. If `signal()` is used, `disp` is the address of a signal handler, and `sig` 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 `disp` is the address of a signal handler, the system adds `sig` 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 `disp` is equal to `SIG_HOLD`, `sig` is added to the calling process’s signal mask and the signal’s disposition remains unchanged.

- `sighold()` adds `sig` to the calling process’s signal mask.
- `sigrelse()` removes `sig` from the calling process’s signal mask.
- `sigignore()` sets the disposition of `sig` to `SIG_IGN`.
- `sigpause()` removes `sig` from the calling process’s signal mask and suspends the calling process until a signal is received.

### RETURN VALUES
On success, `signal()` returns the signal’s previous disposition. On failure, it returns `SIG_ERR` and sets `errno` to indicate the error.

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.

All other functions return `0` on success. On failure, they return `−1` and set `errno` to indicate the error.
## 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

`exit(2)`, `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)`).

The system guarantees that if more than one instance of the same signal is generated to a process, at least one signal will be received. It does not guarantee the reception of every generated signal.
NAME  significand – significand function

SYNOPSIS  cc [ flag ... ] file ... -lm [ library ... ]
            #include <math.h>
            double significand(double x);

DESCRIPTION  The significand() function, along with the logb(3M) and scalb(3M) functions, allows
            users to verify compliance to ANSI/IEEE Std 754-1985 by running certain test vectors dis-
            tributed by the University of California.
            If x equals sig * 2**n with 1 ≤ sig < 2, then significand(x) returns sig for exercising the
            fraction-part(F) test vector. significand(x) is not defined when x is either 0, ±Inf or NaN.

RETURN VALUES  For exceptional cases, matherr(3M) tabulates the values to be returned as dictated by
            various Standards.

ATTRIBUTES  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  logb(3M), matherr(3M), scalb(3M), attributes(5)
**NAME**  
sigqueue – queue a signal to a process

**SYNOPSIS**  
cc [ flag ...] file ... -lposix4 [ library ...]
#include <signal.h>
int sigqueue(pid_t pid, int signo, const union sigval value);

union sigval {
    int sival_int; /* integer value */
    void *sival_ptr; /* pointer value */
};

**DESCRIPTION**  
sigqueue() causes the signal, signo to be sent with the value, value to the process, pid. If signo 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 pid.

The conditions required for a process to have permission to queue a signal to another process are the same as for kill(2).

If resources were not available to queue the signal, sigqueue() exits and returns immediately. If SA_SIGINFO is set for signo in the receiving process, and if the resources were available, the signal is left queued and pending. If SA_SIGINFO is not set for signo, then signo is sent at least once to the receiving process.

If the value of pid causes signo to be generated for the sending process, and if signo is not blocked, either signo or at least the pending, unblocked signal with the lowest number will be delivered to the sending process before sigqueue() returns.

**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**  
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.

EINVAL The value of signo is an invalid or unsupported signal number.

ENOSYS sigqueue() is not supported by this implementation.

EPERM The process does not have the appropriate privilege to send the signal to the receiving process.

ESRCH The process pid does not exist.

**ATTRIBUTES**  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Async-Signal-Safe</td>
</tr>
</tbody>
</table>

modified 30 Dec 1996  
SunOS 5.6  
3R-1441
SEE ALSO  
kill(2), sigwaitinfo(3R), attributes(5), siginfo(5), signal(5)
NAME  sigsetops, sigemptyset, sigfillset, sigaddset, sigdelset, sigismember – manipulate sets of signals

SYNOPSIS  #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);

DESCRIPTION  These functions manipulate sigset_t data types, representing the set of signals supported by the implementation.

            sigemptyset() initializes the set pointed to by set to exclude all signals defined by the system.
            sigfillset() initializes the set pointed to by set to include all signals defined by the system.
            sigaddset() adds the individual signal specified by the value of signo to the set pointed to by set.
            sigdelset() deletes the individual signal specified by the value of signo from the set pointed to by set.
            sigismember() checks whether the signal specified by the value of signo is a member of the set pointed to by set.

Any object of type sigset_t must be initialized by applying either sigemptyset() or sigfillset() before applying any other operation.

RETURN VALUES  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.

ERRORS  sigaddset(), sigdelset(), and sigismember() will fail if the following is true:

            EINVAL           The value of the signo argument is not a valid signal number.
           EFAULT           The set argument specifies an invalid address.

ATTRIBUTES  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

modified 29 Dec 1996
| SEE ALSO       | sigaction(2), sigpending(2), sigprocmask(2), sigsuspend(2), attributes(5), signal(5) |
NAME  sigstack – set and/or get signal stack context

SYNOPSIS  /usr/ucb/cc [ flag ... ] file ...

#include <signal.h>

int sigstack(nss, oss)
    struct sigstack *nss, *oss;

DESCRIPTION  The sigstack() function allows users to define an alternate stack, called the “signal stack”, on which signals are to be processed. When a signal’s action indicates its handler should execute on the signal stack (specified with a sigvec(3B) call), the system checks to see if the process is currently executing on that stack. If the process is not currently executing on the signal stack, the system arranges a switch to the signal stack for the duration of the signal handler’s execution.

A signal stack is specified by a sigstack() structure, which includes the following members:

```
    char       *ss_sp;  /* signal stack pointer */
    int        ss_onstack;  /* current status */
```

The ss_sp member is the initial value to be assigned to the stack pointer when the system switches the process to the signal stack. Note that, on machines where the stack grows downwards in memory, this is not the address of the beginning of the signal stack area.

The ss_onstack member is zero or non-zero depending on whether the process is currently executing on the signal stack or not.

If nss is not a null pointer, sigstack() sets the signal stack state to the value in the sigstack() structure pointed to by nss. If nss is a null pointer, the signal stack state will be unchanged. If oss is not a null pointer, the current signal stack state is stored in the sigstack() structure pointed to by oss.

RETURN VALUES  Upon successful completion, 0 is returned. Otherwise, −1 is returned and errno is set to indicate the error.

ERRORS  The sigstack() function will fail and the signal stack context will remain unchanged if one of the following occurs.

EFAULT  Either nss or oss points to memory that is not a valid part of the process address space.

SEE ALSO  sigaltstack(2), sigvec(3B), signal(3C)

WARNINGS  Signal stacks are not “grown” automatically, as is done for the normal stack. If the stack overflows unpredictable results may occur.

NOTES  Use of these interfaces should be restricted to only applications written on BSD platforms. Use of these interfaces with any of the system libraries or in multi-thread applications is unsupported.

modified 22 Jan 1997

SunOS 5.6

3B-1445
NAME  sigstack – set and/or get alternate signal stack context

SYNOPSIS
#include <signal.h>

int sigstack(struct sigstack *ss, struct sigstack *oss);

DESCRIPTION
The sigstack() function allows the calling process to indicate to the system an area of its address space to be used for processing signals received by the process.

If the ss argument is not a null pointer, it must point to a sigstack structure. The length of the application-supplied stack must be at least SIGSTKSZ bytes. If the alternate signal stack overflows, the resulting behavior is undefined. (See USAGE below.)

- The value of the ss_onstack member indicates whether the process wants the system to use an alternate signal stack when delivering signals.
- The value of the ss_sp member indicates the desired location of the alternate signal stack area in the process' address space.
- If the ss argument is a null pointer, the current alternate signal stack context is not changed.

If the oss argument is not a null pointer, it points to a sigstack structure in which the current alternate signal stack context is placed. The value stored in the ss_onstack member of oss will be non-zero if the process is currently executing on the alternate signal stack. If the oss argument is a null pointer, the current alternate signal stack context is not returned.

When a signal's action indicates its handler should execute on the alternate signal stack (specified by calling sigaction(2)), sigstack() checks to see if the process is currently executing on that stack. If the process is not currently executing on the alternate signal stack, the system arranges a switch to the alternate signal stack for the duration of the signal handler's execution.

After a successful call to one of the exec functions, there are no alternate signal stacks in the new process image.

RETURN VALUES
Upon successful completion, sigstack() returns 0. Otherwise, it returns −1 and sets errno to indicate the error.

ERRORS
The sigstack() function will fail if:

EPERM An attempt was made to modify an active stack.

USAGE
A portable application, when being written or rewritten, should use sigaltstack(2) instead of sigstack().

The direction of stack growth is not indicated in the historical definition of struct sigstack. The only way to portably establish a stack pointer is for the application to determine stack growth direction, or to allocate a block of storage and set the stack pointer to the middle. sigstack() may assume that the size of the signal stack is SIGSTKSZ as found in <signal.h>. An application that would like to specify a signal stack size other than...
SIGSTKSZ should use `sigaltstack(2)`. Applications should not use `longjmp(3C)` to leave a signal handler that is running on a stack established with `sigstack()`. Doing so may disable future use of the signal stack. For abnormal exit from a signal handler, `siglongjmp(3C)`, `setcontext(2)`, or `swapcontext(3C)` may be used. These functions fully support switching from one stack to another.

The `sigstack()` function requires the application to have knowledge of the underlying system’s stack architecture. For this reason, `sigaltstack(2)` is recommended over this function.

SEE ALSO `fork(2)`, `_longjmp(3C)`, `longjmp(3C)`, `setjmp(3C)`, `sigaltstack(2)`, `siglongjmp(3C)`, `sigsetjmp(3C)`
NAME
sigvec – software signal facilities

SYNOPSIS
/usr/ucb/cc [flag ...] file ...
#include <signal.h>
int sigvec(sig, nvec, ovec)
int sig;
struct sigvec *nvec, *ovec;

DESCRIPTION
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 handler to which a signal is delivered, or specify that a signal is to be blocked or
ignored. 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 pro-
cess. This may be changed, on a per-handler basis, so that signals are taken on a special
signal stack.

All signals have the same priority. Signal routines execute with the signal that caused
their invocation to be blocked, but other signals may yet occur. A global signal mask
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.

A process may also specify a set of flags 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 pend-
ing for the process. If the signal is not currently blocked 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 pro-
cess wishes to resume in a different context, then it must arrange to restore the previous
context itself.

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.

The action to be taken when the signal is delivered is specified by a sigvec() structure,
which includes the following members:

<table>
<thead>
<tr>
<th>Type</th>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void</td>
<td>(*sv_handler)()</td>
<td>/* signal handler */</td>
</tr>
<tr>
<td>int</td>
<td>sv_mask;</td>
<td>/* signal mask to apply */</td>
</tr>
<tr>
<td>int</td>
<td>sv_flags;</td>
<td>/* see signal options */</td>
</tr>
<tr>
<td>#define</td>
<td>SV_ONSTACK</td>
<td>/* take signal on signal stack */</td>
</tr>
<tr>
<td>#define</td>
<td>SV_INTERRUPT</td>
<td>/* do not restart system on signal return */</td>
</tr>
<tr>
<td>#define</td>
<td>SV_RESETHAND</td>
<td>/* reset handler to SIG_DFL when signal taken*/</td>
</tr>
</tbody>
</table>

3B-1448 SunOS 5.6 modified 10 Jan 1996
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(7I))
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-caching function, unless the signal is SIGILL, SIGPWR, or SIGTRAP. Also, if this bit is set, the bit for that signal in the signal mask will not be set; unless the signal mask associated with that signal blocks that signal, further occurrences of that signal will not be blocked. The SV_RESETHAND flag is not available in 4.2BSD, hence it should not be used if backward compatibility is needed.

The default action for a signal may be reinstated by setting the signal’s handler to SIG_DFL; this default is termination except for signals marked with ● or †. Signals marked with ● are discarded if the action is SIG_DFL; signals marked with † cause the process to stop. If the process is terminated, a “core image” will be made in the current working directory of the receiving process if the signal is one for which an asterisk appears in the above list (see core(4)).

If the handler for that signal is SIG_IGN, the signal is subsequently ignored, and pending instances of the signal are discarded.

If a caught signal occurs during certain functions, the call is normally restarted. The call can be forced to terminate prematurely with an EINTR error return by setting the SV_INTERRUPT bit in the flags for that signal. The SV_INTERRUPT flag is not available in 4.2BSD, hence it should not be used if backward compatibility is needed. The affected functions are read(2) or write(2) on a slow device (such as a terminal or pipe or other socket, but not a file) and during a wait(2).

After a fork(2) or vfork(2) the child inherits all signals, the signal mask, the signal stack, and the restart/interrupt and reset-signal-handler flags.

The execve(2) call resets all caught signals to default action and resets all signals to be caught on the user stack. Ignored signals remain ignored; the signal mask remains the same; signals that interrupt functions continue to do so.

The accuracy of addr is machine dependent. For example, certain machines may supply an address that is on the same page as the address that caused the fault. If an appropriate addr cannot be computed it will be set to SIG_NOADDR.

RETURN VALUES
A 0 value indicates that the call succeeded. A −1 return value indicates that an error occurred and errno is set to indicate the reason.
ERRORS

sigvec() will fail and no new signal handler will be installed if one of the following occurs:

EFAULT

Either nvec or ovec is not a NULL pointer and points to memory that is not a valid part of the process address space.

EINVAL

sig is not a valid signal number, or, SIGKILL, or SIGSTOP.

SEE ALSO

intro(2), exec(2), fcntl(2), fork(2), getitimer(2), getrlimit(2), ioctl(2), kill(2), ptrace(2), read(2), umask(2), vfork(2), wait(2), write(2), setjmp(3C) sigblock(3B), sigstack(3B), signal(3B), wait(3B), signal(3C), core(4), streamio(7I), termio(7I)

NOTES

Use of these interfaces should be restricted to only applications written on BSD platforms. Use of these interfaces with any of the system libraries or in multi-thread applications is unsupported.

SIGPOLL is a synonym for SIGIO. A SIGIO will be issued when a file descriptor corresponding to a STREAMS (see intro(2)) file has a "selectable" event pending. Unless that descriptor has been put into asynchronous mode (see fcntl(2)), a process may specifically request that this signal be sent using the I_SETSIG ioctl(2) call (see streamio(7I)). Otherwise, the process will never receive SIGPOLLs.

The handler routine can be declared:

    void handler(int sig, int code, struct sigcontext *scp, char *addr);

Here sig is the signal number; code is a parameter of certain signals that provides additional detail; scp is a pointer to the sigcontext structure (defined in signal.h), used to restore the context from before the signal; and addr is additional address information.

The signals SIGKILL, SIGSTOP, and SIGCONT cannot be ignored.
NAME  sigwaitinfo, sigtimedwait – wait for queued signals

SYNOPSIS  cc [ flag ... ] file ... --lposix4 [ library ... ]
#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 */
};

DESCRIPTION  sigwaitinfo() and sigtimedwait() select the pending signal from the set specified by set.
When multiple signals are pending, the lowest numbered one will be selected. The selec-
tion order between realtime and non-realtime signals, or between multiple pending non-
realtime signals, is unspecified.

If no signal in set is pending at the time of the call, sigwaitinfo() suspends the calling
process until one or more signals in set 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 timeout. If the timespec struc-
ture pointed to by timeout is zero-valued, and if none of the signals specified by set are
pending, then sigtimedwait() returns immediately with the error EAGAIN.

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 asynchro-
nously and the wait is interrupted.

If info 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 info is non-NULL, the value is stored in the si_value
member of info. The system resource used to queue the signal is released and made
available to queue other signals.

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.

If no further signals are queued for the selected signal, the pending indication for that sig-
nal 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.

ATTRIBUTES
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Async-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO `time(2)`, `sigqueue(3R)`, `attributes(5)`, `siginfo(5)`, `signal(5)`
NAME       sin – sine function

SYNOPSIS   cc [ flag ... ] file ... -lm [ library ... ]
           #include <math.h>
           double sin(double x);

DESCRIPTION The sin() function computes the sine of its argument \( x \), measured in radians.

RETURN VALUES Upon successful completion, sin() returns the sine of \( x \).
If \( x \) is NaN or ±Inf, NaN is returned.

ERRORS     No errors will occur.

ATTRIBUTES See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO   asin(3M), isnan(3M), attributes(5)
NAME
sinh – hyperbolic sine function

SYNOPSIS
cc [ flag ... ] file ... -lm [ library ... ]
#include <math.h>
double sinh(double x);

DESCRIPTION
The sinh() function computes the hyperbolic sine of x.

RETURN VALUES
Upon successful completion, sinh() returns the hyperbolic sine of x.
If the result would cause an overflow, ±HUGE_VAL is returned and errno is set to ERANGE.
If x is NaN, NaN is returned.
For exceptional cases, matherr(3M) tabulates the values to be returned as dictated by Standards other than XPG4.

ERRORS
The sinh() function will fail if:
ERANGE The result would cause overflow.

USAGE
An application wishing to check for error situations should set errno to 0 before calling sinh(). If errno is non-zero on return, or the return value is NaN, an error has occurred.

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
asinh(3M), cosh(3M), isnan(3M), matherr(3M), tanh(3M), attributes(5), standards(5)
NAME
sleep – suspend execution for interval

SYNOPSIS
/usr/ucb/cc [ flag ... ] file ...
int sleep( seconds )
unsigned seconds;

DESCRIPTION
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.
sleep() is implemented by setting an interval timer and pausing until it expires. The pre-
vious 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.

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Async-Signal-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
alarm(2), getitimer(2), longjmp(3C), siglongjmp(3C), sleep(3C), usleep(3C), attributes(5)

NOTES
Use of these interfaces should be restricted to only applications written on BSD plat-
forms. Use of these interfaces with any of the system libraries or in multi-thread applica-
tions is unsupported.
SIGALRM should not be blocked or ignored during a call to sleep(). Only a prior call to
alarm(2) should generate SIGALRM for the calling process during a call to sleep(). A
signal-catching function should not interrupt a call to sleep() to call siglongjmp(3C) or
longjmp(3C) to restore an environment saved prior to the sleep() call.

WARNINGS
sleep() is slightly incompatible with alarm(2). Programs that do not execute for at least
one second of clock time between successive calls to sleep() indefinitely delay the alarm
signal. Use sleep(3C). Each sleep(3C) call postpones the alarm signal that would have
been sent during the requested sleep period to occur one second later.
NAME  sleep – suspend execution for interval

SYNOPSIS  
#include <unistd.h>

unsigned sleep(unsigned seconds);

DESCRIPTION  The current process is suspended from execution for the number of seconds specified by the argument. The actual suspension time may be less than that requested because any caught signal will terminate the sleep() following execution of that signal’s catching routine. Also, the suspension time may be longer than requested by an arbitrary amount because of the scheduling of other activity in the system. The value returned by sleep() will be the “unslept” amount (the requested time minus the time actually slept) in case the caller had an alarm set to go off earlier than the end of the requested sleep() time, or premature arousal because of another caught signal.

The routine is implemented by setting an alarm signal and pausing until it (or some other signal) occurs. The previous state of the alarm signal is saved and restored. The calling program may have set up an alarm signal before calling sleep(). If the sleep() time exceeds the time until such alarm signal, the process sleeps only until the alarm signal would have occurred. The caller’s alarm catch routine is executed just before the sleep() routine returns. But if the sleep() time is less than the time till such alarm, the prior alarm time is reset to go off at the same time it would have without the intervening sleep().

ATTRIBUTES  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  alarm(2), pause(2), signal(3C), attributes(5)

NOTES  SIGALRM should not be blocked or ignored during a call to sleep(). Only a prior call to alarm(2) should generate SIGALRM for the calling process during a call to sleep(). In a multithreaded program, only the invoking thread is suspended from execution.

modified 29 Dec 1996  SunOS 5.6  3C-1457
NAME
slk_attroff, slk_attr_off, slk_attron, slk_attr_on, slk_attrset, slk_attr_set, slk_clear,
slk_color, slk_init, slk_label, slk_noutrefresh, slk_refresh, slk_restore, slk_set, slk_touch,
slk_wset – manipulate soft labels

SYNOPSIS
#include <term.h>
int slk_attroff(const chtype attrs);
int slk_attr_off(const attr_t attrs, void *opts);
int slk_attron(const chtype attrs);
int slk_attr_on(const attr_t attrs, void *opts);
int slk_attrset(const chtype attrs);
int slk_attr_set(const attr_t attrs, short color_pair, void *opts);
int slk_clear(void);
int slk_color(short color_pair);
int slk_init(int fmt);
char *slk_label(int labnum);
int slk_noutrefresh(void);
int slk_refresh(void);
int slk_restore(void);
int slk_set(int labnum, const char *label, int justify);
int slk_touch(void);
int slk_wset(int labnum, const wchar_t *label, int justify);

ARGUMENTS
attrs are the foreground window attributes to be added or removed.
opts Is reserved for future use. Currently, this must be a null pointer.
color_pair Is a color pair.
fmt Is the format of how the labels are arranged on the screen.
labnum Is the number of the soft label.
label Is the name to be given to a soft label.
justify Is a number indicating how to justify the label name.

DESCRIPTION
These functions manipulate the soft function-key labels that many terminals feature. For
terminals without soft labels, X/Open Curses uses ripofline(3XC) to allocate the bottom
line of stdscr to emulating them. There can be up to eight soft labels, each with a width
of up to eight display columns.

The slk_init() function must be called before calling initscr(3XC), newterm(3XC), or
ripofline() if you are going to use soft labels. It has the effect of calling ripofline() to
reserve a screen line. The fmt argument specifies how the labels are to be arranged on the

3XC-1458 SunOS 5.6 modified 1 Jun 1996
screen. If \texttt{fmt} is 0, there is a 3-2-3 arrangement of labels. If \texttt{fmt} is 1, there is a 4-4 arrangement.

The \texttt{slk_set()} and \texttt{slk_wset()} functions assign the label name \texttt{label} to the soft label numbered \texttt{labnum} (from 1 to 8). \texttt{label} can be no more than eight display columns in width. The \texttt{justify} argument indicates how the label name is justified within its reserved space:

- 0 Left justify the label name
- 1 Center the label name
- 2 Right justify the label name

The \texttt{slk_refresh()} and \texttt{slk_noutrefresh()} functions correspond to the \texttt{wrefresh(3XC)} and \texttt{wnoutrefresh(3XC)} functions described in the \texttt{doupdate(3XC)} man page and are used to update the actual soft label text on the screen.

The \texttt{slk_label()} returns the label name assigned to the label number \texttt{labnum}.

The \texttt{slk_clear()} clears the soft labels from the screen.

The \texttt{slk_restore()} restores the soft label information to the screen after a call to \texttt{slk_clear()}.

The \texttt{slk_touch()} marks all soft labels as needing to be updated when \texttt{slk_refresh()} or \texttt{slk_noutrefresh()} is next called.

The \texttt{slk_attron()}, \texttt{slk_attrset()}, and \texttt{slk_attroff()} functions behave similarly to the \texttt{attron(3XC)}, \texttt{attrset(3XC)}, and \texttt{attroff(3XC)} functions.

The \texttt{slk_attr_on()}, \texttt{slk_attr_off()}, \texttt{slk_attr_set()} and \texttt{slk_color()} functions behave similarly to the \texttt{attr_on(3XC)}, \texttt{attr_off(3XC)}, \texttt{attr_set(3XC)}, and \texttt{color_set(3XC)} functions. As a result, they support color and the attribute constants whose name begin with \texttt{WA}. 

**RETURN VALUES**

On success, the \texttt{slk_label()} function returns the requested label name. Otherwise, it returns a null pointer.

On success, the other functions return \texttt{OK}. Otherwise, they return \texttt{ERR}.

**ERRORS**

None.

**SEE ALSO**

\texttt{attr_get(3XC)}, \texttt{attroff(3XC)}, \texttt{delscreen(3XC)}, \texttt{rioffline(3XC)}
socket (3N) Network Functions

NAME
socket – create an endpoint for communication

SYNOPSIS
cc [flag...] file ... -lssocket -lsocket [library...]  
#include <sys/types.h>
#include <sys/socket.h>

int socket(int domain, int type, int protocol);

DESCRIPTION
socket() creates an endpoint for communication and returns a descriptor.  
The domain 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 protocol has been specified, but no exact match for the tuplet 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:
  - PF_UNIX  UNIX system internal protocols
  - PF_INET  ARPA Internet protocols

The socket has the indicated type, which specifies the communication semantics. Currently defined types are:
  - SOCK_STREAM  sequenced, reliable, two-way connection-based byte streams
  - SOCK_DGRAM  datagrams (connectionless, unreliable messages of a fixed (typically small) maximum length)
  - SOCK_RAW  provides access to internal network interfaces
  - SOCK_SEQPACKET  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.

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.

protocol 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 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 a extended period (for instance 5 minutes). A SIGPIPE signal is raised if a process sends on a broken stream; this causes naive processes, which do not handle the signal, to exit.

SOCK_SEQPACKET sockets employ the same system calls as SOCK_STREAM sockets. The only difference is that read(2) calls will return only the amount of data requested, and any remaining in the arriving packet will be discarded.

SOCK_DGRAM and SOCK_RAW sockets allow datagrams to be sent to correspondents named in sendto(3N) calls. Datagrams are generally received with recvfrom(3N), which returns the next datagram with its return address.

An fcntl(2) call can be used to specify a process group to receive a SIGURG signal when the out-of-band data arrives. It may also enable non-blocking I/O and asynchronous notification of I/O events with SIGIO signals.

The operation of sockets is controlled by socket level options. These options are defined in the file <sys/socket.h>. setsockopt(3N) and getsockopt(3N) are used to set and get options, respectively.

**RETURN VALUES**

A −1 is returned if an error occurs. Otherwise the return value is a descriptor referencing the socket.

**ERRORS**

The socket() call fails if:

- **EACCES**
  Permission to create a socket of the specified type and/or protocol is denied.

- **EMFILE**
  The per-process descriptor table is full.

- **ENOMEM**
  Insufficient user memory is available.
ENOSR There were insufficient STREAMS resources available to complete the operation.

EPROTONOSUPPORT The protocol type or the specified protocol is not supported within this domain.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO close(2), fcntl(2), ioctl(2), read(2), write(2), accept(3N), bind(3N), connect(3N), getsockname(3N), getsockopt(3N), listen(3N), recv(3N), setsockopt(3N), send(3N), shutdown(3N), socketpair(3N), attributes(5), in(5), socket(5)
NAME
socket – create an endpoint for communication

SYNOPSIS
cc [ flag ...] file ... -lxnet [ library ...]
#include <sys/socket.h>
int socket(int domain, int type, int protocol);

DESCRIPTION
The socket() function creates an unbound socket in a communications domain, and returns a file descriptor that can be used in later function calls that operate on sockets. The function takes the following arguments:

- **domain** Specifies the communications domain in which a socket is to be created.
- **type** Specifies the type of socket to be created.
- **protocol** Specifies a particular protocol to be used with the socket. Specifying a *protocol* of 0 causes socket() to use an unspecified default protocol appropriate for the requested socket type.

The domain argument specifies the address family used in the communications domain. The <sys/socket.h> header defines the following values for the domain argument:

- **AF_UNIX** File system pathnames.
- **AF_INET** Internet address.

The type argument specifies the socket type, which determines the semantics of communication over the socket. Socket types include:

- **SOCK_STREAM** Provides sequenced, reliable, bidirectional, connection-mode byte streams, and may provide a transmission mechanism for out-of-band data.
- **SOCK_DGRAM** Provides datagrams, which are connectionless-mode, unreliable messages of fixed maximum length.
- **SOCK_SEQPACKET** Provides sequenced, reliable, bidirectional, connection-mode transmission path for records. A record can be sent using one or more output operations and received using one or more input operations, but a single operation never transfers part of more than one record. Record boundaries are visible to the receiver via the MSG_EOR flag.

If the protocol argument is non-zero, it must specify a protocol that is supported by the address family.

RETURN VALUES
Upon successful completion, socket() returns a nonnegative integer, the socket file descriptor. Otherwise a value of −1 is returned and errno is set to indicate the error.

ERRORS
The socket() function will fail if:
socket (3XN) X/Open Networking Services Library Functions

EACCES The process does not have appropriate privileges.
EAFNOSUPPORT The implementation does not support the specified address family.
EMFILE No more file descriptors are available for this process.
ENFILE No more file descriptors are available for the system.
EPROTONOSUPPORT The protocol is not supported by the address family, or the protocol is not supported by the implementation.
EPROTOTYPE The socket type is not supported by the protocol.

The socket() function may fail if:

ENOBIFS Insufficient resources were available in the system to perform the operation.
ENOMEM Insufficient memory was available to fulfill the request.
ENOSR There were insufficient STREAMS resources available for the operation to complete.

USAGE The documentation for specific address families specify which protocols each address family supports. The documentation for specific protocols specify which socket types each protocol supports.

The application can determine if an address family is supported by trying to create a socket with domain set to the protocol in question.

ATTRIBUTES See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO accept(3XN), bind(3XN), connect(3XN), getsockname(3XN), getsockopt(3XN), listen(3XN), recv(3XN), recvfrom(3XN), recvmsg(3XN), send(3XN), sendmsg(3XN), setsockopt(3XN), shutdown(3XN), socketpair(3XN), attributes(5), in(5), socket(5)
NAME    socketpair – create a pair of connected sockets

SYNOPSIS cc [ flag ... ] file ... –lsocket –lnsl [ library ... ]
#include <sys/types.h>
#include <sys/socket.h>
int socketpair(int domain, int type, int protocol, int sv[2]);

DESCRIPTION The socketpair() library call creates an unnamed pair of connected sockets in the specified address family d, of the specified type, and using the optionally specified protocol. The descriptors used in referencing the new sockets are returned in sv[0] and sv[1]. The two sockets are indistinguishable.

RETURN VALUES socketpair() returns −1 on failure, and 0 on success.

ERRORS The call succeeds unless:
EAFNOSUPPORT The specified address family is not supported on this machine.
EMFILE Too many descriptors are in use by this process.
ENOMEM There was insufficient user memory for the operation to complete.
ENOSR There were insufficient STREAMS resources for the operation to complete.
EOPNOSUPPORT The specified protocol does not support creation of socket pairs.
EPROTONOSUPPORT The specified protocol is not supported on this machine.

ATTRIBUTES See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO pipe(2), read(2), write(2), attributes(5), socket(5)

NOTES This call is currently implemented only for the AF_UNIX address family.

modified 16 May 1997 SunOS 5.6 3N-1465
socketpair (3XN)  

NAME  
socketpair – create a pair of connected sockets

SYNOPSIS  
cc [ flag ...] file ... -lxnet [ library ...]
#include <sys/socket.h>
int socketpair(int domain, int type, int protocol, int socket_vector[2]);

DESCRIPTION  
The socketpair() function creates an unbound pair of connected sockets in a specified domain, of a specified type, under the protocol optionally specified by the protocol argument. The two sockets are identical. The file descriptors used in referencing the created sockets are returned in socket_vector[0] and socket_vector[1].

domain  
Specify the communications domain in which the sockets are to be created.

type  
Specify the type of sockets to be created.

protocol  
Specify a particular protocol to be used with the sockets. Specifying a protocol of 0 causes socketpair() to use an unspecified default protocol appropriate for the requested socket type.

socket_vector  
Specify a 2-integer array to hold the file descriptors of the created socket pair.

The type argument specifies the socket type, which determines the semantics of communications over the socket. Socket types include:

SOCK_STREAM  
Provides sequenced, reliable, bidirectional, connection-mode byte streams, and may provide a transmission mechanism for out-of-band data.

SOCK_DGRAM  
Provides datagrams, which are connectionless-mode, unreliable messages of fixed maximum length.

SOCK_SEQPACKET  
Provides sequenced, reliable, bidirectional, connection-mode transmission path for records. A record can be sent using one or more output operations and received using one or more input operations, but a single operation never transfers part of more than one record. Record boundaries are visible to the receiver via the MSG_EOR flag.

If the protocol argument is non-zero, it must specify a protocol that is supported by the address family.

RETURN VALUES  
Upon successful completion, this function returns 0. Otherwise, −1 is returned and errno is set to indicate the error.

ERRORS  
The socketpair() function will fail if:

EAFNOSUPPORT  
The implementation does not support the specified address family.

EMFILE  
No more file descriptors are available for this process.

3XN-1466  SunOS 5.6  modified 16 May 1997
ENFILE  No more file descriptors are available for the system.
EOPNOTSUPP The specified protocol does not permit creation of socket pairs.
EPROTOONOSUPPORT
   The protocol is not supported by the address family, or the protocol is
   not supported by the implementation.
EPROTOTYPE  The socket type is not supported by the protocol.
The socketpair() function may fail if:
EACCES  The process does not have appropriate privileges.
ENOMEM  Insufficient memory was available to fulfill the request.
ENOBUFS  Insufficient resources were available in the system to perform the opera-
         tion.
ENOSR  There were insufficient STREAMS resources available for the operation to
         complete.

USAGE  The documentation for specific address families specifies which protocols each address
        family supports. The documentation for specific protocols specifies which socket types
        each protocol supports.
The socketpair() function supports only UNIX domain sockets.

ATTRIBUTES  See attributes(5) for descriptions of the following attributes:

      ATTRIBUTE TYPE     ATTRIBUTE VALUE
       MT-Level        MT-Safe

SEE ALSO  socket(3XN), attributes(5), socket(5)
NAME       spray – scatter data in order to test the network

SYNOPSIS   cc [ flag ...] file ... -lsoclet -lnsl [ library ... ]
            #include <rpcsvc/spray.h>
            bool_t xdr_sprayarr(XDR *xdrs, sprayarr *objp);
            bool_t xdr_spraycumul(XDR *xdrs, spraycumul *objp);

DESCRIPTION The spray program sends packets to a given machine to test communications with that machine.

The spray program is not a C function interface, per se, but can be accessed using the generic remote procedure calling interface clnt_call() (see rpc_clnt_calls(3N)). 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.

The spray program currently supports the following procedures, which should be called in the order given:

**SPRAYPROC_CLEAR**
   This procedure clears the counter.

**SPRAYPROC_SPRAY**
   This procedure sends the packet.

**SPRAYPROC_GET**
   This procedure returns the count and the amount of time since the last SPRAYPROC_CLEAR.

EXAMPLES The following code fragment demonstrates how the spray program is used:

```c
#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", SPRAYPROG, SPRAYVERS, "netpath");
if (clnt == (CLIENT *)NULL) {
    /* handle this error */
```

3N-1468 SunOS 5.6 modified 30 Dec 1996
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);

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO spray(1M), rpc_clnt_calls(3N), attributes(5)

NOTES

This interface is unsafe in multithreaded applications. Unsafe interfaces should be called only from the main thread.

A spray program is not useful as a networking benchmark as it uses unreliable connectionless transports, (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 – square root function

SYNOPSIS  cc [ flag ... ] file ... -lm [ library ... ]
#include <math.h>
double sqrt(double x);

DESCRIPTION  The sqrt() function computes the square root of x.

RETURN VALUES  Upon successful completion, sqrt() returns the square root of x.
If x is NaN, NaN is returned.
If x is negative, NaN is returned and errno is set to EDOM.

ERRORS  The sqrt() function will fail if:
EDOM   The value of x is negative.

USAGE  An application wishing to check for error situations should set errno to 0 before calling sqrt(). If errno is non-zero on return, or the return value is NaN, an error has occurred.

ATTRIBUTES  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  isnan(3M), attributes(5)
NAME
SSAAgentIsAlive, SSAGetTrapPort, SSARegSubtable, SSARegSubagent, SSARegSubtree, SSASendTrap, SSASubagentOpen – Sun Solstice Enterprise Agent registration and communication helper functions

SYNOPSIS
cc [ flag ... ] file ... -lssagent -lssasnmp [ library ... ]
#include <impl.h>
extern int SSAAgentIsAlive(IPAddress *agent_addr, int *port, char *community, struct timeval *timeout);
extern int SSAGetTrapPort();
extern int *SSARegSubagent(Agent *agent);
int SSARegSubtable(SSA_Table *table);
int SSARegSubtree(SSA_Subtree *subtree);
extern void SSASendTrap(char *name);
extern int SSASubagentOpen(int *num_of_retry, char *agent_name);

DESCRIPTION
The SSAAgentIsAlive() function returns TRUE if the master agent is alive, otherwise returns FALSE. The agent_addr parameter is the address of the agent. Specify the security token in the community parameter. You can specify the maximum amount of time to wait for a response with the timeout parameter.

The SSAGetTrapPort() function returns the port number used by the Master Agent to communicate with the subagent.

The SSARegSubagent() function enables a subagent to register and unregister with a Master Agent. The agent parameter is a pointer to an Agent structure containing the following members:

int timeout; /* optional */
int agent_id; /* required */
int agent_status; /* required */
char *personal_file; /* optional */
char *config_file; /* optional */
char *executable; /* optional */
char *version_string; /* optional */
char *protocol; /* optional */
int process_id; /* optional */
char *name; /* optional */
int system_up_time; /* optional */
int watch_dog_time; /* optional */
Address address; /* required */
struct _Agent; /* reserved */
struct _Subtree; /* reserved */

modified 17 Dec 1996 SunOS 5.6 3X-1471
The `agent_id` member is an integer value returned by the **SSASubagentOpen()** function. After calling `SSASubagentOpen()`, you pass the `agent_id` in the **SSARegSubagent()** call to register the subagent with the Master Agent.

The following values are supported for `agent_status`:

- `SSA_OPER_STATUS_ACTIVE`
- `SSA_OPER_STATUS_NOT_IN_SERVICE`
- `SSA_OPER_STATUS_DESTROY`

You pass `SSA_OPER_STATUS_DESTROY` as the value in a **SSARegSubagent()** function call when you want to unregister the agent from the Master Agent.

**Address** has the same structure as `sockaddr_in`, that is a common UNIX structure containing the following members:

- `short sin_family;`
- `u_short sin_port;`
- `struct in_addr sin_addr;`
- `char sin_zero[8];`

The **SSARegSubtable()** function registers a MIB table with the Master Agent. If this function is successful, an index number is returned, otherwise 0 is returned. The `table` parameter is a pointer to a **SSA_Table** structure containing the following members:

- `int regTblIndex; /* index value */`
- `int regTblAgentID; /* current agent ID */`
- `Oid regTblOID; /* Object ID of the table */`
- `int regTblStartColumn; /* start column index */`
- `int regTblEndColumn; /* end column index */`
- `int regTblStartRow; /* start row index */`
- `int regTblEndRow; /* end row index */`
- `int regTblStatus; /* status */`

The `regTblStatus` can have one of the following values:

- `SSA_OPER_STATUS_ACTIVE`
- `SSA_OPER_STATUS_NOT_IN_SERVICE`

The **SSARegSubtree()** function registers a MIB subtree with the master agent. If successful this function returns an index number, otherwise 0 is returned. The `subtree` parameter is a pointer to a **SSA_Subtree** structure containing the following members:

- `int regTreeIndex; /* index value */`
- `int regTreeAgentID; /* current agent ID */`
- `Oid name; /* Object ID to register */`
- `int regtreeStatus; /* status */`
The regtreeStatus can have one of the following values:

SSA_OPER_STATUS_ACTIVE
SSA_OPER_STATUS_NOT_IN_SERVICE

The SSASendTrap() function instructs the Master Agent to send a trap notification, based on the keyword passed with name. When your subagent MIB is compiled by mib-codegen, it creates a lookup table of the trap notifications defined in the MIB. By passing the name of the trap notification type as name, the subagent instructs the Master Agent to construct the type of trap defined in the MIB.

The SSASubagentOpen() function initializes communication between the subagent and the Master Agent. You must call this function before calling SSARegSubagent() to register the subagent with the Master Agent. The SSASubagentOpen() function returns a unique agent ID that is passed in the SSARegSubagent() call to register the subagent. If 0 is returned as the agent ID, the attempt to initialize communication with the Master Agent was unsuccessful. Since UDP is used to initialize communication with the Master Agent, you may want to set the value of num_of_retry to make multiple attempts.

The value for agent_name must be unique within the domain for which the Master Agent is responsible.

**ATTRIBUTES**

See attributes (5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

**SEE ALSO** attributes (5)
NAME

SSAOidCmp, SSAOidCpy, SSAOidDup, SSAOidFree, SSAOidInit, SSAOidNew, SSAOidString, SSAOidStrToOid, SSAOidZero – Sun Solstice Enterprise Agent OID helper functions

SYNOPSIS

cc [ flag ... ] file ... -lssasnmp [ library ... ]

#include <impl.h>

int SSAOidCmp(Oid *oid1, Oid *oid2);
int SSAOidCpy(Oid *oid1, Oid *oid2, char *error_label);
Oid *SSAOidDup(Oid *oid, char *error_label);
void SSAOidFree(Oid *oid);
int SSAOidInit(Oid *oid, Subid *subids, int len, char *error_label);
Oid *SSAOidNew();
char *SSAOidString(Oid *oid);
Oid *SSAOidStrToOid(char* name, char *error_label);
void SSAOidZero(Oid *oid);

DESCRIPTION

The SSAOidCmp() function performs a comparison of the given OIDs. This function returns:

0 if oid1 is equal to oid2
1 if oid1 is greater than oid2
−1 if oid1 is less than oid2

The SSAOidCpy() function makes a deep copy of oid2 to oid1. This function assumes oid1 has been processed by the SSAOidZero() function. Memory is allocated inside oid1 and the contents of oid2, not just the pointer, is copied to oid1. If an error is encountered, an error message is stored in the error_label buffer.

The SSAOidDup() function returns a clone of oid, by using the deep copy. Error information is stored in the error_label buffer.

The SSAOidFree() function frees the OID instance, with its content.

The SSAOidNew() function returns a new OID.

The SSAOidInit() function copies the Subid array from subids to the OID instance with the specified length len. This function assumes that the OID instance has been processed by the SSAOidZero() function or no memory is allocated inside the OID instance. If an error is encountered, an error message is stored in the error_label buffer.

The SSAOidString() function returns a char pointer for the printable form of the given oid.

The SSAOidStrToOid() function returns a new OID instance from name. If an error is encountered, an error message is stored in the error_label buffer.
The `SSAOidZero()` function frees the memory used by the OID object for buffers, but not the OID instance itself.

**RETURN VALUES**

The `SSAOidNew()` and `SSAOidStrToOid()` functions return 0 if an error is detected.

**ATTRIBUTES**

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

**SEE ALSO**

`attributes(5)`

modified 17 Dec 1996

SunOS 5.6

3X-1475
NAME
SSAStringCpy, SSAStringInit, SSAStringToChar, SSAStringZero – Sun Solstice Enterprise Agent string helper functions

SYNOPSIS
cc [ flag ... ] file ... -lssasnmp [ library ... ]
#include <impl.h>
void *SSAStringZero(String *string);
int SSAStringInit(String *string, u_char *chars, int len, char *error_label);
int SSAStringCpy(String *string1, String *string2, char *error_label);
char *SSAStringToChar(String string);

DESCRIPTION
The SSAStringCpy() function makes a deep copy of string2 to string1. This function assumes that string1 has been processed by the SSAStringZero() function. Memory is allocated inside the string1 and the contents of string2, not just the pointer, is copied to the string1. If an error is encountered, an error message is stored in the error_label buffer.

The SSAStringInit() function copies the char array from chars to the string instance with the specified length len. This function assumes that the string instance has been processed by the SSAStringZero() function or no memory is allocated inside the string instance. If an error is encountered, an error message is stored in the error_label buffer.

The SSAStringToChar() function returns a temporary char array buffer for printing purposes.

The SSAStringZero() function frees the memory inside of the String instance, but not the string object itself.

RETURN VALUES
The SSAStringInit() and SSAStringCpy() functions return 0 if successful and −1 if error.

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
attributes(5)
NAME  
ssignal, gsignal – software signals

SYNOPSIS  
#include <signal.h>

void (*ssignal (int sig, int (*action)(int)))(int);

int gsignal(int sig);

DESCRIPTION  
ssignal() and gsignal() implement a software facility similar to signal(3C). This facility is made available to users for their own purposes.

Software signals made available to users are associated with integers in the inclusive range 1 through 17. A call to ssignal() associates a procedure, action, with the software signal sig; the software signal, sig, is raised by a call to gsignal(). Raising a software signal causes the action established for that signal to be taken.

The first argument to ssignal() is a number identifying the type of signal for which an action is to be established. The second argument defines the action; it is either the name of a (user-defined) action function or one of the manifest constants SIG_DFL (default) or SIG_IGN (ignore). 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.

gsignal() raises the signal identified by its argument, sig:

If an action function has been established for sig, then that action is reset to SIG_DFL and the action function is entered with argument sig. gsignal() returns the value returned to it by the action function.

If the action for sig is SIG_IGN, gsignal() returns the value 1 and takes no other action.

If the action for sig is SIG_DFL, gsignal() returns the value 0 and takes no other action.

If sig has an illegal value or no action was ever specified for sig, gsignal() returns the value 0 and takes no other action.

ATTRIBUTES  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO  
raise(3C), signal(3C), attributes(5)
<table>
<thead>
<tr>
<th>NAME</th>
<th>standend, standout, wstandend, wstandout – set/clear window attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYNOPSIS</td>
<td>#include &lt;curses.h&gt;</td>
</tr>
<tr>
<td></td>
<td>int standend(void);</td>
</tr>
<tr>
<td></td>
<td>int standout(void);</td>
</tr>
<tr>
<td></td>
<td>int wstandend(WINDOW *win);</td>
</tr>
<tr>
<td></td>
<td>int wstandout(WINDOW *win);</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>win Is a pointer to the window in which attribute changes are to be made.</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>The standend() and wstandend() functions turn off all attributes associated with stdscr and win respectively.</td>
</tr>
<tr>
<td></td>
<td>The standout() and wstandout() functions turn on the A_STANDOUT attribute of stdscr and win respectively.</td>
</tr>
<tr>
<td>RETURN VALUES</td>
<td>These functions always return 1.</td>
</tr>
<tr>
<td>ERRORS</td>
<td>None.</td>
</tr>
<tr>
<td>SEE ALSO</td>
<td>attr_get(3XC), attroff(3XC)</td>
</tr>
</tbody>
</table>
NAME
stdio – standard buffered input/output package

SYNOPSIS
#include <stdio.h>
extern FILE *stdin;
extern FILE *stdout;
extern FILE *stderr;

DESCRIPTION
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(), gets(), getw(), printf(), puts(), putw(), and scanf() all use or act as if they use getc() and putc(); they can be freely intermixed.

A file with associated buffering is called a stream (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:

stdin standard input file
stdout standard output file
stderr standard error file

The following symbolic values in <unistd.h> define the file descriptors that will be associated with the C-language stdin, stdout and stderr when the application is started:

STDIN_FILENO Standard input value 0 stdin
STDOUT_FILENO Standard output value 1 stdout
STDERR_FILENO Standard error value 2 stderr

The constant NULL designates a null pointer.

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).

The integer constant BUFSIZ specifies the size of the buffers used by the particular implementation.

The integer constant FILENAME_MAX specifies the number of bytes needed to hold the longest pathname of a file allowed by the implementation. If the system does not impose a maximum limit, this value is the recommended size for a buffer intended to hold a file’s pathname.

The integer constant FOPEN_MAX specifies the minimum number of files that the implementation guarantees can be open simultaneously. Note that no more than 255 files may be opened using fopen(), and only file descriptors 0 through 255 can be used in a stream.

The functions and constants mentioned in the entries of section 3S of this manual are declared in that header and need no further declaration. The constants and the following “functions” are implemented as macros (redeclaration of these names is perilous): getc(),
stdio (3S) Standard I/O Functions

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 `flush()` 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 `flush()` 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:

```c
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:

```c
flockfile(iop);
while (!feof(iop)) {
    *c++ = getc_unlocked(iop);
}
funlockfile(iop);
```

RETURN VALUES

Invalid stream pointers usually cause grave disorder, possibly including program termination. Individual function descriptions describe the possible error conditions.

SEE ALSO

close(2), lseek(2), open(2), pipe(2), read(2), write(2), ctermid(3S), cuserid(3S), fclose(3S), ferror(3S), fopen(3S), fread(3S), fseek(3S), flockfile(3S), getc(3S), gets(3S), popen(3S), printf(3S), putc(3S), puts(3S), scanf(3S), setbuf(3S), system(3S), tmpfile(3S), tmpnam(3S), ungetc(3S)
NAME
str2sig, sig2str – translation between signal name and signal number

SYNOPSIS
#include <signal.h>
int str2sig(const char *str, int *signum);
int sig2str(int signum, char *str);

DESCRIPTION
The str2sig() function translates the signal name str to a signal number, and stores that result in the location referenced by signum. The name in str 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".

The sig2str() function translates the signal number signum to the symbol for that signal, without the "SIG" prefix, and stores that symbol at the location specified by str. The storage referenced by str 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.

RETURN VALUES
The str2sig() function returns 0 if it recognizes the signal name specified in str; otherwise, it returns −1.

The sig2str() function returns 0 if the value signum corresponds to a valid signal number; otherwise, it returns −1.

EXAMPLES
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 */

SEE ALSO
kill(1), strsignal(3C)

modified 2 March 1994 SunOS 5.6 3C-1483
NAME
strccpy, streadd, strcadd, strecpy – copy strings, compressing or expanding escape codes

SYNOPSIS
cc [ flag ...] file ... -lgen [ library ...]
#include <libgen.h>

char *strccpy(char *output, const char *input);
char *strecpy(char *output, const char *input);
char *streadd(char *output, const char *input, const char *exceptions);
char *streadd(char *output, const char *input, const char *exceptions);

DESCRIPTION
strccpy() copies the input string, up to a null byte, to the output string, compressing the
C-language escape sequences (for example, \n, \001) to the equivalent character. A null
byte is appended to the output. The output argument must point to a space big enough to
accommodate the result. If it is as big as the space pointed to by input it is guaranteed to
be big enough. strccpy() returns the output argument.

strcadd() is identical to strccpy(), except that it returns the pointer to the null byte that
terminates the output.

strecpy() copies the input string, up to a null byte, to the output string, expanding non-
graphic characters to their equivalent C-language escape sequences (for example, \n,
\001). The output argument must point to a space big enough to accommodate the result;
four times the space pointed to by input is guaranteed to be big enough (each character
could become \ and 3 digits). Characters in the exceptions string are not expanded. The
exceptions argument may be zero, meaning all non-graphic characters are expanded.

strecpy() returns the output argument.

streadd() is identical to strecpy(), except that it returns the pointer to the null byte that
terminates the output.

EXAMPLES
/* expand all but newline and tab */
streccpy(output, input, "\n\t");

/* concatenate and compress several strings */
cp = streadd(output, input1);
cp = streadd(cp, input2);
cp = streadd(cp, input3);

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
string(3C), strfind(3G), attributes(5)
NOTES

When compiling multi-thread applications, the _REENTRANT flag must be defined on the compile line. This flag should only be used in multi-thread applications.
NAME    strcoll – string collation

SYNOPSIS #include <string.h>

    int strcoll(const char *s1, const char *s2);

DESCRIPTION Upon successful completion, strcoll() returns an integer greater than, equal to, or less
than zero in direct correlation to whether string s1 is greater than, equal to, or less than
the string s2. The comparison is based on strings interpreted as appropriate to the
program’s locale for category LC_COLLATE (see setlocale(3C)).

On error, strcoll() may set errno, but no return value is reserved to indicate an error.

Both strcoll() and strxfrm(3C) provide for locale-specific string sorting. strcoll() is
intended for applications in which the number of comparisons per string is small. When
strings are to be compared a number of times, strxfrm(3C) is a more appropriate function
because the transformation process occurs only once.

ERRORS The strcoll() function may fail if the following is detected:

    EINVAL   The s1 or s2 arguments contain characters outside the domain of the
collating sequence.

FILES /usr/lib/locale/locale/locale.so.*   LC_COLLATE database for locale

ATTRIBUTES See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe with exceptions</td>
</tr>
<tr>
<td></td>
<td>Enabled</td>
</tr>
</tbody>
</table>

SEE ALSO localedef(1), setlocale(3C), string(3C), strxfrm(3C), wsxfrm(3C), attributes(5),
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>

cchar *strerror(int errnum);

DESCRIPTION  
strerror() maps the error number in errnum 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 errnum is out-of-range.

ATTRIBUTES  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  
gettext(3C), perror(3C), setlocale(3C), attributes(5)

NOTES  
If the application is linked with -lintl, then messages returned from this function are in
the native language specified by the LC_MESSAGES locale category; see setlocale(3C).
NAME  strfind, strrspn, strtrns, str – string manipulations

SYNOPSIS  cc [ flag ... ] file ... -Igen [ library ... ]
#include <libgen.h>
int strfind(const char *as1, const char *as2);
char *strrspn(const char *string, const char *tc);
char *strtrns(const char *string, const char *old, const char *new, char *result);

DESCRIPTION  strfind() returns the offset of the first occurrence of the second string, as2, if it is a substring of string as1. If the second string is not a substring of the first string strfind() returns -1.

strrspn() returns a pointer to the first character in the string that is not one of the characters in tc.

strtrns() transforms string and copies it into result. Any character that appears in old is replaced with the character in the same position in new. The new result is returned.

EXAMPLES  /* find offset to substring "hello" within as1 */
i = strfind(as1, "hello");
/* trim junk from end of string */
s2 = strrspn(s1, "*?#$%");
*s2 = '\0';
/* transform lower case to upper case */
a1[] = "abcdefghijklmnopqrstuvwxyz"
    = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
a2[] = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
s2 = strtrns(s1, a1, a2, s2);

ATTRIBUTES  See attributes(5) for descriptions of the following attributes:

+----------------+------------------+
| ATTRIBUTE TYPE  | ATTRIBUTE VALUE  |
|----------------+------------------|
| MT-Level       | MT-Safe          |
+----------------+------------------+

SEE ALSO  string(3C), attributes(5)

NOTES  When compiling multi-thread applications, the _REENTRANT flag must be defined on the compile line. This flag should only be used in multi-thread applications.
strfmon(3C)

NAME

strfmon – convert monetary value to string

SYNOPSIS

#include <monetary.h>

ssize_t strfmon(char *s, size_t maxsize, const char *format, ...);

DESCRIPTION

The strfmon() function places characters into the array pointed to by s as controlled by
the string pointed to by format. No more than maxsize 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.

A conversion specification consists of the following sequence:

- 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

One or more of the following optional flags can be specified to control the conversion:

- =f An = followed by a single character f 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 char-
acter. 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.

- Do not format the currency amount with grouping characters. The default is to
insert the grouping characters if defined for the current locale.

- + 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.

- ! Suppress the currency symbol from the output conversion.

- Specify the alignment. If this flag is present all fields are left-justified (padded to
the right) rather than right-justified.

Field Width

A decimal digit string w specifying a minimum field width in bytes in which the
result of the conversion is right-justified (or left-justified if the flag ‘−’ is
specified). The default is zero.
### Left Precision

`#n`  A `'#'` followed by a decimal digit string `n` 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 `n`. If more than `n` 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 `=f` flag above).

If grouping has not been suppressed with the `’‘` flag, and it is defined for the current locale, grouping separators are inserted before the fill characters (if any) are added. Grouping separators are not applied to fill characters even if the fill character is a digit.

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.

### Right Precision

`.p`  A period followed by a decimal digit string `p` specifying the number of digits after the radix character. If the value of the right precision `p` 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

<table>
<thead>
<tr>
<th>Character</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>i</code></td>
<td>The <code>double</code> argument is formatted according to the locale’s international currency format (for example, in the U.S.A.: USD 1,234.56).</td>
</tr>
<tr>
<td><code>n</code></td>
<td>The <code>double</code> argument is formatted according to the locale’s national currency format (for example, in the U.S.A.: $1,234.56).</td>
</tr>
<tr>
<td><code>%</code></td>
<td>Convert to a <code>%</code>; no argument is converted. The entire conversion specification must be <code>%%</code>.</td>
</tr>
</tbody>
</table>

### Locale Information

The `LC_MONETARY` category of the program’s locale affects the behavior of this function including the monetary radix character (which may be different from the numeric radix character affected by the `LC_NUMERIC` category), the grouping separator, the currency symbols and formats. The international currency symbol should be in conformance with the ISO 4217: 1987 standard.

### RETURN VALUES

If the total number of resulting bytes (including the terminating null byte) is not more than `maxsize`, `strfmon()` returns the number of bytes placed into the array pointed to by `s`, not including the terminating null byte. Otherwise, `−1` is returned, the contents of the array are indeterminate, and `errno` is set to indicate the error.
### ERRORS

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:

<table>
<thead>
<tr>
<th>Conversion Specification</th>
<th>Output</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>%n</td>
<td>$123.45</td>
<td>default formatting</td>
</tr>
<tr>
<td></td>
<td>-$123.45</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$3,456.78</td>
<td></td>
</tr>
<tr>
<td>%11n</td>
<td>$123.45</td>
<td>right align within an 11 character field</td>
</tr>
<tr>
<td></td>
<td>-$123.45</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$3,456.78</td>
<td></td>
</tr>
<tr>
<td>%#5n</td>
<td>$ 123.45</td>
<td>aligned columns for values up to 99,999</td>
</tr>
<tr>
<td></td>
<td>-$ 123.45</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$ 3,456.78</td>
<td></td>
</tr>
<tr>
<td>%=##5n</td>
<td>$***123.45</td>
<td>specify a fill character</td>
</tr>
<tr>
<td></td>
<td>-***123.45</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$*3,456.78</td>
<td></td>
</tr>
<tr>
<td>%=0#5n</td>
<td>$000123.45</td>
<td>fill characters do not use grouping</td>
</tr>
<tr>
<td></td>
<td>-$000123.45</td>
<td>even if the fill character is a digit</td>
</tr>
<tr>
<td></td>
<td>$03,456.78</td>
<td></td>
</tr>
<tr>
<td>%*#5n</td>
<td>$ 123.45</td>
<td>disable the grouping separator</td>
</tr>
<tr>
<td></td>
<td>-$ 123.45</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$ 3456.78</td>
<td></td>
</tr>
<tr>
<td>%*5.0n</td>
<td>$ 123</td>
<td>round off to whole units</td>
</tr>
<tr>
<td></td>
<td>-$ 123</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$ 3457</td>
<td></td>
</tr>
<tr>
<td>%*5.4n</td>
<td>$ 123.4500</td>
<td>increase the precision</td>
</tr>
<tr>
<td></td>
<td>-$ 123.4500</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$ 3456.7810</td>
<td></td>
</tr>
<tr>
<td>%(#5n</td>
<td>123.45</td>
<td>use an alternative pos/neg style</td>
</tr>
<tr>
<td></td>
<td>$( 123.45</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$ 3,456.78</td>
<td></td>
</tr>
<tr>
<td>%!(#5n</td>
<td>123.45</td>
<td>disable the currency symbol</td>
</tr>
<tr>
<td></td>
<td>( 123.45</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3,456.78</td>
<td></td>
</tr>
</tbody>
</table>
ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe with exceptions</td>
</tr>
<tr>
<td>CSI</td>
<td>Enabled</td>
</tr>
</tbody>
</table>

SEE ALSO
localeconv(3C), setlocale(3C), attributes(5)

NOTES
This function can be used safely in multi-thread applications, as long as setlocale(3C) is not called to change the locale.
NAME  strftime, cftime, asctime – convert date and time to string

SYNOPSIS  
#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);

DESCRIPTION  
strftime(), asctime(), and cftime() place bytes into the array pointed to by s as controlled by the string pointed to by format. The format 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 s. If copying takes place between objects that overlap, the behavior is undefined. For strftime(), no more than maxsize bytes are placed into the array.

If format is (char *), 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 CTIME, and if that is undefined or empty, the default format is used.

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 timeptr for strftime() and asctime(), and by the time represented by clock for cftime().

%%    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
%d    day of month [1,31]; single digits are preceded by 0
%e    day of month [1,31]; single digits are preceded by a space
%H    hour (24-hour clock) [0,23]; single digits are preceded by 0
%I    hour (12-hour clock) [1,12]; single digits are preceded by 0
%j    day number of year [1,366]; single digits are preceded by 0
%k    hour (24-hour clock) [0,23]; single digits are preceded by a blank
%l    hour (12-hour clock) [1,12]; single digits are preceded by a blank
%m    month number [1,12]; single digits are preceded by 0
%n    new line
%p    standard time or daylight saving time
%S    seconds (100); single digits are preceded by a space
%t    character and one or two terminating conversion characters
%T    hour (24-hour clock) [0,23]; single digits are preceded by 0
%U    day of year [1,366]; single digits are preceded by 0
%u    UTC time
%v    new line
%V    year as produced by date
%w    locale’s abbreviated weekday name
%W    locale’s full weekday name
%y    last two digits of the year [00,99]; single digits are preceded by 0
%Y    year [0000,9999]; single digits are preceded by 0
%z    time zone
%Z    time zone

Default
Standard-conforming

modified 26 Feb 1997

SunOS 5.6

3C-1493
%M minute [00,59]; 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 [00,61]
%t insert a tab
%T time as %H:%M:%S
%u weekday as a decimal number [1,7], with 1 representing Sunday
%U week number of year as a decimal number [00,53], with Sunday as the first day of week 1
%V 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 [00,53], with Monday as the first day of week 1
%x locale’s appropriate date representation
%x locale’s appropriate time representation
%y year within century [00,99]
%Y year, including the century (for example 1993)
%Z time zone name or abbreviation, or no bytes if no time zone information exists

If a conversion specification does not correspond to any of the above or to any of the modified conversion specifications listed below, the behavior is undefined and 0 is returned.

The difference between %U and %W (and also between modified conversion specifications %OU and %OW) lies in which day is counted as the first of the week. Week number 1 is the first week in January starting with a Sunday for %U or a Monday for %W. Week number 0 contains those days before the first Sunday or Monday in January for %U and %W, respectively.

**Modified Conversion Specifications**

Some conversion specifications can be modified by the E and O modifiers to indicate that an alternate format or specification should be used rather than the one normally used by the unmodified conversion specification. If the alternate format or specification does not exist in the current locale, the behavior will be as if the unmodified specification were used.

%Ec locale’s alternate appropriate date and time representation
%EC name of the base year (period) in the locale’s alternate representation
%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  weekday as a number in the locale’s alternate numeric symbols
%OU  week number of the year (Sunday as the first day of the week) using the locale’s alternate numeric symbols
%Ow  number of the weekday (Sunday=0) using the locale’s alternate numeric symbols
%OW  week number of the year (Monday as the first day of the week) using the locale’s alternate numeric symbols
%Oy  year (offset from %C) in the locale’s alternate representation and using the locale’s alternate numeric symbols

Selecting the Output Language
By default, the output of strftime(), cftime(), and asctime() appear in U.S. English. The user can request that the output of strftime(), cftime(), or asctime() be in a specific language by setting the LC_TIME category using setlocale().

Time Zone
Local time zone information is used as though tzset(3C) were called.

RETURN VALUES
strftime(), cftime(), and asctime() 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() for the POSIX locale. It shows what the string in str would look like if the structure pointed to by tmptr contains the values corresponding to Thursday, August 28, 1986 at 12:44:36.

```
strftime (str, strsize, "%A %b %d %j", tmptr)
```

This results in str containing "Thursday Aug 28 240".

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
<tr>
<td>CSI</td>
<td>Enabled</td>
</tr>
</tbody>
</table>

SEE ALSO
date(1), ctime(3C), mktime(3C), setlocale(3C), strftime(3C), tzset(3C), TIMEZONE(4), attributes(5), environ(5), standards(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.
NAME

string, strcasecmp, strncasecmp, strcat, strchr, strrchr, strcmp, strncmp, strcpy,
strncpy, strcspn, strspn, strdup, strlen, strpbrk, strstr, strtok, strtok_r – string operations

SYNOPSIS

#include <strings.h>

int strcasecmp(const char *s1, const char *s2);
int strncasecmp(const char *s1, const char *s2, int n);

#include <string.h>

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);

description

The arguments s, s1, s2, src, and dst 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.

strcasecmp() and strncasecmp() are case-insensitive versions of strcmp() and strncmp()
respectively, described below. strcasecmp() and strncasecmp() assume the ASCII char-
acter set and ignore differences in case when comparing lower and upper case characters.

strcat() appends a copy of string src, including the terminating null character, to the end
of string dst. strncat() appends at most n characters. Each returns a pointer to the null-
terminated result. The initial character of src overrides the null character at the end of
dst.

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. **strn cmp()** 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.
ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>See NOTES below.</td>
</tr>
</tbody>
</table>

SEE ALSO  

malloc(3C), setlocale(3C), strxfrm(3C), attributes(5)

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(), strlen(), strncasecmp(), 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
#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);

DESCRIPTION
The char_to_decimal functions parse a numeric token from at most nmax characters in a string **pc or file *pf or function (*pget)() into a decimal record *pd, classifying the form of the string in *pform and *pechar. The accepted syntax is intended to be sufficiently flexible to accommodate many languages:

whitespace value

or

whitespace sign value

where whitespace is any number of characters defined by isspace in <ctype.h>, sign is either of [+−], and value can be number, nan, or inf. inf can be INF (inf_form) or INFINITY (infinity_form) without regard to case. nan can be NAN (nan_form) or NAN(nstring) (nanstring_form) without regard to case; nstring is any string of characters not containing ’ ’ or NULL; nstring is copied to pd->ds and, currently, not used subsequently. number consists of

  significand

or

  significand efield

where significand must contain one or more digits and may contain one point; possible forms are

digits (int_form)
digits. (intdot_form)
digits.digits (dotfrac_form)
digits.digits.digits (intdoffrac_form)

efield consists of
echar digits
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, frac-
tion, 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 >= 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 charac-
ters 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 accord-
ingly. If all unget calls are successful, then **pc will be NULL. No push back will be
attempted if (*punget) is NULL.

3-1500 SunOS 5.6 modified 29 Dec 1996
Typical declarations for *pget() and *punget() are:

```c
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()`.

**ATTRIBUTES**

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**SEE ALSO**

`ctype(3C)`, `localeconv(3C)`, `scanf(3S)`, `setlocale(3C)`, `strtod(3C)`, `ungetc(3S)`, attributes(5)
NAME
strptime – date and time conversion

SYNOPSIS
#include <time.h>
char *strptime(const char *buf, const char *format, struct tm *tm);

DESCRIPTION
The strptime() function converts the character string pointed to by buf to values which are stored in the tm structure pointed to by tm, using the format specified by format. format 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.

The following conversion specifications are supported:

- %% 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
- %k 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
- %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
### Modified Conversion Specifications

Some conversion specifications can be modified by the `E` and `O` modifier characters to indicate that an alternate format or specification should be used rather than the one normally used by the unmodified specification. If the alternate format or specification does not exist in the current locale, the behavior will be as if the unmodified conversion specification were used.

- `%Ec` locale’s alternate appropriate date and time representation
- `%EC` name of the base year (era) in the locale’s alternate representation
- `%Ex` locale’s alternate date representation
- `%EX` locale’s alternate time representation
- `%Ey` offset from `%EC` (year only) in the locale’s alternate representation
- `%EY` full alternate year representation
- `%Od` day of the month using the locale’s alternate numeric symbols
- `%Oe` same as `%Od`
- `%OH` hour (24-hour clock) using the locale’s alternate numeric symbols
- `%OI` hour (12-hour clock) using the locale’s alternate numeric symbols
- `%Om` month using the locale’s alternate numeric symbols
- `%OS` minutes 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 infor-
mation. 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 con-
sist 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()`.

RETURN VALUES
Upon successful completion, `strptime()` returns a pointer to the character following the
last character parsed. Otherwise, a null pointer is returned.

ATTRIBUTES
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe Enabled</td>
</tr>
</tbody>
</table>

SEE ALSO
`ctime(3C)`, `getdate(3C)`, `isspace(3C)`, `setlocale(3C)`, `strftime(3C)`, `attributes(5)`

NOTES
Several “same as” formats, and the special processing of white-space characters are pro-
vided 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.

For `%Z`, local timezone information is used as though `strptime()` called `tzset()` (see
`ctime(3C)`). Errors may not be detected. This behavior is subject to change in a future
release.
NAME  strsignal – get error message string

SYNOPSIS  
```
#include <string.h>
char *strsignal(int sig);
```

DESCRIPTION  strsignal() maps the signal number in sig 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 sig is not a valid signal number.

ATTRIBUTES  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  gettext(3C), psignal(3C), setlocale(3C), str2sig(3C), attributes(5)

NOTES  If the application is linked with -lintl, then messages returned from this function are in the native language specified by the LC_MESSAGES locale category; see setlocale(3C).
NAME

strtod, atof – convert string to double-precision number

SYNOPSIS

#include <stdlib.h>

double strtod(const char *str, char **endptr);

double atof(const char *str);

DESCRIPTION

The `strtod()` function converts the initial portion of the string pointed to by `str` to type `double` representation. First it decomposes the input string into three parts: an initial, possibly empty, sequence of white-space characters (as specified by `isspace(3C)`); a subject sequence interpreted as a floating-point constant; and a final string of one or more unrecognized characters, including the terminating null byte of the input string. Then it attempts to convert the subject sequence to a floating-point number, and returns the result.

The expected form of the subject sequence is an optional `+` or `−` sign, then a non-empty sequence of digits optionally containing a radix character, then an optional exponent part. An exponent part consists of `e` or `E`, followed by an optional sign, followed by one or more decimal digits. The subject sequence is defined as the longest initial subsequence of the input string, starting with the first non-white-space character, that is of the expected form. The subject sequence is empty if the input string is empty or consists entirely of white-space characters, or if the first character that is not white space is other than a sign, a digit or a radix character.

If the subject sequence has the expected form, the sequence starting with the first digit or the radix character (whichever occurs first) is interpreted as a floating constant of the C language, except that the radix character is used in place of a period, and that if neither an exponent part nor a radix character appears, a radix character is assumed to follow the last digit in the string. If the subject sequence begins with a minus sign, the value resulting from the conversion is negated. A pointer to the final string is stored in the object pointed to by `endptr`, provided that `endptr` is not a null pointer.

The radix character is defined in the program’s locale (category `LC_NUMERIC`). In the POSIX locale, or in a locale where the radix character is not defined, the radix character defaults to a period (.).

In other than the POSIX locale, other implementation-dependent subject sequence forms may be accepted.

If the subject sequence is empty or does not have the expected form, no conversion is performed; the value of `str` is stored in the object pointed to by `endptr`, provided that `endptr` is not a null pointer.

`atof()`

`atof(str)` is equivalent to `strtod(str, (char **)NULL)`.

RETURN VALUES

Upon successful completion, `strtod()` returns the converted value. If no conversion could be performed, 0 is returned, and `errno` may be set to `EINVAL`.

`atof()`

...
If the correct value is outside the range of representable values, ±HUGE is returned (according to the sign of the value), and errno is set to ERANGE. When the −Xc or −Xa compilation options are used, HUGE_VAL is returned instead of HUGE.

If the correct value would cause an underflow, 0 is returned and errno is set to ERANGE.

If str is NaN, then atof() returns NaN.

ERRORS

The strtod() function will fail if:

ERANGE   The value to be returned would cause overflow or underflow.

The strtod() function may fail if:

EINVAL   No conversion could be performed.

USAGE

Because 0 is returned on error and is also a valid return on success, an application wishing to check for error situations should set errno to 0, then call strtod(), then check errno and if it is non-zero, assume an error has occurred.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe with exceptions</td>
</tr>
<tr>
<td>CSI</td>
<td>Enabled</td>
</tr>
</tbody>
</table>

SEE ALSO

isspace(3C), localeconv(3C), scanf(3S), setlocale(3C), strtol(3C), attributes(5)

NOTES

These functions can be used safely in multi-thread applications, as long as setlocale(3C) is not called to change the locale.
NAME
strtol, strtoll, atol, atoll, atoi, litostr, ulitostr – string conversion routines

SYNOPSIS
#include <stdlib.h>
long strtol(const char *str, char **endptr, int base);
long long strtoll(const char *str, char **endptr, int base);
long atol(const char *str);
long long atoll(const char *str);
int atoi(const char *str);
char *lltostr(long long value, char *endptr);
char *ulitostr(unsigned long long value, char *endptr);

DESCRIPTION
The strtol() function converts the initial portion of the string pointed to by str to a type long int representation. First it decomposes the input string into three parts: an initial, possibly empty, sequence of white-space characters (as specified by isspace(3C)); a subject sequence interpreted as an integer represented in some radix determined by the value of base; and a final string of one or more unrecognized characters, including the terminating null byte of the input string. Then it attempts to convert the subject sequence to an integer, and returns the result.

If the value of base 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.

If the value of base 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 base, optionally preceded by a + or – sign. The letters from a (or A) to z (or Z) inclusive are ascribed the values 10 to 35; only letters whose ascribed values are less than that of base are permitted. If the value of base is 16, the characters 0x or 0X may optionally precede the sequence of letters and digits, following the sign if present.

The subject sequence is defined as the longest initial subsequence of the input string, starting with the first non-white-space character, that is of the expected form. The subject sequence contains no characters if the input string is empty or consists entirely of white-space characters, or if the first non-white-space character is other than a sign or a permissible letter or digit.

If the subject sequence has the expected form and the value of base is 0, the sequence of characters starting with the first digit is interpreted as an integer constant. If the subject sequence has the expected form and the value of base is between 2 and 36, it is used as the base for conversion, ascribing to each letter its value as given above. If the subject sequence begins with a minus sign, the value resulting from the conversion is negated. A
pointer to the final string is stored in the object pointed to by endptr, provided that endptr is not a null pointer.

In other than the POSIX locale, additional implementation-dependent subject sequence forms may be accepted.

If the subject sequence is empty or does not have the expected form, no conversion is performed; the value of str is stored in the object pointed to by endptr, provided that endptr is not a null pointer.

Except for behavior on error, atol() is equivalent to: strtol(str, (char **)NULL, 10).

Except for behavior on error, atoll() is equivalent to: strtoll(str, (char **)NULL, 10).

Except for behavior on error, atoi() is equivalent to: (int) strtol(str, (char **)NULL, 10).

lltostr() returns a pointer to the string represented by the long long value. endptr is assumed to point to the byte following a storage area into which the decimal representation of value is to be placed as a string. lltostr() converts value to decimal and produces the string, and returns a pointer to the beginning of the string. No leading zeros are produced, and no terminating null is produced. The low-order digit of the result always occupies memory position endptr-1. lltostr()'s behavior is undefined if value is negative. A single zero digit is produced if value is 0.

ulltostr() is similar to lltostr() except that value is an unsigned long long.

RETURN VALUES

Upon successful completion strtol() returns the converted value, if any. If no conversion could be performed, 0 is returned and errno may be set to ERANGE.

If the correct value is outside the range of representable values, LONG_MAX or LONG_MIN is returned (according to the sign of the value), and errno is set to ERANGE.

ERRORS

The strtol() function will fail if:

ERANGE The value to be returned is not representable.

The strtol() function may fail if:

EINVAL The value of base is not supported.

USAGE

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, then call strtol(), then check errno and if it is non-zero, assume an error has occurred.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO isalpha(3C), isspace(3C), scanf(3S), strtod(3C), attributes(5)
| NOTES | \texttt{strtol()} no longer accepts values greater than \texttt{LONG\_MAX} as valid input. Use \texttt{strtoul()} instead. |

3C-1510 SunOS 5.6 modified 29 Dec 1996
NAME   strtoul, strtoull – convert string to unsigned long

SYNOPSIS  
```
#include <stdlib.h>

unsigned long strtoul(const char *str, char **endptr, int base);
unsigned long long strtoull(const char *str, char **endptr, int base);
```

DESCRIPTION The **strtoul()** function converts the initial portion of the string pointed to by *str* to a type **unsigned long int** representation. First it decomposes the input string into three parts: an initial, possibly empty, sequence of white-space characters (as specified by **isspace(3C)**); a subject sequence interpreted as an integer represented in some radix determined by the value of *base*; and a final string of one or more unrecognised characters, including the terminating null byte of the input string. Then it attempts to convert the subject sequence to an unsigned integer, and returns the result.

If the value of *base* 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.

If the value of *base* 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 *base*, optionally preceded by a + or − sign. The letters from a (or A) to z (or Z) inclusive are ascribed the values 10 to 35; only letters whose ascribed values are less than that of *base* are permitted. If the value of *base* is 16, the characters 0x or 0X may optionally precede the sequence of letters and digits, following the sign if present.

The subject sequence is defined as the longest initial subsequence of the input string, starting with the first non-white-space character, that is of the expected form. The subject sequence contains no characters if the input string is empty or consists entirely of white-space characters, or if the first non-white-space character is other than a sign or a permissible letter or digit.

If the subject sequence has the expected form and the value of *base* is 0, the sequence of characters starting with the first digit is interpreted as an integer constant. If the subject sequence has the expected form and the value of *base* is between 2 and 36, it is used as the base for conversion, ascribing to each letter its value as given above. If the subject sequence begins with a minus sign, the value resulting from the conversion is negated. A pointer to the final string is stored in the object pointed to by *endptr*, provided that *endptr* is not a null pointer.

In other than the POSIX locale, additional implementation-dependent subject sequence forms may be accepted.
If the subject sequence is empty or does not have the expected form, no conversion is performed; the value of \textit{str} is stored in the object pointed to by \textit{endptr}, provided that \textit{endptr} is not a null pointer.

The \texttt{strtoull()} function is identical to \texttt{strtoul()} except that it returns the value represented by \textit{str} as an \texttt{unsigned long long}.

**RETURN VALUES**

Upon successful completion \texttt{strtoul()} returns the converted value, if any. If no conversion could be performed, 0 is returned and \texttt{errno} may be set to \texttt{EINVAL}. If the correct value is outside the range of representable values, \texttt{ULONG_MAX} is returned and \texttt{errno} is set to \texttt{ERANGE}.

**ERRORS**

The \texttt{strtoul()} function will fail if:

- \texttt{EINVAL} The value of \textit{base} is not supported.
- \texttt{ERANGE} The value to be returned is not representable.

The \texttt{strtoul()} function may fail if:

- \texttt{EINVAL} No conversion could be performed.

**USAGE**

Because 0 and \texttt{ULONG_MAX} are returned on error and are also valid returns on success, an application wishing to check for error situations should set \texttt{errno} to 0, then call \texttt{strtoul()}, then check \texttt{errno} and if it is non-zero, assume an error has occurred.

Unlike \texttt{strtol(3C)} and \texttt{strtol(3C)}, \texttt{strtoul()} must always return a non-negative number; so, using the return value of \texttt{strtoul()} for out-of-range numbers with \texttt{strtoul()} could cause more severe problems than just loss of precision if those numbers can ever be negative.

**ATTRIBUTES**

See \texttt{attributes(5)} for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**SEE ALSO** \texttt{isalpha(3C)}, \texttt{isspace(3C)}, \texttt{scanf(3S)}, \texttt{strtol(3C)}, \texttt{strtol(3C)}, \texttt{attributes(5)}
NAME
strtows, wstostr – code conversion for Process Code and File Code

SYNOPSIS
#include <widec.h>

wchar_t *strtows(wchart_t *dst, const char *src);
char *wstostr(char *dst, const wchart_t *src);

DESCRIPTION

strtows() takes a character string src, converts it to a Process Code string, terminated by a Process Code NULL, and places the result into dst.

wstostr() takes the Process Code string pointed to by src, converts it to a character string, and places the result into dst.

RETURN VALUES
strtows() returns the Process Code string if it completes successfully. Otherwise, a NULL pointer will be returned and errno will be set for the error EILSEQ.

wstostr() returns the File Code string if it completes successfully. Otherwise, a NULL pointer will be returned and errno will be set for the error EILSEQ.

SEE ALSO
wstring(3C)
NAME  strxfrm – string transformation

SYNOPSIS  

#include <string.h>

size_t strxfrm(char *s1, const char *s2, size_t n);

DESCRIPTION  
The strxfrm() function transforms the string pointed to by s2 and places the resulting string into the array pointed to by s1. The transformation is such that if strcmp(3C) is applied to two transformed strings, it returns a value greater than, equal to or less than 0, corresponding to the result of strcoll(3C) applied to the same two original strings. No more than n bytes are placed into the resulting array pointed to by s1, including the terminating null byte. If n is 0, s1 is permitted to be a null pointer. If copying takes place between objects that overlap, the behavior is undefined.

RETURN VALUES  
Upon successful completion, strxfrm() returns the length of the transformed string (not including the terminating null byte). If the value returned is n or more, the contents of the array pointed to by s1 are indeterminate. On failure, strxfrm() returns (size_t)−1.

USAGE  
The transformation function is such that two transformed strings can be ordered by strcmp(3C) as appropriate to collating sequence information in the program’s locale (category LC_COLLATE). The fact that when n is 0, s1 is permitted to be a null pointer, is useful to determine the size of the s1 array prior to making the transformation. Because no return value is reserved to indicate an error, an application wishing to check for error situations should set errno to 0, then call strcoll(3C), then check errno and if it is non-zero, assume an error has occurred. This issue is aligned with the ANSI C standard; this does not affect compatibility with XPG3 applications. Reliable error detection by this function was never guaranteed.

EXAMPLES  
The value of the following expression is the size of the array needed to hold the transformation of the string pointed to by s.

1 + strxfrm(NULL, s, 0);

FILES  
/usr/lib/locale/locale/locale.so.*

LC_COLLATE database for locale

ATTRIBUTES  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe with exceptions</td>
</tr>
<tr>
<td>CSI</td>
<td>Enabled</td>
</tr>
</tbody>
</table>

SEE ALSO  
localedef(1), setlocale(3C), strcmp(3C), strcoll(3C), wscoll(3C), attributes(5), 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 <unistd.h>

void swab(const void *src, void *dest, ssize_t nbytes);

DESCRIPTION  The swab() function copies nbytes bytes, which are pointed to by src, to the object pointed to by dest, exchanging adjacent bytes. The nbytes argument should be even. If nbytes is odd swab() copies and exchanges nbytes−1 bytes and the disposition of the last byte is unspecified. If copying takes place between objects that overlap, the behaviour is undefined. If nbytes is negative, swab() does nothing.

ERRORS  No errors are defined.

ATTRIBUTES  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  attributes(5)
NAME     sync_instruction_memory – make modified instructions executable

SYNOPSIS  void sync_instruction_memory(caddr_t addr, int len);

DESCRIPTION sync_instruction_memory() performs whatever steps are required to make instructions modified by a program executable.

Some processor architectures, including some SPARC processors, have separate and independent instruction and data caches which are not kept consistent by hardware. For example, if the instruction cache contains an instruction from some address and the program then stores a new instruction at that address, the new instruction may not be immediately visible to the instruction fetch mechanism. Software must explicitly invalidate the instruction cache entries for new or changed mappings of pages that might contain executable instructions. sync_instruction_memory() performs this function, and/or any other functions needed to make modified instructions between addr and addr+len visible. A program should call sync_instruction_memory() after modifying instructions and before executing them.

On processors with unified caches (one cache for both instructions and data) and pipelines which are flushed by a branch instruction, such as the Intel x86 architecture, the function may do nothing and just return.

The changes are immediately visible to the thread calling sync_instruction_memory() when the call returns, even if the thread should migrate to another processor during or after the call. The changes become visible to other threads in the same manner that stores do; that is, they eventually become visible, but the latency is implementation-dependent.

RETURN VALUES     None

ERRORS     The result of executing sync_instruction_memory() are unpredictable if addr through addr+len-1 are not valid for the address space of the program making the call.

ATTRIBUTES     See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO     attributes(5)
NAME  
syncok, wcursyncup, wsyncdown, wsyncup – synchronize window with its parents or children

SYNOPSIS  
#include <curses.h>
int syncok(WINDOW *win, bool bf);
void wcursyncup(WINDOW *win);
void wsyncdown(WINDOW *win);
void wsyncup(WINDOW *win);

ARGUMENTS  
win Is a pointer to a window.
bf Is a Boolean expression.

DESCRIPTION  
The syncok() function uses the value of bf to determine whether or not the window win's ancestors are implicitly touched whenever there is a change to win. If bf is TRUE, this touching occurs. If bf is FALSE, it does not occur. The initial value for bf is FALSE.

The wcursyncup() function moves the cursor in win's ancestors to match its position in win.

The wsyncdown() function touches win if any of its ancestors have been touched.

The wsyncup() function touches all ancestors of win.

RETURN VALUES  
On success, the syncok() function returns OK. Otherwise, it returns ERR.
The other functions do not return a value.

ERRORS  
None.

SEE ALSO  
derwin(3XC), doupdate(3XC), is_linetouched(3XC)
NAME      syscall – indirect system call

SYNOPSIS  /usr/ucb/cc [flag ...] file ...
            #include <sys/syscall.h>
            int syscall(number, arg, ...)

DESCRIPTION syscall() performs the function whose assembly language interface has the specified number, and arguments arg ... . 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.

modified 22 Jan 1993

SunOS 5.6

3B-1519
NAME
sysconf – get configurable system variables

SYNOPSIS
#include <unistd.h>

long sysconf(int name);

DESCRIPTION
The sysconf() function provides a method for an application to determine the current value of a configurable system limit or option (variable).

The name argument represents the system variable to be queried. The following table lists the minimal set of system variables from <limits.h> and <unistd.h> that can be returned by sysconf() and the symbolic constants defined in <unistd.h> that are the corresponding values used for name on the SPARC and x86 platforms.

<table>
<thead>
<tr>
<th>Name</th>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>_SC_ARG_MAX</td>
<td>ARG_MAX</td>
<td>Max size of argv[ ] plus envp[ ]</td>
</tr>
<tr>
<td>_SC_BC_BASE_MAX</td>
<td>BC_BASE_MAX</td>
<td>Maximum obase values allowed by bc</td>
</tr>
<tr>
<td>_SC_BC_DIM_MAX</td>
<td>BC_DIM_MAX</td>
<td>Max number of elements permitted in array by bc</td>
</tr>
<tr>
<td>_SC_BC_SCALE_MAX</td>
<td>BC_SCALE_MAX</td>
<td>Max scale value allowed by bc</td>
</tr>
<tr>
<td>_SC_BC_STRING_MAX</td>
<td>BC_STRING_MAX</td>
<td>Max length of string constant allowed by bc</td>
</tr>
<tr>
<td>_SC_CHILD_MAX</td>
<td>CHILD_MAX</td>
<td>Max processes allowed to a UID</td>
</tr>
<tr>
<td>_SC_CLK_TCK</td>
<td>CLK_TCK</td>
<td>Ticks per second (clock_t)</td>
</tr>
<tr>
<td>_SC_COLL_WEIGHTS_MAX</td>
<td>COLL_WEIGHTS_MAX</td>
<td>Max number of weights that can be assigned to entry of the LC_COLLATE order keyword in locale definition file</td>
</tr>
<tr>
<td>_SC_EXPR_NEST_MAX</td>
<td>EXPR_NEST_MAX</td>
<td>Max number of expressions that can be listed within parentheses by expr</td>
</tr>
<tr>
<td>_SC_LINE_MAX</td>
<td>LINE_MAX</td>
<td>Max length of input line</td>
</tr>
<tr>
<td>_SC_NGROUPS_MAX</td>
<td>NGROUPS_MAX</td>
<td>Max simultaneous groups to which one may belong</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>Max number of significant bytes in a password</td>
</tr>
<tr>
<td>_SC_2_C_BIND</td>
<td>_POSIX2_C_BIND</td>
<td>Supports the C language binding option</td>
</tr>
<tr>
<td>_SC_2_C_DEV</td>
<td>_POSIX2_C_DEV</td>
<td>Supports the C language development utilities option</td>
</tr>
<tr>
<td>_SC_2_C_VERSION</td>
<td>_POSIX2_C_VERSION</td>
<td>Integer value indicating version of ISO POSIX-2 standard (Commands)</td>
</tr>
<tr>
<td>_SC_2_CHAR_TERM</td>
<td>_POSIX2_CHAR_TERM</td>
<td>Supports at least one terminal</td>
</tr>
<tr>
<td>_SC_2_FORT_DEV</td>
<td>_POSIX2_FORT_DEV</td>
<td>Supports FORTRAN Development</td>
</tr>
<tr>
<td>Definition</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td><code>_SC_2_FORT_RUN</code></td>
<td>Supports FORTRAN Run-time Utilities Option</td>
<td></td>
</tr>
<tr>
<td><code>_SC_2_LOCALEDEF</code></td>
<td>Supports the creation of locales by the <code>localedef</code> utility</td>
<td></td>
</tr>
<tr>
<td><code>_SC_2_SW_DEV</code></td>
<td>Supports the Software Development Utility Option</td>
<td></td>
</tr>
<tr>
<td><code>_SC_2_UPE</code></td>
<td>Supports the User Portability Utilities Option</td>
<td></td>
</tr>
<tr>
<td><code>_SC_2_VERSION</code></td>
<td>Integer value indicating version of ISO POSIX-2 standard (C language binding)</td>
<td></td>
</tr>
<tr>
<td><code>_SC_JOB_CONTROL</code></td>
<td>Job control supported?</td>
<td></td>
</tr>
<tr>
<td><code>_SC_SAVED_IDS</code></td>
<td>Saved IDs (<code>seteuid()</code> supported)?</td>
<td></td>
</tr>
<tr>
<td><code>_SC_VERSION</code></td>
<td>POSIX.1 version supported</td>
<td></td>
</tr>
<tr>
<td><code>_SC_RE_DUP_MAX</code></td>
<td>Max number of repeated occurrences of a regular expression permitted when using the interval notation ([m,n]n)</td>
<td></td>
</tr>
<tr>
<td><code>_SC_STREAM_MAX</code></td>
<td>Number of streams one processed can have open at a time</td>
<td></td>
</tr>
<tr>
<td><code>_SC_TZNAME_MAX</code></td>
<td>Max number of bytes supported for name of a time zone</td>
<td></td>
</tr>
<tr>
<td><code>_SC_XOPEN_CRYPT</code></td>
<td>Supports X/Open Encryption Feature Group</td>
<td></td>
</tr>
<tr>
<td><code>_SC_XOPEN_ENH_I18N</code></td>
<td>Supports X/Open Enhance Internationalization Feature Group</td>
<td></td>
</tr>
<tr>
<td><code>_SC_XOPEN_SHM</code></td>
<td>Supports X/Open Shared Memory Feature Group</td>
<td></td>
</tr>
<tr>
<td><code>_SC_XOPEN_VERSION</code></td>
<td>Integer value indicating version of X/Open Portability Guide to which implementation conforms</td>
<td></td>
</tr>
<tr>
<td><code>_SC_XOPEN_XCU_VERSION</code></td>
<td>Integer value indicating version of XCU specification to which implementation conforms</td>
<td></td>
</tr>
<tr>
<td><code>_SC_ATEXIT_MAX</code></td>
<td>Max number of functions that may be registered with <code>atexit()</code></td>
<td></td>
</tr>
<tr>
<td><code>_SC_IOV_MAX</code></td>
<td>Max number of <code>iovec</code> structures that one process has available for use with <code>readv()</code> and <code>writev()</code></td>
<td></td>
</tr>
<tr>
<td><code>_SC_PAGESIZE</code></td>
<td>System memory page size</td>
<td></td>
</tr>
<tr>
<td><code>_SC_PAGE_SIZE</code></td>
<td>Same as <code>_SC_PAGESIZE</code></td>
<td></td>
</tr>
<tr>
<td><code>_SC_XOPEN_UNIX</code></td>
<td>Supports X/Open CAE Specification, August 1994, System Interfaces and Headers,</td>
<td></td>
</tr>
<tr>
<td><code>_POSIX2_FORT_RUN</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>_POSIX2_LOCALEDEF</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>_POSIX2_SW_DEV</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>_POSIX2_UPE</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>_POSIX2_VERSION</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>_POSIX_JOB_CONTROL</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>_POSIX_SAVED_IDS</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>_POSIX_VERSION</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>_XOPEN_CRYPT</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>_XOPEN_ENH_I18N</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>_XOPEN_SHM</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>_XOPEN_VERSION</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>_XOPEN_XCU_VERSION</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ATEXIT_MAX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IOV_MAX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PAGESIZE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SunOS 5.6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3C-1521</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

modified 20 Mar 1997
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_SC_LOGNAME_MAX</td>
<td>LOGNAME_MAX</td>
</tr>
<tr>
<td>_SC_NPROCESSORS_CONF</td>
<td>Number of processors configured</td>
</tr>
<tr>
<td>_SC_NPROCESSORS_ONLN</td>
<td>Number of processors online</td>
</tr>
<tr>
<td>_SC_PHYS_PAGES</td>
<td>Total number of pages of physical memory in system</td>
</tr>
<tr>
<td>_SC_AVPHYS_PAGES</td>
<td>Number physical memory pages not currently in use by system</td>
</tr>
<tr>
<td>_SC_AIO_LISTIO_MAX</td>
<td>AIO_LISTIO_MAX</td>
</tr>
<tr>
<td>_SC_AIO_MAX</td>
<td>AIO_MAX</td>
</tr>
<tr>
<td>_SC_AIO_PRIO_DELTA_MAX</td>
<td>AIO_PRIO_DELTA_MAX</td>
</tr>
<tr>
<td>_SC_DELAYTIMER_MAX</td>
<td>DELAYTIMER_MAX</td>
</tr>
<tr>
<td>_SC_GETGR_R_SIZE_MAX</td>
<td>NSS_BUFLEN_GROUP</td>
</tr>
<tr>
<td>_SC_GETPW_R_SIZE_MAX</td>
<td>NSS_BUFLEN_PASSWD</td>
</tr>
<tr>
<td>_SC_LOGIN_NAME_MAX</td>
<td>LOGNAME_MAX + 1</td>
</tr>
<tr>
<td>_SC_MQ_OPEN_MAX</td>
<td>MQ_OPEN_MAX</td>
</tr>
<tr>
<td>_SC_MQ_PRIO_MAX</td>
<td>MQ_PRIO_MAX</td>
</tr>
<tr>
<td>_SC_RTSIG_MAX</td>
<td>RTSIG_MAX</td>
</tr>
<tr>
<td>_SC_SEM_NSEMS_MAX</td>
<td>SEM_NSEMS_MAX</td>
</tr>
<tr>
<td>_SC_SEM_VALUE_MAX</td>
<td>SEM_VALUE_MAX</td>
</tr>
<tr>
<td>_SC_SIGQUEUE_MAX</td>
<td>SIGQUEUE_MAX</td>
</tr>
<tr>
<td>_SC_TIMER_MAX</td>
<td>TIMER_MAX</td>
</tr>
<tr>
<td>_SC.getAsynchronous_IO</td>
<td>_POSIXgetAsynchronous_IO</td>
</tr>
<tr>
<td>_SC_FSYNC</td>
<td>_POSIX_FSYNC</td>
</tr>
<tr>
<td>_SC_MAPPED_FILES</td>
<td>_POSIX_MAPPED_FILES</td>
</tr>
<tr>
<td>_SC_MEMLOCK</td>
<td>_POSIX_MEMLOCK</td>
</tr>
<tr>
<td>_SC_MEMLOCK_RANGE</td>
<td>_POSIX_MEMLOCK_RANGE</td>
</tr>
<tr>
<td>_SC_MEMORY_PROTECTION</td>
<td>_POSIX_MEMORY_PROTECTION</td>
</tr>
<tr>
<td>_SC_MESSAGE_PASSING</td>
<td>_POSIX_MESSAGE_PASSING</td>
</tr>
</tbody>
</table>

Issue 4, Version 2

Number of processors configured
Number of processors online
Total number of pages of physical memory in system
Number physical memory pages not currently in use by system
Max number of I/O operations in a single list I/O call supported by implementation
Max number of outstanding asynchronous I/O operations supported by implementation
Max amount by which a process can decrease its asynchronous I/O priority level from its own scheduling priority
Max number of timer expiration overruns
Max size of group entry buffer.
Max size of password entry buffer.
Max length of login name.
Max number of open message queues a process may hold.
Max number of message priorities supported by implementation.
Max number of realtime signals reserved for application use in this implementation.
Max number of semaphores that a process may have.
Max value a semaphore may have.
Max number of queued signals that a process may send and have pending at receiver(s) at a time.
Max number of timers per process supported by implementation.
Supports Asynchronous I/O.
Supports File Synchronization.
Supports Memory Mapped Files.
Supports Process Memory Locking.
Supports Range Memory Locking.
Supports Memory Protection.
Supports Message Passing.

3C-1522 SunOS 5.6 modified 20 Mar 1997
The following table lists the names and return values for SPARC and x86 platform variables.

<table>
<thead>
<tr>
<th>Name</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>_SC_COHER_BLKSZ</td>
<td>EINVAL</td>
</tr>
<tr>
<td>_SC_SPLIT_CACHE</td>
<td>EINVAL</td>
</tr>
<tr>
<td>_SC_ICACHE_SZ</td>
<td>EINVAL</td>
</tr>
<tr>
<td>_SC_DCACHE_SZ</td>
<td>EINVAL</td>
</tr>
<tr>
<td>_SC_ICACHE_LINESZ</td>
<td>EINVAL</td>
</tr>
<tr>
<td>_SC_DCACHE_LINESZ</td>
<td>EINVAL</td>
</tr>
<tr>
<td>_SC_ICACHE_BLKSZ</td>
<td>EINVAL</td>
</tr>
<tr>
<td>_SC_DCACHE_BLKSZ</td>
<td>EINVAL</td>
</tr>
<tr>
<td>_SC_DCACHE_TBLKSZ</td>
<td>EINVAL</td>
</tr>
<tr>
<td>_SC_PRIORITY_SCHEDULING</td>
<td>_POSIX_PRIORITY_SCHEDULING</td>
</tr>
<tr>
<td>_SC_REALTIME_SIGNALS</td>
<td>_POSIX_REALTIME_SIGNALS</td>
</tr>
<tr>
<td>_SC_SEMAPHORES</td>
<td>_POSIX_SEMAPHORES</td>
</tr>
<tr>
<td>_SC_SCALED_MEMORY_OBJECTS</td>
<td>_POSIX_SCALED_MEMORY_OBJECTS</td>
</tr>
<tr>
<td>_SC_SYNCHRONIZED_IO</td>
<td>_POSIX_SYNCHRONIZED_IO</td>
</tr>
<tr>
<td>_SC_TIMERS</td>
<td>_POSIX_TIMERS</td>
</tr>
<tr>
<td>_SC_THREAD_DESTRUCTOR_ITERATIONS</td>
<td>_PTHREAD_DESTRUCTOR_ITERATIONS</td>
</tr>
<tr>
<td>_SC_THREAD_KEYS_MAX</td>
<td>_PTHREAD_KEYS_MAX</td>
</tr>
<tr>
<td>_SC_THREAD_STACK_MIN</td>
<td>_PTHREAD_STACK_MIN</td>
</tr>
<tr>
<td>_SC_THREAD_THREADS_MAX</td>
<td>_PTHREAD_THREADS_MAX</td>
</tr>
<tr>
<td>_SC_TTY_NAME_MAX</td>
<td>TTYNAME_MAX</td>
</tr>
<tr>
<td>_SC_THREADS</td>
<td>_POSIX_THREADS</td>
</tr>
<tr>
<td>_SC_THREAD_ATTR_STACKADDR</td>
<td>_POSIX_THREAD_ATTR_STACKADDR</td>
</tr>
<tr>
<td>_SC_THREAD_ATTR_STACKSIZE</td>
<td>_POSIX_THREAD_ATTR_STACKSIZE</td>
</tr>
<tr>
<td>_SC_THREAD_PRIORITY_SCHEDULING</td>
<td>_POSIX_THREAD_PRIORITY_SCHEDULING</td>
</tr>
<tr>
<td>_SC_THREAD_PRIO_INHERIT</td>
<td>_POSIX_THREAD_PRIO_INHERIT</td>
</tr>
<tr>
<td>_SC_THREAD_PRIO_PROTECT</td>
<td>_POSIX_THREAD_PRIO_PROTECT</td>
</tr>
<tr>
<td>_SC_THREAD_PROCESS_SHARED</td>
<td>_POSIX_THREAD_PROCESS_SHARED</td>
</tr>
<tr>
<td>_SC_THREAD_SAFE_FUNCTIONS</td>
<td>_POSIX_THREAD_SAFE_FUNCTIONS</td>
</tr>
</tbody>
</table>

modified 20 Mar 1997 SunOS 5.6 3C-1523
RETURN VALUES

If name is an invalid value, sysconf() returns –1 and sets errno to indicate the error. If the variable corresponding to name is associated with functionality that is not supported by the system, sysconf() returns –1 without changing the value of errno.

Otherwise, sysconf() returns the current variable value on the system. The value returned will not be more restrictive than the corresponding value described to the application when it was compiled with the implementation’s <limits.h>, <unistd.h> or <time.h>. The value will not change during the lifetime of the calling process.

ERRORS

The sysconf() function will fail if:

EINVAL   The value of the name argument is invalid.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>SPARC and x86</td>
</tr>
<tr>
<td>MT- Level</td>
<td>MT-Safe, Async-Signal-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO

fpathconf(2), seteuid(2), setrlimit(2), attributes(5), standards(5)

NOTES

A call to setrlimit() may cause the value of OPEN_MAX to change.

Multiplying sysconf(_SC_PHYS_PAGES) or sysconf(_SC_AVPHYS_PAGES) by
sysconf(_SC_PAGESIZE) to determine memory amount in bytes can exceed the maximum
values representable in a long or unsigned long.

_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.

Calling sysconf() with _SC_THREAD_KEYS_MAX or _SC_THREAD_THREADS_MAX
returns –1, because no maximum limit can be determined. The system supports at least
the minimum values defined by _POSIX_THREAD_KEYS_MAX and
_POSIX_THREAD_THREADS_MAX and can support higher values depending upon sys-

tem resources.

The _SC_THREAD_PRIO_INHERIT and _SC_THREAD_PRIO_PROTECT variables are
currently not supported. A call to sysconf() with these variables as arguments returns
–1.
NAME
syslog, openlog, closelog, setlogmask – control system log

SYNOPSIS
#include <syslog.h>
void openlog(const char *ident, int logopt, int facility);
void syslog(int priority, const char *message, .../* arguments */);
void closelog(void);
int setlogmask(int maskpri);

DESCRIPTION
The syslog() function sends a message to syslogd(1M), which, depending on the
configuration of /etc/syslog.conf, logs it in an appropriate system log, writes it to the sys-
tem console, forwards it to a list of users, or forwards it to syslogd on another host over
the network. The logged message includes a message header and a message body. The
message header consists of a facility indicator, a severity level indicator, a timestamp, a
tag string, and optionally the process ID.
The message body is generated from the message and following arguments in the same
manner as if these were arguments to printf(3B), except that occurrences of %m in the
format string pointed to by the message argument are replaced by the error message string
associated with the current value of errno. A trailing NEWLINE character is added if
needed.
Values of the priority argument are formed by ORing together a severity level value and an
optional facility value. If no facility value is specified, the current default facility value is
used.
Possible values of severity level include:
LOG_EMERG A panic condition. This is normally broadcast to all users.
LOG_ALERT A condition that should be corrected immediately, such as a cor-
rupted 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 spe-
cial handling.
LOG_INFO Informational messages.
LOG_DEBUG Messages that contain information normally of use only when
debugging a program.
The facility indicates the application or system component generating the message. Possi-
ble facility values include:
LOG_KERN Messages generated by the kernel. These cannot be generated by
any user processes.
LOG_USER Messages generated by random user processes. This is the default
modiﬁed 29 Dec 1996
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).
LOG_LPR The line printer spooling system: lpr(1B), lpc(1B).
LOG_NEWS Reserved for the USENET network news system.
LOG_UUCP Reserved for the UUCP network news system; it does not currently use syslog.
LOG_CRON The cron/at facility; crontab(1), at(1), cron(1M).
LOG_LOCAL0 Reserved for local use.
LOG_LOCAL1 Reserved for local use.
LOG_LOCAL2 Reserved for local use.
LOG_LOCAL3 Reserved for local use.
LOG_LOCAL4 Reserved for local use.
LOG_LOCAL5 Reserved for local use.
LOG_LOCAL6 Reserved for local use.
LOG_LOCAL7 Reserved for local use.

The openlog() function sets process attributes that affect subsequent calls to syslog(). The ident argument is a string that is prepended to every message. The logopt argument indicates logging options. Values for logopt are constructed by a bitwise-inclusive OR of zero or more of the following:

LOG_PID Log the process ID with each message. This is useful for identifying specific daemon processes (for daemons that fork).
LOG_CONS Write messages to the system console if they cannot be sent to 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_ODELAY Delay open until syslog() is called.
LOG_NOWAIT Do not wait for child processes that have been forked to log messages onto the console. This option should be used by processes that enable notification of child termination using SIGCHLD, since syslog() may otherwise block waiting for a child whose exit status has already been collected.

The facility argument encodes a default facility to be assigned to all messages that do not have an explicit facility already encoded. The initial default facility is LOG_USER.
The `openlog()` function may allocate a file descriptor. It is not necessary to call `openlog()` prior to calling `syslog()`.

The `closelog()` function closes any open file descriptors allocated by previous calls to `openlog()` or `syslog()`.

The `setlogmask()` function sets the log priority mask for the current process to `maskpri` and returns the previous mask. If the `maskpri` argument is 0, the current log mask is not modified. Calls by the current process to `syslog()` with a priority not set in `maskpri` are rejected. The mask for an individual priority `pri` is calculated by the macro `LOG_MASK(pri)`; the mask for all priorities up to and including `toppri` is given by the macro `LOG_UPTOP(toppri)`. The default log mask allows all priorities to be logged.

Symbolic constants for use as values of the `logopt`, `facility`, `priority`, and `maskpri` arguments are defined in the `<syslog.h>` header.

### RETURN VALUES
The `setlogmask()` function returns the previous log priority mask. The `closelog()`, `openlog()`, and `syslog()` functions return no value.

### ERRORS
No errors are defined.

### EXAMPLES
This call logs a message at priority `LOG_ALERT`:

```c
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:

```c
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:

```c
setlogmask(LOG_UPTO(LOG_ERR));
```

Then, to log a message at priority `LOG_INFO`, it would make the following call to `syslog`:

```c
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:

```c
syslog(LOG_INFO|LOG_LOCAL2, "error: %m");
```

### ATTRIBUTES
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>
SEE ALSO  at(1), crontab(1), logger(1), login(1), lpc(1B), lpr(1B), cron(1M), getty(1M), in.ftpd(1M), su(1M), syslogd(1M), printf(3B), syslog.conf(4), attributes(5)
NAME  sysmem, asysmem – return physical memory information

SYNOPSIS  long sysmem(void);
        long asysmem(void);

DESCRIPTION  These routines are obsolete and have been replaced by arguments to sysconf(3C). They were mistakenly published in the System V Interface Definition, 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: sysmem(AS_LIB), asysmem(AS_LIB), … They are therefore removed."

The routine sysmem() 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 sysmem() 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).)

The routine asysmem() 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 asysmem() returns a long integer it is limited similar to sysmem(). sysconf(_SC_AVPHYS_PAGES) should be used to avoid this limitation. (See sysconf(3C).)

RETURN VALUES  Upon successful completion, these routines return the amount of memory in bytes; otherwise, they return -1.

SEE ALSO  sysconf(3C)

NOTES  sysmen() and asysmem() are obsolete and should be replaced with sysconf(3C).
NAME system – issue a shell command

SYNOPSIS #include <stdlib.h>

int system(const char *string);

DESCRIPTION The system() function causes the string to be given to the shell 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 string is a null pointer, system() checks if the shell exists and is executable. If the shell is available, system() returns non-zero; otherwise it returns zero. If the application is standard-conforming (see standards(5)), system() uses /usr/bin/ksh (see ksh(1)); otherwise system() uses /usr/bin/sh (see sh(1)).

RETURN VALUES The system() function forks to create a child process that in turn execs the shell in order to execute string. If the fork() or exec() fails, system() returns a value of −1 and sets errno.

ERRORS The system() function 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.

ATTRIBUTES See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO ksh(1), sh(1), useradd(1M), exec(2), fork(2), setuid(2), waitpid(2), attributes(5), standards(5)

NOTES The system() function will fail to execute setuid() or setgid() if either the uid or gid of the application’s owner/group is less than 100. (see useradd(1M) and setuid(2)).
NAME
t_accept – accept a connection request

SYNOPSIS
cc [ flag ... ] file ... -lnsl [ library ... ]
#include <xti.h>
int t_accept(int fd, int resfd, struct t_call *call);

DESCRIPTION
This routine is part of the XTI interfaces which evolved from the TLI interfaces. XTI represents the future evolution of these interfaces. However, TLI interfaces are supported for compatibility. When using a TLI routine that has the same name as an XTI routine, a different header file, tiuser.h, must be used. Refer to the section, TLI COMPATIBILITY, for a description of differences between the two interfaces.

This function is issued by a transport user to accept a connection request. The parameter fd identifies the local transport endpoint where the connection indication arrived; resfd specifies the local transport endpoint where the connection is to be established, and call contains information required by the transport provider to complete the connection. call points to a t_call structure that contains the following members:

- struct netbuf addr;
- struct netbuf opt;
- struct netbuf udata;
- int sequence;

The netbuf structure is described in t_connect(3N).

In call, addr is the address of the calling transport user, opt indicates any options 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(3N) that uniquely associates the response with a previously received connection indication. The address of the caller, addr may be null (length zero). Where addr is not null then it may optionally be checked by XTI A transport user may accept a connection on either the same, or on a different local transport endpoint than the one on which the connection indication arrived. Before the connection can be accepted on the same endpoint (resfd==fd), the user must have responded to any previous connection indications received on that transport endpoint (using t_accept() or t_snddis(3N)). Otherwise, t_accept() will fail and set t_errno to TINDOUT.

If a different transport endpoint is specified (resfd!=fd), then the user may or may not choose to bind the endpoint before the t_accept() is issued. If the endpoint is not bound prior to the t_accept(), then the transport provider will automatically bind it to the same protocol address fd is bound to. If the transport user chooses to bind the endpoint it must be bound to a protocol address with a qlen of zero and must be in the T_IDLE state before the t_accept() is issued.

Responding endpoints should be supplied to t_accept() in the state T_UNBND.

The call to t_accept() will fail with t_errno set to TLOOK if there are indications (for example, a connection or disconnect) waiting to be received on the endpoint fd.
The `udata` argument enables the called transport user to send user data to the caller and the amount of user data must not exceed the limits supported by the transport provider, as returned in the `connect` field of the `info` argument of `t_open(3N)` or `t_getinfo(3N)`. If the `len` field of `udata` is zero, no data will be sent to the caller. All the `maxlen` fields are meaningless.

When the user does not indicate any option (`call->opt.len = 0`), the connection shall be accepted with the option values currently set for the responding endpoint `resfd`.

### VALID STATES

Legitimate states (see `t_getstate(3N)`) for a call to this routine are:

- **T_INCON** for `fd`
- **T_IDLE** for `resfd` when `fd != resfd`

### RETURN VALUES

`t_accept` returns:

- **0** On success.
- **−1** On failure.

On failure, `t_errno` is set to indicate the error, and possibly `errno` is set.

### ERRORS

On failure, `t_errno` is 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.
- **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 `fd` or `resfd` does not refer to a transport endpoint.
- **TBADOPT** The specified options were in an incorrect format or contained illegal information.
- **TBADSEQ** An invalid sequence number was specified.
- **TINDOUT** The function was called with `fd == resfd` but there are outstanding connection indications on the endpoint. Those other connection indications must be handled either by rejecting them using `t_snddis(3N)` or accepting them on a different endpoint using `t_accept()`.
- **TLOOK** An asynchronous event has occurred on the transport endpoint referenced by `fd` and requires immediate attention.
- **TNOTSUPPORT** This function is not supported by the underlying transport provider.
- **TOUTSTATE** The communications endpoint referenced by `fd` or by `resfd` is not in one of the states in which a call to this function is valid.
- **TPROVMISMATCH** The file descriptors `fd` and `resfd` do not refer to the same transport.
This error indicates that a communication problem has been detected between XTI and the transport provider for which there is no other suitable XTI \texttt{t_errno} value.

This transport provider requires both \texttt{fd} and \texttt{resfd} to be bound to the same address. This error results if they are not.

The endpoint referenced by \texttt{resfd} (where \texttt{resfd} \neq \texttt{fd}) was bound to a protocol address with a \texttt{qlen} that is greater than zero.

A system error has occurred during execution of this function; \texttt{errno} will be set to the specific error.

The XTI and TLI interface definitions have common names but use different header files. This, and other semantic differences between the two interfaces are described in the subsections below.

The XTI interfaces use the header file, \texttt{xti.h}. TLI interfaces should not use this header. They should use the header:

\begin{verbatim}
#include <tiuser.h>
\end{verbatim}

The \texttt{t_errno} values that can be set by the XTI interface and cannot be set by the TLI interface are:

\begin{verbatim}
TPROTO
tindout
tprovmismatch
tresaddr
tresqlen
\end{verbatim}

The format of the options in an \texttt{opt} buffer is dictated by the transport provider. Unlike the XTI interface, the TLI interface does not specify the buffer format.

For more information refer to the \textit{Transport Interfaces Programming Guide}.

See \textit{attributes(5)} for descriptions of the following attributes:

\begin{center}
\begin{tabular}{|c|c|}
\hline
\textbf{ATTRIBUTE TYPE} & \textbf{ATTRIBUTE VALUE} \\
\hline
MT-Level & MT-Safe \\
\hline
\end{tabular}
\end{center}

\texttt{t_accept(3N)}, \texttt{t_connect(3N)}, \texttt{t_getinfo(3N)}, \texttt{t_getstate(3N)}, \texttt{t_listen(3N)}, \texttt{t_open(3N)}, \texttt{t_optmgmt(3N)}, \texttt{t_snddis(3N)}, \texttt{t_rcvconnect(3N)}, \texttt{attributes(5)}

\textit{Transport Interfaces Programming Guide}

modified 10 Feb 1997

SunOS 5.6

3N-1533
NOTES

There may be transport provider-specific restrictions on address binding. Some transport providers do not differentiate between a connection indication and the connection itself. An example of such a transport provider is TCP. It may be able to establish a connection by the time `t_listen()` returns.

If the connection has already been established after a successful return of `t_listen()`, `t_accept()` will assign the existing connection to the transport endpoint specified by `resfd`. 
NAME       t_alloc – allocate a library structure

SYNOPSIS   cc [ flag ... ] file ... -lnsl [ library ... ]
#include <xti.h>

void *t_alloc(int fd, int struct_type, int fields);

DESCRIPTION This routine is part of the XTI interfaces which evolved from the TLI interfaces. XTI represents the future evolution of these interfaces. However, TLI interfaces are supported for compatibility. When using a TLI routine that has the same name as an XTI routine, a different header file, tiuser.h, must be used. Refer to the section, TLI COMPATIBILITY, for a description of differences between the two interfaces.

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 the struct_type parameter, which can be one of the following:

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

Each of the allocated structures may subsequently be used as an argument to one or more transport functions.

All of the structures (but not those generated using T_INFO) will contain at least one field of type struct netbuf. For each field of the netbuf type, the user may specify that the buffer for that field should be allocated as well.

The length of the buffer allocated will be equal to or greater than the appropriate size as returned in the info argument of t_open(3N) or t_getinfo(3N). The relevant fields of the info argument are described in the following list.

The fields argument determines which buffers to allocate. Its value can be specified using bitwise-OR operations with the following values:

T_ADDR  The addr field of t_bind, t_call, t_unitdata, or t_uderr structures.
T_OPT   The opt field of t_optmgmt, t_call, t_unitdata, or t_uderr structures.
T_UDATA The udata field of t_call, t_discon, or t_unitdata structures.
T_ALL   All relevant fields of struct_type. Fields which are not supported by the transport provider specified by fd will not be allocated.
For each relevant field specified in the parameter fields, `t_alloc()` allocates memory for the buffer associated with the field, it initializes the len field to zero, and it initializes the buf pointer and maxlen field accordingly. Irrelevant or unknown values passed in fields are ignored.

Since the length of the buffer allocated will be based on the same size information that is returned to the user on a call to `t_open()` or `t_getinfo()`, fd must refer to the transport endpoint through which the newly allocated structure will be passed. (However, when a T_INFO structure is being allocated, fd may be set to any value.) In this way the appropriate size information can be accessed. If the size value associated with any specified field is −1 or −2 (see `t_open(3N)` or `t_getinfo(3N)`), `t_alloc()` will be unable to determine the size of the buffer to allocate and will fail, setting t_errno to TSYSERR and errno to EINVAL. For any field not specified in fields, buf will be set to the null pointer and len 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 functions.

**VALID STATES**

Legitimate states (see `t_getstate(3N)`) for a call to this routine are every one except T_UNINIT.

**RETURN VALUES**

On successful completion, `t_alloc()` returns a pointer to the newly allocated structure. On failure, a null pointer 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 struct_type parameter was specified as something other than T_INFO and the specified file descriptor does not refer to a transport endpoint.
- **TSYSERR** A system error has occurred during execution of this function. Accordingly, errno will have been set to the specific error.
- **TNOSTRUCTYPE** Unsupported struct_type requested. This can include a request for a structure type which is inconsistent with the transport provider type specified: either connection-mode or connectionless-mode.
- **TPROTO** This error indicates that a communication problem has been detected between XTI and the transport provider for which there is no other suitable XTI t_errno value.

**TLI COMPATIBILITY**

The XTI and TLI interface definitions have common names but use different header files. This, and other semantic differences between the two interfaces are described in the subsections below.

The XTI interfaces use the header file, xti.h. TLI interfaces should not use this header. They should use the header:
#include <tiuser.h>

## Error Description

The `t_errno` values that can be set by the XTI interface and cannot be set by the TLI interface are:

- TPROTO
- TNOSTRUCTYPE

## Special Buffer Sizes

Assume that the value associated with any field of `struct t_info` (argument returned by `t_open()` or `t_getinfo()` that describes buffer limits is −1. Then 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, which may be handled as described in the next paragraph.

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).

Assume that the value associated with any field of `struct t_info` (argument returned by `t_open()` or `t_getinfo()` that describes nbuffer limits is −2. Then `t_alloc()` will set the buffer pointer to NULL and the buffer maximum size to 0, and then will return success (see `t_open(3N)` or `t_getinfo(3N)`).

For more information refer to the *Transport Interfaces Programming Guide*.

## ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

## SEE ALSO

`free(3C), malloc(3C), t_connect(3N), t_free(3N), t_getinfo(3N), t_getstate(3N), t_open(3N), attributes(5)`

*Transport Interfaces Programming Guide*
NAME      tan – tangent function

SYNOPSIS  cc [ flag ...] file ... -lm [ library ...]
           #include <math.h>
           double tan(double x);

DESCRIPTION The tan() function computes the tangent of its argument x, measured in radians.

RETURN VALUES Upon successful completion, tan() returns the tangent of x. If x is NaN or ±Inf, NaN is returned.

ERRORS No errors will occur.

ATTRIBUTES See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO atan(3M), isnan(3M), attributes(5)
NAME

tanh – hyperbolic tangent function

SYNOPSIS

cc [ flag ... ] file ... -lmath [ library ... ]
#include <math.h>

double tanh(double x);

DESCRIPTION

The tanh() function computes the hyperbolic tangent of x.

RETURN VALUES

Upon successful completion, tanh() returns the hyperbolic tangent of x.
If x is NaN, NaN is returned.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO

atanh(3M), isnan(3M), tan(3M), attributes(5)
NAME
  t_bind – bind an address to a transport endpoint

SYNOPSIS
  cc [ flag ...] file ... -lnsl [ library ...]
  #include <xti.h>

  int t_bind(int fd, const struct t_bind *req, struct t_bind *ret);

DESCRIPTION
  This routine is part of the XTI interfaces which evolved from the TLI interfaces. XTI
  represents the future evolution of these interfaces. However, TLI interfaces are supported
  for compatibility. When using a TLI routine that has the same name as an XTI routine, a
  different header file, tiuser.h, must be used. Refer to the section, TLI COMPATIBILITY,
  for a description of differences between the two interfaces.

  This function associates a protocol address with the transport endpoint specified by fd
  and activates that transport endpoint. In connection mode, the transport provider may
  begin enqueuing incoming connection indications, or servicing a connection request on
  the transport endpoint. In connectionless mode, the transport user may send or receive
  data units through the transport endpoint.

  The req and ret arguments point to a t_bind structure containing the following members:

  struct netbuf addr;
  unsigned qlen;

  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 outstand-
  ing connection indications.

  The parameter req is used to request that an address, represented by the netbuf structure,
  be bound to the given transport endpoint. The len field of this netbuf structure specifies
  the number of bytes in the address and the buf field of this netbuf structure points to the
  address buffer. The maxlen field has no meaning for the req argument.

  On return, ret contains an encoding for the address that the transport provider actually
  bound to the transport endpoint; if an address was specified in req, this will be an encod-
  ing of the same address. In ret, 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 equals zero, no address is returned. If maxlen is greater than zero and less than
  the length of the address, t_bind() fails.

  If the requested address is not available, t_bind() will return -1 with t_errno set as
  appropriate. If no address is specified in req (the len field of addr in req is zero or req is
  NULL), the transport provider will assign an appropriate address to be bound, and will
  return that address in the addr field of ret.

  If the transport provider could not allocate an address, t_bind() will fail with t_errno set
  to TNOADDR.
The parameter `req` may be null pointer if the user does not wish to specify an address to be bound. Here, the value of `qlen` is assumed to be zero, and the transport provider will assign an address to the transport endpoint. Similarly, `ret` may be a null pointer if the user does not care what address was bound by the provider and is not interested in the negotiated value of `qlen`. It is valid to set `req` and `ret` to the null pointer for the same call, in which case the provider chooses the address to bind to the transport endpoint and does not return that information to the user.

The `qlen` field has meaning only when initializing a connection-mode service. It specifies the number of outstanding connection indications that the transport provider should support for the given transport endpoint. An outstanding connection indication is one that has been passed to the transport user by the transport provider but which has not been accepted or rejected. A value of `qlen` greater than zero is only meaningful when issued by a passive transport user that expects other users to call it. The value of `qlen` will be negotiated by the transport provider and may be changed if the transport provider cannot support the specified number of outstanding connection indications.

However, this value of `qlen` will never be negotiated from a requested value greater than zero to zero. This is a requirement on transport providers; see NOTES.

On return, the `qlen` field in `ret` will contain the negotiated value.

If `fd` refers to a connection-mode service, this function allows more than one transport endpoint to be bound to the same protocol address (however, the transport provider must support this capability also), but it is not possible 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 connection indications associated with that protocol address. In other words, only one `t_bind()` for a given protocol address may specify a value of `qlen` greater than zero. In this way, the transport provider can identify which transport endpoint should be notified of an incoming connection indication. If a user attempts to bind a protocol address to a second transport endpoint with a value of `qlen` greater than zero, `t_bind()` will return −1 and set `t_errno` to TADDRBUSY. When a user accepts a connection on the transport endpoint that is being used as the listening endpoint, the bound protocol address will be found to be busy for the duration of that connection, until a `t_unbind(3N)` or `t_close(3N)` call has been issued.

No other transport endpoints may be bound for listening on that same protocol address while that initial listening endpoint is active (in the data transfer phase or in the T_IDLE state). This will prevent more than one transport endpoint bound to the same protocol address from accepting connection indications.

If `fd` refers to a connectionless-mode service, only one endpoint may be associated with a protocol address. If a user attempts to bind a second transport endpoint to an already bound protocol address, `t_bind()` will return −1 and set `t_errno` to TADDRBUSY.

**VALID STATES**

The only legitimate state (see `t_getstate(3N)`) for a call to this routine is T_UNBIND.

---

modified 10 Feb 1997

SunOS 5.6

3N-1541
t_bind(3N)  Network Functions

RETURN VALUES

**t_bind()** returns:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>On success.</td>
</tr>
<tr>
<td>-1</td>
<td>On failure.</td>
</tr>
</tbody>
</table>

On failure, **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:

- **TACCE** The user does not have permission to use the specified address.
- **TADDRBUSY** The requested address is in use.
- **TBADADDR** The specified protocol address was in an incorrect format or contained illegal information.
- **TBADF** The specified file descriptor does not refer to a transport endpoint.
- **TBUFOVFLW** The number of bytes allowed for an incoming argument (**maxlen**) is greater than 0 but not sufficient to store the value of that argument. The provider’s state will change to **T_IDLE** and the information to be returned in **ret** will be discarded.
- **TNOADDR** The transport provider could not allocate an address.
- **TOUTSTATE** The communications endpoint referenced by **fd** or **resfd** is not in one of the states in which a call to this function is valid.
- **TPROTO** This error indicates that a communication problem has been detected between XTI and the transport provider for which there is no other suitable XTI **t_errno** value.
- **TSYSERR** A system error has occurred during execution of this function, **errno** will be set to the specific error.

TLI COMPATIBILITY

The XTI and TLI interface definitions have common names but use different header files. This, and other semantic differences between the two interfaces are described in the subsections below.

**Interface Header**

The XTI interfaces use the header file, **xti.h**. TLI interfaces should not use this header. They should use the header:

```c
#include <tiuser.h>
```

**Address Bound**

The user can compare the addresses in **req** and **ret** to determine whether the transport provider bound the transport endpoint to a different address than that requested.

**Error Description Values**

The **t_errno** values that can be set by the XTI interface and cannot be set by the TLI interface are:

- **TPROTO**
- **TADDRBUSY**

3N-1542  SunOS 5.6 modified 10 Feb 1997
A **t_errno** value that this routine can return under different circumstances than its XTI counterpart is **TBUFOVFLW**. It can be returned even when the **maxlen** field of the corresponding buffer has been set to zero.

For more information refer to the *Transport Interfaces Programming Guide*.

**ATTRIBUTES**

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**SEE ALSO**

*t_accept*(3N), *t_alloc*(3N), *t_close*(3N), *t_connect*(3N), *t_getstate*(3N), *t_open*(3N),
*t_optmgmt*(3N), *t_unbind*(3N), attributes(5)

*Transport Interfaces Programming Guide*

**NOTES**

The requirement that the value of **qlen** never be negotiated from a requested value greater than zero to zero implies that transport providers, rather than the XTI implementation itself, accept this restriction.

An implementation need not allow an application explicitly to bind more than one communications endpoint to a single protocol address, while permitting more than one connection to be accepted to the same protocol address. That means that although an attempt to bind a communications endpoint to some address with **qlen=0** might be rejected with **TADDRBUSY**, the user may nevertheless use this (unbound) endpoint as a responding endpoint in a call to *t_accept*( ). To become independent of such implementation differences, the user should supply unbound responding endpoints to *t_accept*( ).
NAME
tcdrain – wait for transmission of output

SYNOPSIS
#include <termios.h>
int tcdrain(int fd);

DESCRIPTION
The `tcdrain()` function waits until all output written to the object referred to by `fd` is transmitted. The `fd` argument is an open file descriptor associated with a terminal. Any attempts to use `tcdrain()` from a process which is a member of a background process group on a `fd` associated with its controlling terminal, will cause the process group to be sent a `SIGTTOU` signal. If the calling process is blocking or ignoring `SIGTTOU` signals, the process is allowed to perform the operation, and no signal is sent.

RETURN VALUES
Upon successful completion, 0 is returned. Otherwise, −1 is returned and `errno` is set to indicate the error.

ERRORS
The `tcdrain()` function will fail if:

- **EBADF** The `fd` argument is not a valid file descriptor.
- **EINTR** A signal interrupted `tcdrain()`.
- **ENOTTY** The file associated with `fd` is not a terminal.

The `tcdrain()` function may fail if:

- **EIO** The process group of the writing process is orphaned, and the writing process is not ignoring or blocking `SIGTTOU`.

ATTRIBUTES
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe, and Async-Signal-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
`tcflush(3)`, `attributes(5)`, `termio(7I)`
NAME
tcflow – suspend or restart the transmission or reception of data

SYNOPSIS
#include <termios.h>
int tcflow(int fd, int action);

DESCRIPTION
The tcflow() function suspends transmission or reception of data on the object referred to
by fd, depending on the value of action. The fd argument is an open file descriptor
associated with a terminal.

- If action is TCOFF, output is suspended.
- If action is TCOON, suspended output is restarted.
- If action is TCIOFF, the system transmits a STOP character, which is intended to cause
the terminal device to stop transmitting data to the system.
- If action is TCION, the system transmits a START character, which is intended to cause
the terminal device to start transmitting data to the system.

The default on the opening of a terminal file is that neither its input nor its output are
suspended.

Attempts to use tcflow() from a process which is a member of a background process
group on a fd associated with its controlling terminal, will cause the process group to
be sent a SIGTTOU signal. If the calling process is blocking or ignoring SIGTTOU signals,
the process is allowed to perform the operation, and no signal is sent.

RETURN VALUES
Upon successful completion, 0 is returned. Otherwise, −1 is returned and errno is set to
indicate the error.

ERRORS
The tcflow() function will fail if:
EBADF  The fd argument is not a valid file descriptor.
EINVAL  The action argument is not a supported value.
ENOTTY  The file associated with fd is not a terminal.

The tcflow() function may fail if:
EIO      The process group of the writing process is orphaned, and the writing process
is not ignoring or blocking SIGTTOU.

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe, and Async-Signal-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
tcsendbreak(3), attributes(5), termio(7I)

modified 29 Dec 1996
NAME
tcflush – flush non-transmitted output data, non-read input data or both

SYNOPSIS
#include <termios.h>
int tcflush(int fildes, int queue_selector);

DESCRIPTION
Upon successful completion, tcflush() discards data written to the object referred to by
fildes (an open file descriptor associated with a terminal) 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 writ-
ten but not transmitted.

Attempts to use tcflush() from a process which is a member of a background process
group on a fildes associated with its controlling terminal, will cause the process group to
be sent a SIGTTOU signal. If the calling process is blocking or ignoring SIGTTOU signals,
the process is allowed to perform the operation, and no signal is sent.

RETURN VALUES
Upon successful completion, 0 is returned. Otherwise, −1 is returned and errno is set to
indicate the error.

ERRORS
The tcflush() function will fail if:
EBADF The fildes argument is not a valid file descriptor.
EINVAL The queue_selector argument is not a supported value.
ENOTTY The file associated with fildes is not a terminal.

The tcflush() function may fail if:
EIO The process group of the writing process is orphaned, and the writing process
is not ignoring or blocking SIGTTOU.

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe, and Async-Signal-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
tcdrain(3), attributes(5), termio(7I)
NAME
tcgetattr – get the parameters associated with the terminal

SYNOPSIS
#include <termios.h>
int tcgetattr(int fildes, struct termios *termios_p);

DESCRIPTION
The tcgetattr() function gets the parameters associated with the terminal referred to by
fildes and stores them in the termios structure (see termio(7I)) referenced by termios_p.
The fildes argument is an open file descriptor associated with a terminal.
The termios_p argument is a pointer to a termios structure.
The tcgetattr() operation is allowed from any process.
If the terminal device supports different input and output baud rates, the baud rates
stored in the termios structure returned by tcgetattr() reflect the actual baud rates, even
if they are equal. If differing baud rates are not supported, the rate returned as the output
baud rate is the actual baud rate. If the terminal device does not support split baud rates,
the input baud rate stored in the termios structure will be 0.

RETURN VALUES
Upon successful completion, 0 is returned. Otherwise, –1 is returned and errno is set to
indicate the error.

ERRORS
The tcgetattr() function will fail if:
EBADF The fildes argument is not a valid file descriptor.
ENOTTY The file associated with fildes is not a terminal.

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe, and Async-Signal-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
tcsetattr(3), attributes(5), termio(7I)
NAME
tcgetpgrp – get foreground process group ID

SYNOPSIS
#include <sys/types.h>
#include <unistd.h>

pid_t tcgetpgrp(int fd);

DESCRIPTION
The `tcgetpgrp()` function will return the value of the process group ID of the foreground
process group associated with the terminal.
If there is no foreground process group, `tcgetpgrp()` returns a value greater than 1 that
does not match the process group ID of any existing process group.
The `tcgetpgrp()` function is allowed from a process that is a member of a background
process group; however, the information may be subsequently changed by a process that
is a member of a foreground process group.

RETURN VALUES
Upon successful completion, `tcgetpgrp()` returns the value of the process group ID of the
foreground process associated with the terminal. Otherwise, −1 is returned and `errno`
is set to indicate the error.

ERRORS
The `tcgetpgrp()` function will fail if:
EBADF The `fd` argument is not a valid file descriptor.
ENOTTY The calling process does not have a controlling terminal, or the file is not the
controlling terminal.

ATTRIBUTES
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe, and Async-Signal-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
setpgid(2), setsid(2), tcsetpgrp(3C), attributes(5), termio(7I)
C Library Functions

NAME tcgetsid – get process group ID for session leader for controlling terminal

SYNOPSIS

```
#include <termios.h>

pid_t tcgetsid(int fildes);
```

DESCRIPTION

The `tcgetsid()` function obtains the process group ID of the session for which the terminal specified by `fildes` is the controlling terminal.

RETURN VALUES

Upon successful completion, `tcgetsid()` returns the process group ID associated with the terminal. Otherwise, a value of `(pid_t)-1` is returned and `errno` is set to indicate the error.

ERRORS

The `tcgetsid()` function will fail if:

- `EACCES` The `fildes` argument is not associated with a controlling terminal.
- `EBADF` The `fildes` argument is not a valid file descriptor.
- `ENOTTY` The file associated with `fildes` is not a terminal.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

```
<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
```

SEE ALSO

`attributes(5)`, `termio(7I)`

modified 29 Dec 1996

SunOS 5.6

3-1549
NAME  
t_close – close a transport endpoint

SYNOPSIS  
cc [ flag ... ] file ... -lnsl [ library ... ]
#include <xti.h>
int t_close(int fd);

DESCRIPTION  
This routine is part of the XTI interfaces which evolved from the TLI interfaces. XTI represents the future evolution of these interfaces. However, TLI interfaces are supported for compatibility. When using a TLI routine that has the same name as an XTI routine, a different header file, tiuser.h, must be used. Refer to the section, TLI COMPATIBILITY, for a description of differences between the two interfaces.

The t_close() function informs the transport provider that the user is finished with the transport endpoint specified by fd, and frees any local library resources associated with the endpoint. In addition, t_close() closes the file associated with the transport endpoint.

The t_close() function 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() will be issued for that file descriptor; if there are no other descriptors in this process or in another process which references the communications endpoint, any connection that may be associated with that endpoint will be broken when the close() function is called.

The connection may be terminated in an orderly or abortive manner depending on the service type supported by the underlying transport provider.

Issuing a t_close() function on a connection endpoint may cause data previously sent, or data not yet received, to be lost. It is the responsibility of the transport user to ensure that data is received by the remote peer.

VALID STATES  
Legitimate states (see t_getstate(3N)) for a call to this routine are every one except T_UNINIT.

RETURN VALUES  
t_close returns:

0  On success.

-1  On failure.

On failure, t_errno is set to indicate the error, and possibly errno is set.

ERRORS  
On failure, t_errno is set to the following:

TBADF  The specified file descriptor does not refer to a transport endpoint.

TPROTO  This error indicates that a communication problem has been detected between XTI and the transport provider for which there is no other suitable XTI t_errno.

TSYSERR  A system error occurred during execution of this function, errno will be

3N-1550  SunOS 5.6  modified 10 Feb 1997
Network Functions

TLI COMPATIBILITY

The XTI and TLI interface definitions have common names but use different header files. This, and other semantic differences between the two interfaces are described in the subsections below.

Interface Header

The XTI interfaces use the header file, xti.h. TLI interfaces should not use this header. They should use the header:

#include <tiuser.h>

Error Description

The t_errno value that can be set by the XTI interface and cannot be set by the TLI interface is:

TPROTO

For more information refer to the Transport Interfaces Programming Guide.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO

close(2), t_getstate(3N), t_open(3N), t_unbind(3N), attributes(5)

Transport Interfaces Programming Guide

modified 10 Feb 1997
SunOS 5.6
3N-1551
NAME  
\hspace{1em} t_connect – establish a connection with another transport user

SYNOPSIS  
\hspace{1em} cc [ flag . . .] file . . . -lnsl [ library . . .]
\hspace{1em} #include <xti.h>
\hspace{1em} int t_connect(int fd, const struct t_call *sndcall, struct t_call *rcvcall);

DESCRIPTION  
This routine is part of the XTI interfaces which evolved from the TLI interfaces. XTI represents the future evolution of these interfaces. However, TLI interfaces are supported for compatibility. When using a TLI routine that has the same name as an XTI routine, a different header file, tiuser.h, must be used. Refer to the section, TLI COMPATIBILITY, for a description of differences between the two interfaces.

This function enables a transport user to request a connection to the specified destination transport user. This function can only be issued in the T_IDLE state. The parameter \textit{fd} identifies the local transport endpoint where communication will be established, while \textit{sndcall} and \textit{rcvcall} point to a \textit{t_call} structure, which contains the following members:

\begin{verbatim}
struct netbuf addr;
struct netbuf opt;
struct netbuf udata;
int sequence;
\end{verbatim}

\textit{sndcall} specifies information needed by the transport provider to establish a connection and \textit{rcvcall} specifies information that is associated with the newly established connection.

The address is specified in the \textit{netbuf} structure which has the following format:

\begin{verbatim}
struct netbuf {
    unsigned int maxlen;
    unsigned int len;
    char *buf;
}
\end{verbatim}

where \textit{maxlen} specifies the maximum length of the buffer in bytes, \textit{len} specifies the bytes of data in the buffer, and \textit{buf} points to the buffer that contains the data.

In \textit{sndcall}, \textit{addr} specifies the protocol address of the destination transport user, \textit{opt} presents any protocol-specific information that might be needed by the transport provider, \textit{udata} points to optional user data that may be passed to the destination transport user during connection establishment, and \textit{sequence} has no meaning for this function.

On return, the \textit{addr} field in \textit{rcvcall} contains the protocol address associated with the responding transport endpoint, \textit{opt} represents any protocol-specific information associated with the connection, \textit{udata} points to optional user data that may be returned by the destination transport user during connection establishment, and \textit{sequence} has no meaning for this function.

The \textit{opt} argument permits users to define the options that may be passed to the transport provider. These options are specific to the underlying protocol of the transport provider and are described in protocol-specific documentation. The user may choose not to
negotiate protocol options by setting the \texttt{len} field of \texttt{opt} to zero. In this case, the provider uses the values currently set for the communications endpoint.

If used, \texttt{sndcall->opt.buf} must point to a buffer with the corresponding options, and \texttt{sndcall->opt.len} must specify its length. The \texttt{maxlen} and \texttt{buf} fields of the \texttt{netbuf} structure pointed by \texttt{rcvcall->addr} and \texttt{rcvcall->opt} must be set before the call.

The \texttt{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 \texttt{connect} field of the \texttt{info} argument of \texttt{t_open(3N)} or \texttt{t_getinfo(3N)}. If the \texttt{len} field of \texttt{udata} in the \texttt{t_call} structure referenced by \texttt{sndcall} is zero, no data will be sent to the destination transport user.

On return, the \texttt{addr}, \texttt{opt}, and \texttt{udata} fields of \texttt{rcvcall} will be updated to reflect values associated with the connection. Thus, the \texttt{maxlen} (see \texttt{netbuf} in \texttt{t_connect()}) field of each argument must be set before issuing this function to indicate the maximum size of the buffer for each. However, \texttt{maxlen} can be set to zero, in which case no information for this specific argument will be given to the user when \texttt{t_connect()} returns. If \texttt{rcvcall} is set to \texttt{NULL}, no information at all is returned.

By default, \texttt{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 \texttt{O_NONBLOCK} is set using \texttt{t_open()} or \texttt{fcntl(2)}, \texttt{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, returning \texttt{-1} with \texttt{t_errno} set to \texttt{TNODATA} to indicate that the connection has not yet been established. In this way, the function simply initiates the connection establishment procedure by sending a connection request to the destination transport user.

The \texttt{t_rcvconnect(3N)} function is used in conjunction with \texttt{t_connect()} to determine the status of the requested connection.

When a synchronous \texttt{t_connect()} call is interrupted by the arrival of a signal, the state of the corresponding transport endpoint is \texttt{T_OUTCON}, allowing a further call to either \texttt{t_rcvconnect()}, \texttt{t_rcvdis(3N)}, or \texttt{t_snddis(3N)}. When an asynchronous \texttt{t_connect()} call is interrupted by the arrival of a signal, the state of the corresponding transport endpoint is \texttt{T_IDLE}.

### VALID STATES

The only legitimate state (see \texttt{t_getstate(3N)}) for a call to this routine is \texttt{T_IDLE}.

### RETURN VALUES

\texttt{t_connect()} returns:

- \texttt{0} \quad \text{On success.}
- \texttt{-1} \quad \text{On failure.}

On failure, \texttt{t_errno} is set to indicate the error, and possibly \texttt{errno} is set.
ERRORS

On failure, \texttt{t_errno} is set to one of the following:

\textbf{TACCES} The user does not have permission to use the specified address or options.

\textbf{TADDRBUSY} This transport provider does not support multiple connections with the same local and remote addresses. This error indicates that a connection already exists.

\textbf{TBADADDR} The specified protocol address was in an incorrect format or contained illegal information.

\textbf{TBADDATA} The amount of user data specified was not within the bounds allowed by the transport provider.

\textbf{TBADF} The specified file descriptor does not refer to a transport endpoint.

\textbf{TBADOPT} The specified protocol options were in an incorrect format or contained illegal information.

\textbf{TBUFOVFLW} The number of bytes allocated for an incoming argument (\texttt{maxlen}) is greater than 0 but not sufficient to store the value of that argument. If executed in synchronous mode, the provider's state, as seen by the user, changes to \texttt{T\_DATAXFER}, and the information to be returned in \texttt{rcvcall} is discarded.

\textbf{TLOOK} An asynchronous event has occurred on this transport endpoint and requires immediate attention.

\textbf{TNODATA} \texttt{O\_NONBLOCK} was set, so the function successfully initiated the connection establishment procedure, but did not wait for a response from the remote user.

\textbf{TNOTSUPPORT} This function is not supported by the underlying transport provider.

\textbf{TOUTSTATE} The communications endpoint referenced by \texttt{fd} or \texttt{resfd} is not in one of the states in which a call to this function is valid.

\textbf{TPROTO} This error indicates that a communication problem has been detected between XTI and the transport provider for which there is no other suitable XTI \texttt{t_errno} value.

\textbf{TSYSERR} A system error has occurred during execution of this function, \texttt{errno} will be set to the specific error.

\textbf{TLI COMPATIBILITY} The XTI and TLI interface definitions have common names but use different header files. This, and other semantic differences between the two interfaces are described in the subsections below.

\textbf{Interface Header} The XTI interfaces use the header file, \texttt{xti.h}. TLI interfaces should not use this header. They should use the header:

SunOS 5.6 modified 10 Feb 1997
Error Description Values
The t_errno values that can be set by the XTI interface and cannot be set by the TLI interface are:

TPROTO
TADDRBUSY

A t_errno value that this routine can return under different circumstances than its XTI counterpart is TBUFOVFLW. It can be returned even when the maxlen field of the corresponding buffer has been set to zero.

Option Buffers
The format of the options in an opt buffer is dictated by the transport provider. Unlike the XTI interface, the TLI interface does not fix the buffer format.

For more information refer to the Transport Interfaces Programming Guide.

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
fcntl(2), t_accept(3N), t_alloc(3N), t_getinfo(3N), t_getstate(3N), t_listen(3N), t_open(3N), t_optmgmt(3N), t_rcvconnect(3N), t_rcvdis(3N), t_snddis(3N), attributes(5)

Transport Interfaces Programming Guide
NAME  tcsendbreak – send a “break” for a specific duration

SYNOPSIS  
#include <termios.h>

int tcsendbreak(int fd, int duration);

DESCRIPTION  The fd argument is an open file descriptor associated with a terminal. If the terminal is using asynchronous serial data transmission, tcsendbreak() will cause transmission of a continuous stream of zero-valued bits for a specific duration. If duration is 0, it will cause transmission of zero-valued bits for at least 0.25 seconds, and not more than 0.5 seconds. If duration is not 0, it behaves in a way similar to tcdrain(3). If the terminal is not using asynchronous serial data transmission, it sends data to generate a break condition or returns without taking any action.

Attempts to use tcsendbreak() from a process which is a member of a background process group on a fd associated with its controlling terminal will cause the process group to be sent a SIGTTOU signal. If the calling process is blocking or ignoring SIGTTOU signals, the process is allowed to perform the operation, and no signal is sent.

RETURN VALUES  Upon successful completion, 0 is returned. Otherwise, –1 is returned and errno is set to indicate the error.

ERRORS  The tcsendbreak() function will fail if:

EBADF  The fd argument is not a valid file descriptor.
ENOTTY  The file associated with fd is not a terminal.

The tcsendbreak() function may fail if:

EIO  The process group of the writing process is orphaned, and the writing process is not ignoring or blocking SIGTTOU.

ATTRIBUTES  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe, and Async-Signal-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  tcdrain(3), attributes(5), termio(7I)

3-1556  SunOS 5.6  modified 29 Dec 1996
NAME
tcsetattr – set the parameters associated with the terminal

SYNOPSIS
#include <termios.h>
int tcsetattr(int fildes, int optional_actions, const struct termios *termios_p);

DESCRIPTION
The tcsetattr() function sets the parameters associated with the terminal referred to by
the open file descriptor fildes (an open file descriptor associated with a terminal) from the
termios structure (see termio(7I)) referenced by termios_p as follows:
• If optional_actions is TCSANOW, the change will occur immediately.
• If optional_actions is TCSADRAIN, the change will occur after all output written to fildes
  is transmitted. This function should be used when changing parameters that affect output.
• If optional_actions is TCSAFLUSH, the change will occur after all output written to fildes
  is transmitted, and all input so far received but not read will be discarded before the
  change is made.

If the output baud rate stored in the termios structure pointed to by termios_p is the zero
baud rate, B0, the modem control lines will no longer be asserted. Normally, this will
disconnect the line.

If the input baud rate stored in the termios structure pointed to by termios_p is 0, the
input baud rate given to the hardware will be the same as the output baud rate stored in
the termios structure.

The tcsetattr() function will return successfully if it was able to perform any of the
requested actions, even if some of the requested actions could not be performed. It will
set all the attributes that implementation supports as requested and leave all the attri-
butes not supported by the implementation unchanged. If no part of the request can be
honoured, it will return −1 and set errno to EINVAL. If the input and output baud rates
differ and are a combination that is not supported, neither baud rate is changed. A subse-
quent call to tcgetattr(3) will return the actual state of the terminal device (reflecting both
the changes made and not made in the previous tcsetattr() call). The tcsetattr() function
will not change the values in the termios structure whether or not it actually accepts
them.

The effect of tcsetattr() is undefined if the value of the termios structure pointed to by
termios_p was not derived from the result of a call to tcgetattr(3) on fildes; an application
should modify only fields and flags defined by this document between the call to
tcgetattr(3) and tcsetattr(), leaving all other fields and flags unmodified.

No actions defined by this document, other than a call to tcsetattr() or a close of the last
file descriptor in the system associated with this terminal device, will cause any of the ter-
minal attributes defined by this document to change.

Attempts to use tcsetattr() from a process which is a member of a background process
group on a fildes associated with its controlling terminal, will cause the process group to
be sent a SIGTTOU signal. If the calling process is blocking or ignoring SIGTTOU signals,
the process is allowed to perform the operation, and no signal is sent.
RETURN VALUES
Upon successful completion, 0 is returned. Otherwise, –1 is returned and errno is set to indicate the error.

ERRORS
The tcsetattr() function will fail if:

- EBADF  The fd argument is not a valid file descriptor.
- EINTR  A signal interrupted tcsetattr().
- EINVAL  The optional_actions argument is not a supported value, or an attempt was made to change an attribute represented in the termios structure to an unsupported value.
- ENOTTY  The file associated with fd is not a terminal.

The tcsetattr() function may fail if:

- EIO  The process group of the writing process is orphaned, and the writing process is not ignoring or blocking SIGTTOU.

USAGE
If trying to change baud rates, applications should call tcsetattr() then call tcgetattr(3) in order to determine what baud rates were actually selected.

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe, and Async-Signal-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
cfgetispeed(3), tcgetattr(3), attributes(5), termio(7I)
NAME
tcsetpgrp – set foreground process group ID

SYNOPSIS
#include <sys/types.h>
#include <unistd.h>
int tcsetpgrp(int fildes, pid_t pgid_id);

DESCRIPTION
If the process has a controlling terminal, tcsetpgrp() will set the foreground process
group ID associated with the terminal to pgid_id. The file associated with fildes 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 pgid_id must
match a process group ID of a process in the same session as the calling process.

RETURN VALUES
Upon successful completion, 0 is returned. Otherwise, −1 is returned and errno is set to
indicate the error.

ERRORS
The tcsetpgrp() function will fail if:
EBADF The fildes argument is not a valid file descriptor.
EINVAL This implementation does not support the value in the pgid_id argument.
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 The value of pgid_id does not match the process group ID of a process in the
same session as the calling process.

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe, and Async-Signal-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
tcgetpgrp(3), attributes(5), termio(7)
NAME
tcsetpgrp – set foreground process group ID of terminal

SYNOPSIS
#include <unistd.h>
int tcsetpgrp(int fd, pid_t pgid);

DESCRIPTION
The tcsetpgrp() function sets the foreground process group ID of the terminal specified by fd to pgid. The file associated with fd 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 pgid must match a process group ID of a process in the same session as the calling process.

RETURN VALUES
Upon successful completion, tcsetpgrp() returns 0. Otherwise, −1 is returned and errno is set to indicate the error.

ERRORS
The tcsetpgrp() function fails if one or more of the following is true:
EBADF The fd argument is not a valid file descriptor.
EINVAL The fd argument is a terminal that does not support tcsetpgrp(), or pgid is not a valid process group ID.
EIO The process is not ignoring or holding SIGTTOU and is a member of an orphaned process group.
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 pgid does not match the process group ID of an existing process in the same session as the calling process.

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
attributes(5), termio(7I)
NAME      td_init – performs initialization for libthread_db library of interfaces

SYNOPSIS  cc [flag ...] file ... /lib/libthread_db.so.1 [library ... ]
#include <proc_service.h>
#include <thread_db.h>

void td_init();

DESCRIPTION  td_init() is the global initialization function for the libthread_db library of interfaces. It
must be called exactly once by any process using the libthread_db library before any
other libthread_db function can be called.

RETURN VALUES  TD_OK    The libthread_db library of interfaces successfully initialized.
TD_ERR    Initialization failed.

ATTRIBUTES  See attributes(5) for description of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO    libthread_db(3T), libthread_db(4), attributes(5)
NAME          td_log – placeholder for future logging functionality

SYNOPSIS      cc [ flag ... ] file ... /lib/libthread_db.so.1 [ library ... ]
              #include <proc_service.h>
              #include <thread_db.h>
              void td_log();

DESCRIPTION   This function presently does nothing; it is merely a placeholder for future logging functionality in libthread_db(3T).

ATTRIBUTES    See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO      libthread(3T), libthread_db(3T), libthread_db(4), attributes(5)
NAME

td_sync_get_info, td_sync_setstate, td_sync_waiters – operations on a synchronization object in libthread_db

SYNOPSIS

cc [flag ...] file ... /lib/libthread_db.so.1 [ library ... ]
#include <proc_service.h>
#include <thread_db.h>

#include <sys/service.h>

#include <proc_service.h>
#include <thread_db.h>

# include <proc_service.h>
#include <thread_db.h>


td_err_e td_sync_get_info(const td_synchandle_t *sh_p, td_syncinfo_t *si_p);

td_err_e td_sync_setstate(const td_synchandle_t *sh_p, int value);

td_err_e td_sync_waiters(const td_synchandle_t *sh_p, td_thr_iter_f *cb, 
void *cb_data_p);

DESCRIPTION

Synchronization objects include mutexes, condition variables, semaphores, and reader-writer locks. In the same way that thread operations use a thread handle of type td_thrhandle_t, operations on synchronization objects use a synchronization object handle of type td_synchandle_t.

The controlling process obtains synchronization object handles either by calling the function td_ta_sync_iter() to obtain handles for all synchronization objects of the target process that are known to the libthread_db library of interfaces, or by mapping the address of a synchronization object in the address space of the target process to a handle by calling td_ta_map_addr2sync().

Note that not all synchronization objects that a process uses may be known to the libthread_db library and returned by td_ta_sync_iter. A synchronization object is known to libthread_db only if it was ever waited on after libthread_db was attached to the process. For example, a mutex may have been widely used, but if no thread ever blocked waiting to acquire it, it will not be known to libthread_db interfaces.

td_sync_get_info()

Fills in the td_syncinfo_t structure *si_p with values for the synchronization object identified by sh_p. The td_syncinfo_t structure contains the following fields:

  td_thragent_t *si_ta_p
  The internal process handle identifying the target process through which this synchronization object handle was obtained. Synchronization objects may be process-private or process-shared. In the latter case, the same synchronization object may have multiple handles, one for each target process’s “view” of the synchronization object.

  psaddr_t si_sv_addr
  The address of the synchronization object in this target process’s address space.

  td_sync_type_e si_type
  The type of the synchronization variable: mutex, condition variable, semaphore, or reader-writer lock.
int si_shared_type
    USYNC_THREAD if this synchronization object is process-private;
    USYNC_PROCESS if it is process-shared.

td_sync_flags_t si_flags
    Flags dependent on the type of the synchronization object.

int si_state.sema_count
    Semaphores only. The current value of the semaphore.

int si_state.nreaders
    Reader-writer locks only. The number of readers currently holding
    the lock, or -1, if a writer is currently holding the lock.

int si_state.mutex_locked
    For mutexes only. Non-zero if and only if the mutex is currently
    locked.

int si_size
    The size of the synchronization object.

uchar_t si_has_waiters
    Non-zero if and only if at least one thread is blocked on this syn-
    chronization object.

uchar_t si_is_wlocked
    For reader-writer locks only. The value is non-zero if and only if this
    lock is held by a writer.

td_thrhandle_t si_owner
    Mutexes and reader-writer locks only. This is the thread holding the
    mutex, or the write lock, if this is a reader-writer lock. The value is
    NULL if no one holds the mutex or write-lock.

psaddr_t si_data
    A pointer to optional data associated with the synchronization object.
    Currently useful only for debugging libthread() interfaces.

**td_sync_setstate** modifies the state of synchronization object *si_p*, depending on the syn-
chronization object type. For mutexes, **td_sync_setstate** is unlocked if the value is 0. Oth-
erwise it is locked. For semaphores, the semaphore's count is set to the value. For
reader-writer locks, the reader count set to the value if value is >0. The count is set to
write-locked if value is −1. It is set to unlocked if the value is 0.
Setting the state of a synchronization object from a **libthread_db** interface may cause the
synchronization object’s semantics to be violated from the point of view of the threads in
the target process. For example, if a thread holds a mutex, and **td_sync_setstate** is used
to set the mutex to unlocked, then a different thread will also be able to subsequently
acquire the same mutex.

**td_sync_waiters** iterates over the set of thread handles of threads blocked on *sh_p*. The
callback function *cb* is called once for each such thread handle, and is passed the thread
handle and *cb_data_p*. If the callback function returns a non-zero value, iteration is ter-
minated early. See also **td_ta_thr_iter**(3T).
RETURN VALUES

- **TD_OK**: The call returned successfully.
- **TD_BADTH**: An invalid thread handle was passed in.
- **TD_DBERR**: A call to one of the imported interface routines failed.
- **TD_ERR**: A libthread_db-internal error occurred.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO

libthread_db(3T), td_ta_map_addr2sync(3T), td_ta_sync_iter(3T), td_ta_thr_iter(3T), libthread_db(4), attributes(5)
NAME
td_ta_enable_stats, td_ta_reset_stats, td_ta_get_stats – collect target process statistics for
libthread_db

SYNOPSIS
cc [flag ...] file ... /lib/libthread_db.so.1 [library ...]
#include <proc_service.h>
#include <thread_db.h>
td_err_e td_ta_enable_stats(const td_thragent_t *ta_p,
   int on_off);
td_err_e td_ta_reset_stats(const td_thragent_t *ta_p);
td_err_e td_ta_get_stats(const td_thragent_t *ta_p,
   td_ta_stats_t *tstats);

DESCRIPTION
The controlling process may request the collection of certain statistics about a target pro-
çess. Statistics gathering is disabled by default; however, each target process has a
td_ta_stats_t structure that contains up to date values when statistic gathering is enabled.
td_ta_enable_stats() turns statistics gathering on or off for the process identified by ta_p
depending on whether or not on_off is non-zero. When statistics gathering is turned on,
all statistics are implicitly reset as though td_ta_reset_stats() had been called. Statistics
are not reset when statistics gathering is turned off. Except for nthreads and
r_concurrency, the values do not change further, but they remain available for inspection
by way of td_ta_get_stats(). td_ta_reset_stats() resets all counters in the td_ta_stats_t
structure to zero for the target process. td_ta_get_stats() returns the td_ta_stats_t struc-
ture for the process in *stats_t. The td_ta_stats_t structure is defined as follows:

typedef struct {
   int nthreads;    /* total number of threads in use */
   int r_concurrency; /* requested concurrency level */
   int nrunnable_num; /* numerator of avg. runnable threads */
   int nrunnable_den; /* denominator of avg. runnable threads */
   int a_concurrency_num; /* numerator, avg. achieved concurrency */
   int a_concurrency_den; /* denominator, avg. achieved concurrency */
   int nlwps_num;    /* numerator, average number of LWP's in use */
   int nlwps_den;    /* denominator, avg. number of LWP's in use */
   int nidle_num;    /* numerator, avg. number of idling LWP's */
   int nidle_den;    /* denominator, avg. number of idling LWP's */
} td_ta_stats_t;

nthreads is the number of threads that are currently part of the target process.
r_concurrency is the current requested concurrency level, such as would be returned by
thr_setconcurrency(3T). The remaining fields are averages over time, each expressed as
a fraction with an integral numerator and denominator. nrunnable is the average
number of runnable threads. a_concurrency is the average achieved concurrency, the
number of actually running threads. a_concurrency is less than or equal to nrunnable.

nlwps is the average number of lightweight processes (LWP's) participating in this
process. It must be greater than or equal to `a_concurrency`, as every running thread is assigned to an LWP, but there may at times be additional idling LWP’s with no thread assigned to them. `nidle` is the average number of idle LWP’s.

**RETURN VALUES**

- **TD_OK** The call completed successfully.
- **TD_BADTA** An invalid internal process handle was passed in.
- **TD_DBERR** A call to one of the imported interface routines failed.
- **TD_ERR** Something else went wrong.

**ATTRIBUTES**

See [attributes(5)] for description of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

**SEE ALSO**

libthread_db(3T), thr_getconcurrency(3T), libthread_db(4), attributes(5)
<table>
<thead>
<tr>
<th>NAME</th>
<th>td_ta_event_addr, td_thr_event_enable, td_ta_set_event, td_thr_set_event, td_ta_clear_event, td_thr_clear_event, td_ta_event_getmsg, td_thr_event_getmsg, td_event_emptyset, td_event_fillset, td_event_addset, td_event_delset, td_eventismember, td_eventisempty – thread events in libthread_db</th>
</tr>
</thead>
</table>
| SYNOPSIS | cc [ flag ... ] file ... /lib/libthread_db.so.1 [ library ... ]  
#include <proc_service.h>  
#include <thread_db.h>  

```
    td_err_e td_ta_event_addr(const td_thragent_t *ta_p,  
                              u_long event, td_notify_t *notify_p);
    td_err_e td_thr_event_enable(const td_thrhandle_t *th_p,  
                                  int onoff);
    td_err_e td_thr_set_event(const td_thrhandle_t *th_p,  
                              td_thr_events_t *events);
    td_err_e td_ta_set_event(const td_thragent_t *ta_p,  
                              td_thr_events_t *events);
    td_err_e td_thr_clear_event(const td_thrhandle_t *th_p,  
                               td_thr_events_t *events);
    td_err_e td_ta_clear_event(const td_thragent_t *ta_p,  
                               td_thr_events_t *events);
    td_err_e td_thr_event_getmsg(const td_thrhandle_t *th_p,  
                                 td_event_msg_t *msg);
    td_err_e td_ta_event_getmsg(const td_thragent_t *ta_p,  
                               td_event_msg_t *msg);
    void td_event_emptyset(td_thr_events_t *)
    void td_event_fillset(td_thr_events_t *);
    void td_event_addset(td_thr_events_t *,  
                         td_thr_events_t *e n);
    void td_event_delset(td_thr_events_t *,  
                         td_thr_events_t *e n);
    void td_eventismember(td_thr_events_t *,  
                          td_thr_events_t *e n);
    void td_eventisempty(td_thr_events_t *);
```

| DESCRIPTION | These routines comprise the thread event facility for **libthread_db**(3T). This facility allows the controlling process to be notified when certain thread-related events occur in a target process and to retrieve information associated with these events. An event consists of an event type, and optionally, some associated event data, depending on the event type. See the section titled "Event Set Manipulation Macros" that follows. |
The event type and the associated event data, if any, constitute an "event message." Reporting an event means delivering an event message to the controlling process by way of **libthread_db**.

Several flags can control event reporting, both a per-thread and per event basis. Event reporting may further be enabled or disabled for a thread. There is not only a per-thread event mask that specifies which event types should be reported for that thread, but there is also a global event mask that applies to all threads.

An event is reported, if and only if, the executing thread has event reporting enabled, and either the event type is enabled in the executing thread’s event mask, or the event type is enabled in the global event mask.

Each thread has associated with it an event buffer in which it stores the most recent event message it has generated, the type of the most recent event that it reported, and, depending on the event type, some additional information related to that event. See the section titled "Event Set Manipulation Macros" for a description of the **td_thr_events_e** and **td_event_msg_t** types and a list of the event types and the values reported with them.

The thread handle, type **td_thrhandle_t**, the event type, and the possible value, together constitute an event message. Each thread’s event buffer holds at most one event message.

Each event type has an event reporting address associated with it. A thread reports an event by writing the event message into the thread’s event buffer and having control reach the event reporting address for that event type.

Typically, the controlling process sets a breakpoint at the event reporting address for one or more event types. When the breakpoint is hit, the controlling process knows that an event of the corresponding type has occurred.

The event types, and the additional information, if any, reported with each event, are:

- **TD_READY** The thread became ready to execute.
- **TD_SLEEP** The thread has blocked on a synchronization object.
- **TD_SWITCHTO** A runnable thread is being assigned to LWP.
- **TD_SWITCHFROM** A running thread is being removed from its LWP.
- **TD_LOCK_TRY** A thread is trying to get an unavailable lock.
- **TD_CATCHSIG** A signal was posted to a thread.
- **TD_IDLE** An LWP is becoming idle.
- **TD_CREATE** A thread is being created.
- **TD_DEATH** A thread has terminated.
- **TD_PREEMPT** A thread is being preempted.
- **TD_PRI_INHERIT** A thread is inheriting an elevated priority from another thread.
- **TD_REAP** A thread is being reaped.
- **TD_CONCURRENCY** The number of LWPs is changing.
TD_TIMEOUT

A condition-variable timed wait expired.

td_ta_event_addr() returns in *notify_p the event reporting address associated with event type event. The controlling process may then set a breakpoint at that address. If a thread hits that breakpoint, it reports an event of type event.

td_thr_event_enable() enables or disables event reporting for thread th_p. If a thread has event reporting disabled, it will not report any events. Threads are started with event reporting disabled. Event reporting is enabled if onoff is non-zero; otherwise, it is disabled. To find out whether or not event reporting is enabled on a thread, call td_thr_getinfo() for the thread and examine the ti_traceme field of the td_thrinfo_t structure it returns.

td_thr_set_event() and td_thr_clear_event() set and clear, respectively, a set of event types in the event mask associated with the thread th_p. To inspect a thread’s event mask, call td_thr_getinfo() for the thread, and examine the ti_events field of the td_thrinfo_t structure it returns.

td_ta_set_event() and td_ta_clear_event() are just like td_thr_set_event() and td_thr_clear_event(), respectively, except that the target process’s global event mask is modified. There is no provision for inspecting the value of a target process’s global event mask.

td_thr_event_getmsg() returns in *msg the event message associated with thread *th_p. Reading a thread’s event message consumes the message, emptying the thread’s event buffer. As noted above, each thread’s event buffer holds at most one event message; if a thread reports a second event before the first event message has been read, the second event message overwrites the first.

td_ta_event_getmsg() is just like td_thr_event_getmsg(), except that it is passed a process handle rather than a thread handle. It selects some thread that has an event message buffered, and it returns that thread’s message. The thread selected is undefined, except that as long as at least one thread has an event message buffered, it will return an event message from some such thread.

Event Set Manipulation Macros

Several macros are provided for manipulating event sets of type td_thr_events_t:

td_event_emptyset Sets its argument to the NULL event set.

td_event_fillset Sets its argument to the set of all events.

td_event_addset Adds a specific event type to an event set.

td_event_delsset Deletes a specific event type from an event set.

td_eventismember Tests whether a specific event type is a member of an event set.

td_eventisempty Tests whether an event set is the NULL set.

RETURN VALUES

The following values may be returned for all thread event routines:

TD_OK The call returned successfully.

TD_BADTH An invalid thread handle was passed in.

TD_BADTA An invalid internal process handle was passed in.
Threads Library

dtd_ta_event_addr(3T)

<table>
<thead>
<tr>
<th>CODE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>TD_BADPH</td>
<td>There is a NULL external process handle associated with this internal process handle.</td>
</tr>
<tr>
<td>TD_DBERR</td>
<td>A call to one of the imported interface routines failed.</td>
</tr>
<tr>
<td>TD_NOMSG</td>
<td>No event message was available to return to td_thr_event_getmsg() or td_ta_event_getmsg().</td>
</tr>
<tr>
<td>TD_ERR</td>
<td>Some other parameter error occurred, or a libthread_db internal error occurred.</td>
</tr>
</tbody>
</table>

The following value may be returned for td_thr_event_enable(), td_thr_set_event(), and td_thr_clear_event() only:

<table>
<thead>
<tr>
<th>CODE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>TD_NOCAPAB</td>
<td>The agent thread in the target process has not completed initialization, so this operation cannot be performed. The operation can be performed after the target process has been allowed to make some forward progress. See also libthread_db(3T).</td>
</tr>
</tbody>
</table>

ATTRIBUTES

See attributes(5) for description of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO libthread_db(3T), libthread_db(4), attributes(5)
NAME

`td_ta_get_nthreads` - gets the total number of threads in a process for libthread_db

SYNOPSIS

```c
cc [ flag ...] file ... /lib/libthread_db.so.1 [ library ... ]
#include <proc_service.h>
#include <thread_db.h>

td_err_e td_ta_get_nthreads(const td_thragent_t *ta_p,
    int *nthread_p);
```

DESCRIPTION

`td_ta_get_nthreads` returns the total number of threads in process `ta_p`, including any system threads. System threads are those created by `libthread` or `libthread_db` on its own behalf. The number of threads is written into `*nthread_p`.

RETURN VALUES

- **TD_OK** - The call completed successfully.
- **TD_BADTA** - An invalid internal process handle was passed in.
- **TD_BADPH** - There is a NULL external process handle associated with this internal process handle.
- **TD_DBERR** - A call to one of the imported interface routines failed.
- **TD_ERR** - `nthread_p` was NULL, or a `libthread_db` internal error occurred.

ATTRIBUTES

See attributes(5) for description of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO

`libthread(3T)`, `libthread_db(3T)`, `libthread_db(4)`, attributes(5)
NAME  td_ta_map_addr2sync – get a synchronization object handle from a synchronization object’s address

SYNOPSIS  cc [ flag ... ] file ... /lib/libthread_db.so.1 [ library ... ]
#include <proc_service.h>
#include <thread_db.h>

#include <thread_db.h>

dt_ta_map_addr2sync(const td_thragent_t *ta_p, psaddr_t addr,
                     td_synchandle_t *sh_p);

DESCRIPTION  td_ta_map_addr2sync( ) produces the synchronization object handle of type td_synchandle_t that corresponds to the address of the synchronization object (mutex, semaphore, condition variable, or reader/writer lock). Some effort is made to validate addr and verify that it does indeed point at a synchronization object. The handle is returned in *sh_p.

RETURN VALUES  TD_OK  The call completed successfully.
TD_BADTA  An invalid internal process handle was passed in.
TD_BADPH  There is a NULL external process handle associated with this internal process handle.
TD_BADSH  sh_p is NULL, or addr does not appear to point to a valid synchronization object.
TD_DBERR  A call to one of the imported interface routines failed.
TD_ERR  addr is NULL, or a libthread_db internal error occurred.

ATTRIBUTES  See attributes(5) for description of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  libthread_db(3T), libthread_db(4), attributes(5)
td_ta_map_id2thr (3T)

NAME
td_ta_map_id2thr, td_ta_map_lwp2thr – convert a thread id or LWP id to a thread handle

SYNOPSIS
c [flag ...] file ... /lib/libthread_db.so.1 [library ...]
#include <proc_service.h>
#include <thread_db.h>

td_ta_map_id2thr(const td_thragent_t *ta_p,
                  thread_t tid, td_thrhandle_t *th_p);

td_ta_map_lwp2thr(const td_thragent_t *ta_p,
                  lwpid_t lwpid, td_thrhandle_t *th_p);

DESCRIPTION
td_ta_map_id2thr() produces the td_thrhandle_t thread handle that corresponds to a particular thread id, as returned by thr_create(3T) or thr_self(3T). The thread handle is returned in *th_p.

td_ta_map_lwp2thr() produces the td_thrhandle_t thread handle for the thread that is currently executing on the light weight process (LWP) and has an id of lwpid.

RETURN VALUES
TD_OK The call completed successfully.
TD_BADTA An invalid internal process handle was passed in.
TD_BADPH There is a NULL external process handle associated with this internal process handle.
TD_DBERR A call to one of the imported interface routines failed.
TD_NOTHR Either there is no thread with the given thread id (td_ta_map_id2thr) or no thread is currently executing on the given LWP (td_ta_map_lwp2thr).
TD_ERR The call did not complete successfully.

ATTRIBUTES
See attributes(5) for description of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO libthread_db(3T), thr_create(3T), thr_self(3T), libthread_db(4), attributes(5)

3T-1574 SunOS 5.6 modified 6 Jan 1997
NAME

`td_ta_new`, `td_ta_delete`, `td_ta_get_ph` – allocate and deallocate process handles for `libthread_db`

SYNOPSIS

```c
cc [flag ...] file ... /lib/libthread_db.so.1 [ library ... ]
#include <proc_service.h>
#include <thread_db.h>

td_err_e td_ta_new(const struct ps_prochandle *ph_p, td_thragent_t **ta_pp);

#include <thread_db.h>

td_err_e td_ta_new(const struct ps_prochandle *ph_p, td_thragent_t **ta_pp);

#include <thread_db.h>

td_err_e td_ta_new(const struct ps_prochandle *ph_p, td_thragent_t **ta_pp);

td_err_e td_ta_delete(const td_thragent_t *ta_p);

td_err_e td_ta_get_ph(const td_thragent_t *ta_p, struct ps_prochandle **ph_pp);
```

DESCRIPTION

`td_ta_new()` registers a target process with `libthread_db` and allocates an internal process handle of type `td_thragent_t` for this target process. Subsequent calls to `libthread_db` can use this handle to refer to this target process.

There are actually two process handles, an internal process handle assigned by `libthread_db` and an external process handle assigned by the `libthread_db` client. There is a one-to-one correspondence between the two handles. When the client calls a `libthread_db` routine, it uses the internal process handle. When `libthread_db` calls one of the client-provided routines listed in `proc_service(3T)`, it uses the external process handle.

`ph` is the external process handle that `libthread_db` should use to identify this target process to the controlling process when it calls routines in the imported interface.

If this call is successful, the value of the newly allocated `td_thragent_t` handle is returned in `*ta_pp`. `td_ta_delete()` deregisters a target process with `libthread_db`, which deallocates its internal process handle and frees any other resources `libthread_db` has acquired with respect to the target process. `ta_p` specifies the target process to be deregistered.

`td_ta_get_ph()` returns in `*ph_pp` the external process handle that corresponds to the internal process handle `ta_p`. This is useful for checking internal consistency.

RETURN VALUES

- **TD_OK**: The call completed successfully.
- **TD_BADPH**: A NULL external process handle was passed in to `td_ta_new`.
- **TD_ERR**: `ta_pp` is NULL, or an internal error occurred.
- **TD_DBERR**: A call to one of the imported interface routines failed.
- **TD_MALLOC**: Memory allocation failure.
- **TD_NOLIBTHREAD**: The target process does not appear to be multithreaded.
### ATTRIBUTES

See attributes(5) for description of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

### SEE ALSO

libthread_db(3T), proc_service(3T), libthread_db(4), attributes(5)
NAME      td_ta_setconcurrency – set concurrency level for target process

SYNOPSIS  
cc [ flag ...] file ... /lib/libthread_db.so.1 [ library ... ]
#include <proc_service.h>
#include <thread_db.h>

dt_err_e td_ta_setconcurrency(const td_thragent_t *ta_p,  
int level);

DESCRIPTION td_ta_setconcurrency() sets the desired concurrency level for the process identified by  
ta_p to level, just as if a thread within the process had called thr_setconcurrency(). See thr_setconcurrency(3T).

RETURN VALUES  
TD_OK     The call completed successfully.
TD_BADTA  An invalid internal process handle was passed in.
TD_BADPH  There is a NULL external process handle associated with this internal  
process handle. TD_NOCAPAB The client did not implement the  
ps_kill() routine in the imported interface. See ps_kill(3T).
TD_DBERR  A call to one of the imported interface routines failed.
TD_ERR    A libthread_db internal error occurred.

ATTRIBUTES  See attributes(5) for description of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  
libthread_db(3T), ps_kill(3T), thr_setconcurrency(3T), libthread_db(4), attributes(5)
NAME
td_ta_sync_iter, td_ta_thr_iter, td_ta_tsd_iter – iterator functions on process handles from libthread_db library of interfaces

SYNOPSIS
c
cc [flag ...] file ... /lib/libthread_db.so.1 [library ...]
#include <proc_service.h>
#include <thread_db.h>

d_t_d_err_e td_ta_sync_iter(const td_thragent_t *ta_p,
    td_sync_iter_f *cb, void *cbdata_p);
d_t_d_err_e td_ta_tsd_iter(const td_thragent_t *ta_p,
    td_key_iter_f *cb, void *cbdata_p);
d_t_d_err_e td_ta_sync_iter(const td_thragent_t *ta_p,
    td_syn_iter_f *cb, void *cbdata_p);

DESCRIPTION
td_ta_sync_iter(), td_ta_thr_iter(), and td_ta_tsd_iter() are iterator functions that when given a target process handle as an argument, return sets of handles for objects associated with the target process. The method is to call back a client-provided function once for each associated object, passing back a handle as well as the client-provided pointer cb_data_p. This enables a client to easily build a linked list of the associated objects.

td_ta_sync_iter() returns handles of synchronization objects (mutexes, preader-writer locks, semaphores, and condition variables) associated with a process. Some synchronization objects may not be known to libthread_db and will not be returned. If the process has initialized the synchronization object (by calling mutex_init, for example) or a thread in the process has blocked on this object after libthread_db attached to the synchronization object, then a handle for the synchronization object will be returned by libthread_db. See td_sync_get_info(3T) to see operations that can be performed on synchronization object handles.

td_ta_thr_iter() returns handles for threads that are part of the target process. For td_ta_thr_iter, the caller specifies several criteria to select a subset of threads for which the callback function should be called. Any of these selection criteria may be wildcarded. If all of them are wild-carded, then handles for all threads in the process will be returned.

The selection parameters and corresponding wild-card values are:

state (TD_THR_ANY_STATE): Select only threads whose state matches state. See td_thr_get_info(3T) for a list of thread states.

ti_pri (TD_THR_LOWEST_PRIORITY): Select only threads for which the priority is at least ti_pri.

ti_sigmask_p (TD_SIGNO_MASK): Select only threads whose signal mask exactly matches *ti_sigmask_p.
**ti_user_flags (TD_THR_ANY_USER_FLAGS):**
Select only threads whose user flags (specified at thread creation time) exactly match `ti_user_flags`.

**td_ta_tsd_iter()** returns the thread-specific data keys in use by the current process. Thread-specific data for a particular thread and key may be obtained by calling `td_thr_tsd(3T)`.

**RETURN VALUES**
- **TD_OK** The call completed successfully.
- **TD_BADTA** An invalid process handle was passed in.
- **TD_DBERR** A call to one of the imported interface routines failed.
- **TD_ERR** The call did not complete successfully.

**ATTRIBUTES**
See [attributes(5)](3T) for description of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

**SEE ALSO**
[libthread_db(3T)](3T), [td_sync_get_info(3T)](3T), [td_thr_get_info(3T)](3T), [td_thr_tsd(3T)](3T), [libthread_db(4)](3T), [attributes(5)](3T)
td_thr_dbsuspend (3T)  Threads Library

NAME
td_thr_dbsuspend, td_thr_dbresume – suspend and resume threads in libthread_db

SYNOPSIS
cc [ flag . . . ] file . . . /lib/libthread_db.so.1 [ library . . . ]
#include <proc_service.h>
#include <thread_db.h>

td_err_e td_thr_dbsuspend(const td_thrhandle_t *th_p);

td_err_e td_thr_dbresume(const td_thrhandle_t *th_p);

DESCRIPTION
These operations suspend and resume the thread identified by th_p. A thread that has been suspended with td_thr_dbsuspend() is said to be in the "dbsuspended" state. A thread whose "dbsuspended" flag is set will not execute. If an unbound thread enters the "dbsuspended" state and is currently assigned to a lightweight process (LWP), then the LWP becomes available for assignment to a different thread.

A thread’s "dbsuspended" state is independent of the suspension state controlled by calls to thr_suspend(3T) and thr_continue(3T) from within the target process. Calling thr_continue(3T) within the target process on a thread that has been suspended during a call to td_thr_dbsuspend() will not cause that thread to resume execution; only a call to td_thr_dbresume() will do that.

RETURN VALUES
TD_OK The call completed successfully.
TD_BADTH An invalid thread handle was passed in.
TD_DBERR A call to one of the imported interface routines failed.
TD_NOCAPAB The "agent thread" in the target process has not completed initialization, so this operation cannot be performed. The operation can be performed after the target process has been allowed to make some forward progress. See also libthread_db(3T).
TD_ERR A libthread_db internal error occurred.

ATTRIBUTES
See attributes(5) for description of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
libthread_db(3T), thr_continue(3T), thr_suspend(3T), libthread_db(4), attributes(5)

3T-1580 SunOS 5.6 modified 6 Jan 1997
NAME  td_thr_getgregs, td_thr_setgregs, td_thr_getfpregs, td_thr_setfpregs, td_thr_getxregsize,
td_thr_getxregs, td_thr_setxregs – reading and writing thread registers in libthread_db

SYNOPSIS  cc [flag ...] file ... /lib/libthread_db.so.1 [ library ... ]
#include <proc_service.h>
#include <thread_db.h>

td_err_e td_thr_getgregs(const td_thrhandle_t *th_p, prgregset_t gregset);

td_err_e td_thr_setgregs(const td_thrhandle_t *th_p, prgregset_t gregset);

td_err_e td_thr_getfpregs(const td_thrhandle_t *th_p, prfpregset_t *fpregset);

td_err_e td_thr_setfpregs(const td_thrhandle_t *th_p, prfpregset_t *fpregset);

td_err_e td_thr_getxregsize(const td_thrhandle_t *th_p, int *xregsize);

td_err_e td_thr_getxregs(const td_thrhandle_t *th_p, prxregset_t *xregset);

td_err_e td_thr_setxregs(const td_thrhandle_t *th_p, prxregset_t *xregset);

DESCRIPTION  These routines read and write the register sets associated with thread th_p.
td_thr_getgregs() and td_thr_setgregs() get and set, respectively, the general registers of
thread th_p.  td_thr_getfpregs() and td_thr_setfpregs() get and set, respectively, the
thread’s floating point register set.  td_thr_getxregsize(), td_thr_getxregs(), and
td_thr_setxregs() are SPARC-specific.  td_thr_getxregsize() returns in *xregsize the size
of the architecture-dependent extra state registers.  td_thr_getxregs() and
td_thr_setxregs() get and set, respectively, those extra state registers.  On non-SPARC
architectures, these routines return TD_NOXREGS.
If thread th_p is currently executing on a lightweight process (LWP), these routines will
read or write, respectively, the appropriate register set to the LWP using the imported
interface.  If the thread is not currently executing on a LWP, then the floating point and
extra state registers may not be read or written.  Some of the general registers may also
not be readable or writable, depending on the architecture.  In this case,
td_thr_getfpregs() and td_thr_setfpregs() will return TD_NOFPREGS, and
td_thr_getxregs() and td_thr_setxregs() will return TD_NOXREGS.  Calls to
td_thr_getregs() and td_thr_setregs() will succeed, but values returned for unread-
able registers will be undefined, and values specified for unwritable registers will be
ignored.  In this instance, a value of TD_PARTIALREGS will be returned.  See the
architecture-specific notes that follow regarding the registers that may be read and written
for a thread not currently executing on a LWP.

SPARC  On a thread not currently assigned to a LWP, only %i0-%i7, %l0-%l7, %g7, %pc, and %sp
(%o6) may be read or written.  %pc and %sp refer to the program counter and stack
pointer that the thread will have when it resumes execution.

Intel x86  On a thread not currently assigned to a LWP, only %pc, %sp, %ebp, %edi, %esi, and %ebx
may be read.

modified 6 Jan 1997  SunOS 5.6  3T-1581
RETURN VALUES

TD_OK The call completed successfully.
TD_BADTH An invalid thread handle was passed in.
TD_DBERR A call to one of the imported interface routines failed.
TD_PARTIALREGS Because the thread is not currently assigned to a LWP, not all registers were read or written. See DESCRIPTION for a discussion about which registers are not saved when a thread is not assigned to an LWP.
TD_NOFPREGS Floating point registers could not be read or written, either because the thread is not currently assigned to an LWP, or because the architecture does not have such registers.
TD_NOXREGS Architecture-dependent extra state registers could not be read or written, either because the thread is not currently assigned to an LWP, or because the architecture does not have such registers, or because the architecture is not a SPARC architecture.
TD_ERR A libthread_db internal error occurred.

ATTRIBUTES

See attributes(5) for description of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO

libthread_db(3T), libthread_db(4), attributes(5)
NAME  
td_thr_get_info – get thread information in libthread_db library of interfaces

SYNOPSIS  
cc [ flag ... ] file ... /lib/libthread_db.so.1 [ library ... ]
#include <proc_service.h>
#include <thread_db.h>
td_err_t td_thr_get_info(const td_thrhandle_t *th_p,
                         td_thrinfo_t *ti_p);

DESCRIPTION  
The td_thr_get_info() routine fills in the td_thrinfo_t structure *ti_p with values for the
thread identified by th_p.

The td_thrinfo_t structure contains the following fields:

typedef struct td_thrinfo_t {
    td_thragent_t *ti_ta_p; /* internal process handle */
    unsigned ti_user_flags; /* value of flags parameter */
    thread_t ti_tid; /* thread identifier */
    char *ti_tls; /* pointer to thread-local storage */
    paddr ti_startfunc; /* address of function at which thread execution began */
    paddr ti_stkbase; /* base of thread’s stack area */
    int ti_stksize; /* size in bytes of thread’s allocated stack region */
    paddr ti_ro_area; /* address of uthread_t structure */
    int ti_ro_size /* size of the uthread_t structure in bytes */
    td_thr_state_e ti_state /* state of the thread */
    uchar_t ti_db_suspended /* non-zero if thread suspended by td_thr_dbsuspend */
    td_thr_type_e ti_type /* type of the thread */
    int ti_pc /* value of thread’s program counter */
    int ti_sp /* value of thread’s stack counter */
    short ti_flags /* set of special flags used by libthread */
    int ti_pri /* priority of thread returned by thr_getprio(3T) */
    lwpid_t ti_lid /* id of light weight process (LWP) executing this thread */
    sigset_t ti_sigmask /* thread’s signal mask. See thr_sigsetmask(3T) */
    u_char ti_traceme /* non-zero if event tracing is on */
    u_char_t ti_preemptflag /* non-zero if thread preempted when last active */
    u_char_t ti_pirecflag /* non-zero if thread runs priority beside regular */
    sigset_t ti_pending /* set of signals pending for this thread */
    td_thr_events_t ti_events /* bitmap of events enabled for this thread */
} ;


td_thragent_t *ti_ta_p is the internal process handle identifying the process of which the
thread is a member.

unsigned ti_user_flags is the value of the flags parameter passed to thr_create(3T) when
the thread was created.
thread_t ti_tid is the thread identifier for the thread returned by libthread when created with thr_create(3T).

char *ti_tls is the thread’s pointer to thread-local storage.

psaddr_t ti_startfunc is the address of the function at which thread execution began, as specified when the thread was created with thr_create(3T).

psaddr_t ti_stkbase is the base of the thread’s stack area.

int ti_stksize is the size in bytes of the thread’s allocated stack region.

psaddr_t ti_ro_area is the address of the libthread-internal uthread_t structure for this thread. Since accessing the uthread_t structure directly violates the encapsulation provided by libthread_db, this field should generally not be used. However, it may be useful as a prototype for extensions.

td_thr_state_e ti_state is the state in which the thread is. The td_thr_state_e enumeration type may contain the following values:

- **TD_THR_ANY_STATE**
  - Never returned by td_thr_get_info.
  - TD_THR_ANY_STATE is used as a wildcard to select threads in td_ta_thr_iter().

- **TD_THR_UNKNOWN**
  - libthread_db cannot determine the state of the thread.

- **TD_THR_STOPPED**
  - The thread has been stopped by a call to thr_suspend(3T).

- **TD_THR_RUN**
  - The thread is runnable, but it is not currently assigned to a LWP.

- **TD_THR_ACTIVE**
  - The thread is currently executing on a LWP.

- **TD_THR_ZOMBIE**
  - The thread has exited, but it has not yet been deallocated by a call to thr_join(3T).

- **TD_THR_SLEEP**
  - The thread is not currently runnable.

- **TD_THR_STOPPED_ASLEEP**
  - The thread is both blocked by TD_THR_SLEEP, and stopped by a call to td_thr_dbsuspend(3T).

uchar_t ti_db_suspended is non-zero if and only if this thread is currently suspended because the controlling process has called td_thr_dbsuspend on it.

**td_thr_type_e ti_type** is a type of thread. It will be either **TD_THR_USER** for a user thread (one created by the application), or **TD_THR_SYSTEM** for one created by libthread.

int ti_pc is the value of the thread’s program counter, provided that the thread’s ti_state value is **TD_THR_SLEEP**, **TD_THR_STOPPED**, or **TD_THR_STOPPED_ASLEEP**. Otherwise, the value of this field is undefined.

int ti_sp is the value of the thread’s stack pointer, provided that the thread’s ti_state value is **TD_THR_SLEEP**, **TD_THR_STOPPED**, or **TD_THR_STOPPED_ASLEEP**. Otherwise, the value of this field is undefined.
short ti_flags is a set of special flags used by libthread, currently of use only to those debugging libthread.

int ti_pri is the thread’s priority, as it would be returned by thr_getprio(3T).

lwpid_t ti_lid is the ID of the LWP executing this thread, or the ID of the LWP that last executed this thread, if this thread is not currently assigned to a LWP.

sigset_t ti_sigmask is this thread’s signal mask. See thr_sigsetmask(3T).

u_char ti_traceflag is non-zero if and only if event tracing for this thread is on.

uchar_t ti_preemptflag is non-zero if and only if the thread was preempted the last time it was active.

uchar_t ti_pirecflag is non-zero if and only if due to priority inheritance the thread is currently running at a priority other than its regular priority.

td_thr_events_t ti_events is the bitmap of events enabled for this thread.

RETURN VALUES

TD_OK The call completed successfully.
TD_BADTH An invalid thread handle was passed in.
TD_DBERR A call to one of the imported interface routines failed.
TD_ERR The call did not complete successfully.

ATTRIBUTES

See attributes(5) for description of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO

libthread(3T), libthread_db(3T), td_ta_thr_iter(3T), td_thr_dbsuspend(3T),
thr_create(3T), thr_getprio(3T), thr_join(3T), thr_sigsetmask(3T), thr_suspend(3T), lib-
thread(4), libthread_db(4), attributes(5)
td_thr_lockowner – iterate over the set of locks owned by a thread

SYNOPSIS

cc [flag ...] file ... /lib/libthread_db.so.1 [library ...]
#include <proc_service.h>
#include <thread_db.h>

INCLUDED:

- td_err_e td_thr_lockowner(const td_thrhandle_t *th_p,
- td_sync_iter_f *cb, void *cb_data_p);

DESCRIPTION

- td_thr_lockowner() calls the iterator function cb once for every mutex that is held by the thread whose handle is th_p. The synchronization handle and the pointer cb_data_p are passed to the function. See td_ta_thr_iter(3T) for a similarly structured function.
- Iteration terminates early if the callback function cb returns a non-zero value.

RETURN VALUES

- TD_OK The call completed successfully.
- TD_BADTH An invalid thread handle was passed in.
- TD_BADPH There is a NULL external process handle associated with this internal process handle.
- TD_DBERR A call to one of the imported interface routines failed.
- TD_ERR A libthread_db internal error occurred.

ATTRIBUTES

- See attributes(5) for description of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO

- libthread_db(3T), td_ta_thr_iter(3T), libthread_db(4), attributes(5)
NAME

    td_thr_setprio – set the priority of a thread

SYNOPSIS

    cc [ flag ... ] file ... /lib/libthread_db.so.1 [ library ... ]
    #include <proc_service.h>
    #include <thread_db.h>

    td_err_e td_thr_setprio(const td_thrhandle_t *th_p, const int new_prio);

DESCRIPTION

    td_thr_setprio() sets thread th_p's priority to new_prio, just as if a thread within the process had called thr_setprio(). See thr_setprio(3T).

RETURN VALUES

    TD_OK  The call completed successfully.
    TD_BADTH  An invalid thread handle was passed in.
    TD_DBERR  A call to one of the imported interface routines failed.
    TD_ERR  new_prio is an illegal value (out of range).

ATTRIBUTES

    See attributes(5) for description of the following attributes:

    | ATTRIBUTE TYPE | ATTRIBUTE VALUE |
    |----------------|-----------------|
    | MT-Level       | Safe            |

SEE ALSO

    libthread_db(3T), thr_setprio(3T), libthread_db(4), attributes(5)
NAME

`td_thr_setsigpending`, `td_thr_sigsetmask` – manage thread signals for `libthread_db`

SYNOPSIS

```
cc [ flag ... ] file ... /lib/libthread_db.so.1 [ library ... ]
#include <proc_service.h>
#include <thread_db.h>

td_err_e td_thr_setsigpending(const td_thrhandle_t *th_p,
                              const uchar_t ti_sigpending_flag,
                              const sigset_t ti_sigpending);

td_err_e td_thr_sigsetmask(const td_thrhandle_t *th_p,
                            const sigset_t ti_sigmask);
```

DESCRIPTION

The `td_thr_setsigpending()` and `td_thr_setsigmask()` operations affect the signal state of the thread identified by `th_p`.

`td_thr_setsigpending()` sets the set of pending signals for thread `th_p` to `ti_sigpending()`. The value of the `libthread_db`-internal field that indicates whether a thread has any signal pending is set to `ti_sigpending_flag`. To be consistent, `ti_sigpending_flag` should be zero if and only if all of the bits in `ti_sigpending` are zero.

`td_thr_sigsetmask()` sets the signal mask of the thread `th_p` as if the thread had set its own signal mask by way of `thr_sigsetmask(3T)`. The new signal mask is the value of `ti_sigmask`.

There is no equivalent to the `SIG_BLOCK` or `SIG_UNBLOCK` operations of `thr_sigsetmask(3T)`, which mask or unmask specific signals without affecting the mask state of other signals. To block or unblock specific signals, either stop the whole process, or the thread, if necessary, by `td_thr_dbsuspend()`. Then determine the thread’s existing signal mask by calling `td_thr_get_info()` and reading the `ti_sigmask` field of the `td_thrinfo_t` structure returned. Modify it as desired, and set the new signal mask with `td_thr_sigsetmask()`.

RETURN VALUES

`TD_OK` The call completed successfully.

`TD_BADTH` An invalid thread handle was passed in.

`TD_DBERR` A call to one of the imported interface routines failed.

`TD_ERR` A `libthread_db` internal error occurred.

ATTRIBUTES

See `attributes(5)` for description of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO

`libthread_db(3T)`, `td_thr_dbsuspend(3T)`, `td_thr_get_info(3T)`, `libthread_db(4)`, `attributes(5)`
NAME    td_thr_sleepinfo – return the synchronization handle for the object on which a thread is blocked

SYNOPSIS    cc [ flag ... ] file ... /lib/libthread_db.so.1 [ library ... ]
    #include <proc_service.h>
    #include <thread_db.h>
    td_err_t td_thr_sleepinfo(const td_thrhandle_t *th_p, td_synchandle_t *sh_p);

DESCRIPTION    td_thr_sleepinfo() returns in *sh_p the handle of the synchronization object on which a sleeping thread is blocked.

RETURN VALUES    TD_OK    The call completed successfully.
    TD_BADTH    An invalid thread handle was passed in.
    TD_DBERR    A call to one of the imported interface routines failed.
    TD_ERR      The thread th_p is not blocked on a synchronization object, or a libthread_db internal error occurred.

ATTRIBUTES    See attributes(5) for description of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO    libthread_db(3T), libthread_db(4), attributes(5)
NAME

td_thr_tsd – get a thread’s thread-specific data for libthread_db library of interfaces

SYNOPSIS

cc [ flag ... ] file ... /lib/libthread_db.so.1 [ library ... ]
#include <proc_service.h>
#include <thread_db.h>
td_err_e td_thr_tsd(const td_thrhandle_t,
    const thread_key_t key, void **data_pp);

DESCRIPTION

td_thr_tsd() returns in *data_pp the thread-specific data pointer for the thread identified by th_p and the thread-specific data key key. This is the same value that thread th_p would obtain if it called thr_getspecific(3T).

To find all the thread-specific data keys in use in a given target process, call td_ta_tsd_iter(3T).

RETURN VALUES

TD_OK The call completed successfully.
TD_BADTH An invalid thread handle was passed in.
TD_DBERR A call to one of the imported interface routines failed.
TD_ERR A libthread_db internal error occurred.

ATTRIBUTES

See attributes(5) for description of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO

libthread_db(3T), td_ta_tsd_iter(3T), thr_getspecific(3T), libthread_db(4), attributes(5)
NAME        td_thr_validate – test a thread handle for validity

SYNOPSIS    cc [ flag ... ] file ... /lib/libthread_db.so.1 [ library ... ]
#include <proc_service.h>
#include <thread_db.h>
td_err_e td_thr_validate(const td_thrhandle_t *th_p);

DESCRIPTION  td_thr_validate() tests whether th_p is a valid thread handle. A valid thread handle may become invalid if its thread exits.

RETURN VALUES TD_OK        The call completed successfully. th_p is a valid thread handle.
TD_BADTH    th_p was NULL.
TD_DBERR     A call to one of the imported interface routines failed.
TD_NOTHR    th_p is not a valid thread handle.
TD_ERR      A libthread_db internal error occurred.

ATTRIBUTES   See attributes(5) for description of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO     libthread_db(3T), libthread_db(4), attributes(5)
NAME  tell – return a file offset for a file descriptor

SYNOPSIS  #include <unistd.h>
off_t tell(int fd);

DESCRIPTION  The tell() function obtains the current value of the file-position indicator for the file
descriptor fd.

RETURN VALUES  Upon successful completion, tell() returns the current value of the file-position indicator
for fd measured in bytes from the beginning of the file.
Otherwise, it returns −1 and sets errno to indicate the error.

ERRORS  The tell() function will fail if:

EBADF  The file descriptor fd is not an open file descriptor.
EOVERFLOW  The current file offset cannot be represented correctly in an object of
type off_t.
ESPIPE  The file descriptor fd is associated with a pipe or FIFO.

USAGE  The tell() function is equivalent to lseek(fd, 0, SEEK_CUR).

ATTRIBUTES  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  lseek(2), attributes(5)
NAME  telldir – current location of a named directory stream

SYNOPSIS  
#include <dirent.h>
long int telldir(DIR *dirp);

DESCRIPTION  The telldir() function obtains the current location associated with the directory stream specified by dirp.
If the most recent operation on the directory stream was a seekdir(3C), the directory position returned from the telldir() is the same as that supplied as a loc argument for seekdir().

RETURN VALUES  Upon successful completion, telldir() returns the current location of the specified directory stream.

ERRORS  No errors are defined.

ATTRIBUTES  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  opendir(3C), readdir(3C), seekdir(3C), attributes(5)
NAME

termattrs – return the video attributes supported by the terminal

SYNOPSIS

#include <curses.h>
attr_t termattrs(void);

DESCRIPTION

The `termattrs()` function determines which video attributes are supported by the terminal.

RETURN VALUES

The `termattrs()` function returns a logical OR of all video attributes available on the terminal.

ERRORS

None.

SEE ALSO

`attr_get(3XC)`, `attroff(3XC)`
NAME
  termios – general terminal interface

SYNOPSIS
#include <termios.h>

int tcgetattr(int fd, struct termios *termios_p);
int tcsetattr(int fd, int optional_actions, const struct termios *termios_p);
int tcsendbreak(int fd, int duration);
int tcdrain(int fd);
int tcflush(int fd, int queue_selector);
int tcflow(int fd, int action);
speed_t cfgetospeed(const struct termios *termios_p);
int cfsetospeed(struct termios *termios_p, speed_t speed);
speed_t cfgetispeed(const struct termios *termios_p);
int cfsetispeed(struct termios *termios_p, speed_t speed);

#include <sys/types.h>
#include <termios.h>

pid_t tcgetpgrp(int fd);
int tcsetpgrp(int fd, pid_t pgid);

pid_t tcgetsid(int fd);

DESCRIPTION
These functions describe a general terminal interface for controlling asynchronous communications ports. A more detailed overview of the terminal interface can be found in termio(7I), which also describes an ioctl(2) interface that provides the same functionality. However, the function interface described by these functions is the preferred user interface.

Each of these functions is now described on a separate manual page.

SEE ALSO
  ioctl(2), cfgetispeed(3), cfgetospeed(3), cfsetispeed(3), cfsetospeed(3), tcdrain(3),
tcflow(3), tcflush(3), tcgetattr(3), tcgetpgrp(3), tcgetsid(3), tcsendbreak(3), tcsetattr(3),
tcgetpgrp(3), tcsendbreak(3), termio(7I)
NAME  
termname – return the value of the environmental variable TERM

SYNOPSIS
#include <curses.h>
char *termname(void);

DESCRIPTION
The termname() function returns a pointer to the value of the environmental variable TERM (truncated to 14 characters).

RETURN VALUES
The termname() returns a pointer to the terminal’s name.

ERRORS
None.

SEE ALSO
del_curterm(3XC)
NAME  t_error – produce error message

SYNOPSIS  cc [ flag ... ] file ... -lnsl [ library ... ]
#include <xti.h>
int t_error(const char *errmsg);

DESCRIPTION  This routine is part of the XTI interfaces which evolved from the TLI interfaces. XTI represents the future evolution of these interfaces. However, TLI interfaces are supported for compatibility. When using a TLI routine that has the same name as an XTI routine, a different header file, tiuser.h, must be used. Refer to the section, TLI COMPATIBILITY, for a description of differences between the two interfaces.

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 errmsg is a user-supplied error message that gives context to the error. The error message is written as follows: first (if errmsg is not a null pointer and the character pointed to by errmsg is not the null character) the string pointed to by errmsg followed by a colon and a space; then a standard error message string for the current error defined in t_errno. If t_errno has a value different from T_SYSERR, the standard error message string is followed by a newline character. If, however, t_errno is equal to T_SYSERR, the t_errno string is followed by the standard error message string for the current error defined in errno followed by a newline.

The language for error message strings written by t_error() is that of the current locale. If it is in English, the error message string describing the value in t_errno is identical to the comments following the t_errno codes defined in xti.h. The contents of the error message strings describing the value in errno are the same as those returned by the strerror() function with an argument of errno.

The error number, t_errno, is only set when an error occurs and it is not cleared on successful calls.

VALID STATES  Legitimate states (see t_getstate(3N)) for a call to this routine are every one except T_UNINIT.

RETURN VALUES  Upon completion, a value of 0 is returned.

ERRORS  No errors are defined for the t_error() function.

EXAMPLES  If a t_connect(3N) function fails on transport endpoint fd2 because a bad address was given, the following call might follow the failure:

       t_error(“t_connect failed on fd2”);

The diagnostic message would be produced:
t_error (3N)

Network Functions

**t_connect failed on fd2: incorrect addr format**

where,

- **t_connect failed** identifies the function that failed
- **on fd2:** identifies the transport endpoint on which the failure occurred
- **incorrect addr format** identifies the specific error that occurred

**TLI COMPATIBILITY**

The XTI and TLI interface definitions have common names but use different header files. This, and other semantic differences between the two interfaces are described in the subsections below.

**Interface Header**

The XTI interfaces use the header file, `xti.h`. TLI interfaces should not use this header. They should use the header:

```c
#include <tiuser.h>
```

**Error Description Values**

The `t_errno` value that can be set by the XTI interface and cannot be set by the TLI interface is:

- **TPROTO**

For more information refer to the Transport Interfaces Programming Guide.

**ATTRIBUTES**

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**SEE ALSO**

intro(2), t_connect(3N), t_getstate(3N), attributes(5)

Transport Interfaces Programming Guide
NAME  
t_free – free a library structure

SYNOPSIS  
cc [ flag ...] file ... -lnsl [ library ... ]

#include <xti.h>

int t_free(void *ptr, int struct_type);

DESCRIPTION  
This routine is part of the XTI interfaces which evolved from the TLI interfaces. XTI represents the future evolution of these interfaces. However, TLI interfaces are supported for compatibility. When using a TLI routine that has the same name as an XTI routine, a different header file, tiuser.h, must be used. Refer to the section, TLI COMPATIBILITY, for a description of differences between the two interfaces.

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.

ptr points to one of the seven structure types described for t_alloc(), and struct_type identifies the type of that structure, which must be one of the following:

<table>
<thead>
<tr>
<th>T_BIND</th>
<th>struct t_bind</th>
</tr>
</thead>
<tbody>
<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>
</tbody>
</table>

where each of these structures is used as an argument to one or more transport functions. (buf and other members of the netbuf structure are shown in t_connect(3N).) 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 structure. If buf is a null pointer, t_free() will not attempt to free memory. After all buffers are freed, t_free() will free the memory associated with the structure pointed to by ptr.

Undefined results will occur if ptr or any of the buf pointers points to a block of memory that was not previously allocated by t_alloc().

VALID STATES  
Legitimate states (see t_getstate(3N)) for a call to this routine are every one except T_UNINIT.

RETURN VALUES  
t_free() returns:

0      On success.

−1     On failure.

On failure, t_errno is set to indicate the error, and possibly errno is set.

modified 10 Feb 1997
SunOS 5.6
3N-1599
On failure, t_errno is set to the following:

- **TNOSTRUCTYPE**: Unsupported struct_type requested.
- **TPROTO**: This error indicates that a communication problem has been detected between XTI and the transport provider for which there is no other suitable XTI t_errno value.
- **TSYSERR**: A system error has occurred during execution of this function, errno will be set to the specific error.

**TLI COMPATIBILITY**

The XTI and TLI interface definitions have common names but use different header files. This, and other semantic differences between the two interfaces are described in the subsections below.

**Interface Header**

The XTI interfaces use the header file, xti.h. TLI interfaces should not use this header. They should use the header:

```c
#include <tiuser.h>
```

**Error Description Values**

The t_errno value that can be set by the XTI interface and cannot be set by the TLI interface is:

**TPROTO**

For more information refer to the *Transport Interfaces Programming Guide*.

**ATTRIBUTES**

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**SEE ALSO**

- t_connect(3N), t_alloc(3N), t_getstate(3N), attributes(5)
- *Transport Interfaces Programming Guide*
NAME
tgetent, tgetflag, tgetnum, tgetstr, tgoto – emulate the termcap database

SYNOPSIS
#include <term.h>
int tgetent (char *bp, const char *name);
int tgetflag (char id[2]);
int tgetnum (char id[2]);
char *tgetstr (char cap[2], char **area);
char *tgoto (const char *cap, int col, int row);

ARGUMENTS
bp Is a pointer to a buffer. This parameter is ignored.
name Is the termcap entry to look up.
cap Is the pointer to a termcap capability.
area Is a pointer to the area where tgetstr() stores the decoded string.
col Is the column placement of the new cursor.
row Is the row placement of the new cursor.

DESCRIPTION
These functions provide an interface to the termcap database.
The tgetent() function looks up the termcap entry for the terminal name. The bp parameter is ignored by this function.
The tgetflag() function returns the Boolean value of the termcap capability pointed to by cap.
The tgetnum() function looks up the numeric entry for cap.
The tgetstr() function looks up the string entry for the termcap capability pointed to by cap, placing the decoded string at area and advancing the area pointers. The puts(3XC) function should be used to output the string.
The tgoto() function decodes cursor values returned from tgetstr(). A pointer to a cursor addressing string is returned that, when sent to the terminal with puts(), moves the cursor to the new location.

These functions are included for compatibility purposes with programs that use the termcap library. New programs should use terminfo functions described on the tigetflag(3XC) man page.

RETURN VALUES
On success, those functions that return integers return OK. Otherwise, they return ERR. Those functions that return pointers return a null pointer when an error occurs.

ERRORS
None.

SEE ALSO
putp(3XC), setupterm(3XC), tigetflag(3XC)
NAME  
t_getinfo – get protocol-specific service information

SYNOPSIS  
c
cc  [ flag ... ]  file ...  −l  xsl  [ library ... ]
#include <xti.h>
int t_getinfo(int fd, struct t_info *info);

DESCRIPTION  
This routine is part of the XTI interfaces which evolved from the TLI interfaces. XTI
represents the future evolution of these interfaces. However, TLI interfaces are supported
for compatibility. When using a TLI routine that has the same name as an XTI routine, a
different header file, tiuser.h, must be used. Refer to the section, TLI COMPATIBILITY,
for a description of differences between the two interfaces.

This function returns the current characteristics of the underlying transport protocol or
transport connection associated with file descriptor fd. The info pointer is used to return
the same information returned by t_open(3N), although not necessarily precisely the
same values. This function enables a transport user to access this information during any
phase of communication.

The info argument points to a t_info structure, which contains the following members:

- long addr; /* max size in octets of the transport protocol address */
- long options; /* max number of bytes of protocol-specific options */
- long tsdu; /* max size in octets of transport service data unit */
- long etsdu; /* max size in octets of expedited transport service */
- data unit (ETSDU) */
- long connect; /* max number of octets allowed on connection */
- /* establishment functions */
- long discon; /* max number of octets of data allowed on t_snddis() and
t_rcvdis() functions */
- long servtype; /* service type supported by the transport provider */
- long flags; /* other info about the transport provider */

The values of the fields have the following meanings:

addr  
A value greater than zero (>T_NULL) indicates the maximum octet size of
a transport protocol address; a value of −2 (T_INVALID) specifies that the
transport provider does not provide user access to transport protocol
addresses.

options  
A value greater than zero (>T_NULL) indicates the maximum number of
bytes of protocol-specific options supported by the provider; a value of
−2 (T_INVALID) specifies that the transport provider does not support
user-settable options.

tsdu  
A value greater than zero (>T_NULL) specifies the maximum octet size of
a transport service data unit (TSDU); a value of zero (T_NULL) specifies
that the transport provider does not support the concept of TSDU,
although it does support the sending of a data stream across a connection with no logical boundaries preserved for the connection; a tsdu value of \(-1\) (T_INFINITE) specifies that there is no limit to the size of a TSDU; a value of \(-2\) (T_INVALID) specifies that the transfer of normal data is not supported by the transport provider.

etsdu A value greater than zero (>T_NULL) specifies the maximum amount of octets for an expedited transport service data unit (ETSDU); an etsdu value of zero (T_NULL) specifies that the transport provider does not support the concept of ETSDU, although it does support the sending of an expedited data stream across a connection with no logical boundaries preserved for the connection; an etsdu value of \(-1\) (T_INFINITE) specifies that there is no limit to the size of a ETSDU; an etsdu value of \(-2\) (T_INVALID) specifies that the transfer of expedited data is not supported by the transport provider. Note that the semantics of expedited data may be quite different for different transport providers.

connect A value greater than zero (>T_NULL) specifies the maximum amount of octets that may be associated with connection establishment functions; a connect field value of \(-2\) (T_INVALID) specifies that the transport provider does not allow data to be sent with connection establishment functions.

discon A discon field value greater than zero (>T_NULL) specifies the maximum amount of octets that may be associated with the t_snddis() and t_rcvdis() functions; a discon field value of \(-2\) (T_INVALID) 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.

flags This is a bit field used to specify other information about the communications provider. If the T_SENDZERO bit is set in flags, this indicates that the underlying transport provider supports the sending of zero-length TSDUs.

If a transport user is concerned with protocol independence, the above sizes may be accessed to determine how large the buffers must be to hold each piece of information. Alternatively, the t_alloc(3N) function may be used to allocate these buffers. An error will result if a transport user exceeds the allowed data size on any function.

Because the value of each field may change as a result of option negotiation, a user may need to call t_getinfo() to retrieve the current characteristics.

The value of each field may change as a result of protocol option negotiation during connection establishment (the t_optmgt(3N) call has no effect on the values returned by t_getinfo()). These values will only change from the values presented to t_open(3N) after the endpoint enters the T_DATAXFER state.
The `servtype` field of `info` specifies one of the following values on return:

- **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`.

### VALID STATES
Legitimate states (see `t_getstate(3N)`) for a call to this routine are every one except T_UNINIT.

### RETURN VALUES
`t_getinfo()` returns:

- **0**: On success.
- **−1**: On failure.

On failure, `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:

- **T_BADF**: The specified file descriptor does not refer to a transport endpoint.
- **T_PROTO**: This error indicates that a communication problem has been detected between XTI and the transport provider for which there is no other suitable XTI `t_errno`.
- **T_SYSERR**: A system error has occurred during execution of this function, `errno` will be set to the specific error.

### TLI COMPATIBILITY
The XTI and TLI interface definitions have common names but use different header files. This, and other semantic differences between the two interfaces are described in the subsections below.

### Interface Header
The XTI interfaces use the header file, `xti.h`. TLI interfaces should not use this header. They should use the header:

```c
#include <tiuser.h>
```

### Error Description Values
The `t_errno` value that can be set by the XTI interface and cannot be set by the TLI interface is:

- **T_PROTO**

### Notes
For TLI, the `t_info` structure referenced by `info` lacks the following structure member:

```c
long flags;
/* other info about the transport provider */
```

This member was added to `struct t_info` in the XTI interfaces.

3N-1604

SunOS 5.6

modified 10 Feb 1997
When a value of −1 is observed as the return value in various t_info structure members, it signifies that the transport provider can handle an infinite length buffer for a corresponding attribute, such as address data, option data, TSDU (octet size), ETSDU (octet size), connection data, and disconnection data. The corresponding structure members are addr, options, tsdu, estdu, connect, and discon, respectively.

For more information refer to the Transport Interfaces Programming Guide.

**ATTRIBUTES**

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**SEE ALSO**

t_alloc(3N), t_getstate(3N), t_open(3N), t_optmgmt(3N), t_rcvdis(3N), t_snddis(3N), attributes(5)

Transport Interfaces Programming Guide

modified 10 Feb 1997

SunOS 5.6

3N-1605
NAME
t_getprotaddr – get the protocol addresses

SYNOPSIS
cc [ flag ... ] file ... -lnsl [ library ... ]
#include <xti.h>
int t_getprotaddr(int fd, struct t_bind *boundaddr, struct t_bind *peeraddr);

DESCRIPTION
t_getprotaddr() returns local and remote protocol addresses currently associated with
the transport endpoint specified by fd.
(maxlen and other members of netbuf are described in t_connect(3N).) In boundaddr and
peeraddr, the user specifies maxlen, which is the maximum size (in bytes) of the address
buffer, and buf which points to the buffer where the address is to be placed.
On return, the buf field of boundaddr points to the address, if any, currently bound to fd, and
the len field specifies the length of the address. If the transport endpoint is in the
T_UNBND state, zero is returned in the len field of boundaddr. The buf field of peeraddr
points to the address, if any, currently connected to fd, and the len field specifies the
length of the address. If the transport endpoint is not in the T_DATAXFER state, zero is
returned in the len field of peeraddr. If the maxlen field of boundaddr or peeraddr was set
to zero, no address is returned.

VALID STATES
Legitimate states (see t_getstate(3N)) for a call to this routine are every one except
T_UNINIT.

RETURN VALUES
t_getprotaddr() returns:
0 On success.
−1 On failure.
On failure, t_errno is set to indicate the error, and possibly errno is set.

ERRORS
On failure, t_errno is set to one of the following:
TBADF The specified file descriptor does not refer to a transport endpoint.
TBUFOVFLW The number of bytes allocated for an incoming argument (maxlen)
is greater than 0 but not sufficient to store the value of that argu-
ment.
TSYSERR A system error has occurred during execution of this function.
TPROTO This error indicates that a communication problem has been
detected between XTI and the transport provider for which there is
no other suitable XTI t_errno value.
In the TLI interface definition, no counterpart of this routine was defined.

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO `t_bind(3N), t_connect(3N), t_getstate(3N), attributes(5)`
NAME t_getstate – get the current state

SYNOPSIS cc [ flag ... ] file ... -lnsl [ library ... ]
#include <xti.h>
int t_getstate(int fd);

DESCRIPTION This routine is part of the XTI interfaces which evolved from the TLI interfaces. XTI represents the future evolution of these interfaces. However, TLI interfaces are supported for compatibility. When using a TLI routine that has the same name as an XTI routine, a different header file, tiuser.h, must be used. Refer to the section, TLI COMPATIBILITY, for a description of differences between the two interfaces.
The t_getstate() function returns the current state of the provider associated with the transport endpoint specified by fd.

VALID STATES Legitimate states (see t_getstate(3N)) for a call to this routine are every one except T_UNINIT.

RETURN VALUES t_getstate() returns:
The Current State On success.
−1 On failure.
On failure, t_errno is set to indicate the error, and possibly errno is set. The current state may be one of the following:
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 to send an orderly release request)
If the provider is undergoing a state transition when t_getstate() is called, the function will fail.

ERRORS On failure, t_errno is set to one of the following:
TBADF The specified file descriptor does not refer to a transport endpoint.
TPROTO This error indicates that a communication problem has been detected between XTI and the transport provider for which there is no other suitable XTI t_errno value.
TSTATECHNG The transport provider is undergoing a transient state change.
Network Functions

**TLI COMPATIBILITY**

The XTI and TLI interface definitions have common names but use different header files. This, and other semantic differences between the two interfaces are described in the subsections below.

**Interface Header**

The XTI interfaces use the header file, `xti.h`. TLI interfaces should *not* use this header. They should use the header:

```c
#include <tiuser.h>
```

**Error Description Values**

The `t_errno` value that can be set by the XTI interface and cannot be set by the TLI interface is:

TPROTO

For more information refer to the *Transport Interfaces Programming Guide*.

**ATTRIBUTES**

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**SEE ALSO**

`t_open(3N), attributes(5)`

*Transport Interfaces Programming Guide*

---

modified 10 Feb 1997

SunOS 5.6

3N-1609
### NAME
threads, pthreads, libpthread, libthread – thread libraries: libpthread and libthread

### SYNOPSIS

**POSIX**
```bash
c [ flag ... ] file ... -lpthread [ -lposix4 library ... ]
#include <pthread.h>
```

**Solaris**
```bash
c [ flag ... ] file ... -lthread [ library ... ]
#include <thread.h>
#include <sched.h>
```

### DESCRIPTION

Two threads libraries are available, POSIX and Solaris. Both implementations are interoperable, their functionality similar, and can be used within the same application. However, only POSIX threads are guaranteed to be fully portable to other POSIX-compliant environments. As indicated by the "Synopsis" section above, their use requires different source include files and different linking libraries.

### Similarities
Most of the functions in both libraries, **libpthread** and **libthread**, have a counterpart in the other's library. POSIX functions and Solaris functions, whose names have similar endings, usually have similar functionality, number of arguments, and use of arguments. i.e.:

<table>
<thead>
<tr>
<th>POSIX</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td>pthread_kill( )</td>
<td>thr_kill( )</td>
</tr>
<tr>
<td>pthread_sigmask( )</td>
<td>thr_sigsetmask( )</td>
</tr>
<tr>
<td>pthread_mutex_lock( )</td>
<td>mutex_lock()</td>
</tr>
<tr>
<td>sem_wait()</td>
<td>sema_wait()</td>
</tr>
</tbody>
</table>

All POSIX threads function names begin with the prefix "pthread", with semaphore names being the exception.

### Differences

#### POSIX
- is more portable,
- establishes characteristics for each thread according to configurable attribute objects,
- implements thread cancellation,
- enforces scheduling algorithms, and
- allows for clean-up handlers for `fork(2)` calls.

#### Solaris
- threads can be suspended and continued,
- implements an optimized mutex, reader/writer locking.
- may increase the concurrency, and
- implements daemon threads, for whose demise the process does not wait.
<table>
<thead>
<tr>
<th>IMPLEMENTATION</th>
<th>POSIX</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creation</td>
<td>pthread_create()</td>
<td>thr_create()</td>
</tr>
<tr>
<td></td>
<td>pthread_attr_init()</td>
<td>---</td>
</tr>
<tr>
<td></td>
<td>pthread_attr_setdetachstate()</td>
<td>---</td>
</tr>
<tr>
<td></td>
<td>pthread_attr_getdetachstate()</td>
<td>---</td>
</tr>
<tr>
<td></td>
<td>pthread_attr_setinheritsched()</td>
<td>---</td>
</tr>
<tr>
<td></td>
<td>pthread_attr_getinheritsched()</td>
<td>---</td>
</tr>
<tr>
<td></td>
<td>pthread_attr_setschedparam()</td>
<td>---</td>
</tr>
<tr>
<td></td>
<td>pthread_attr_getschedparam()</td>
<td>---</td>
</tr>
<tr>
<td></td>
<td>pthread_attr_setschedpolicy()</td>
<td>---</td>
</tr>
<tr>
<td></td>
<td>pthread_attr_getschedpolicy()</td>
<td>---</td>
</tr>
<tr>
<td></td>
<td>pthread_attr_setscope()</td>
<td>---</td>
</tr>
<tr>
<td></td>
<td>pthread_attr_getscope()</td>
<td>---</td>
</tr>
<tr>
<td></td>
<td>pthread_attr_setstackaddr()</td>
<td>---</td>
</tr>
<tr>
<td></td>
<td>pthread_attr_getstackaddr()</td>
<td>---</td>
</tr>
<tr>
<td></td>
<td>pthread_attr_setstacksize()</td>
<td>---</td>
</tr>
<tr>
<td></td>
<td>pthread_attr_getstacksize()</td>
<td>---</td>
</tr>
<tr>
<td></td>
<td>pthread_attr_destroy()</td>
<td>---</td>
</tr>
<tr>
<td></td>
<td>thr_min_stack()</td>
<td>---</td>
</tr>
<tr>
<td>Exit</td>
<td>pthread_exit()</td>
<td>thr_exit()</td>
</tr>
<tr>
<td></td>
<td>pthread_join()</td>
<td>thr_join()</td>
</tr>
<tr>
<td></td>
<td>pthread_detach()</td>
<td>---</td>
</tr>
<tr>
<td>Thread Specific Data</td>
<td>pthread_key_create()</td>
<td>thr_keycreate()</td>
</tr>
<tr>
<td></td>
<td>pthread_setspecific()</td>
<td>thr_setspecific()</td>
</tr>
<tr>
<td></td>
<td>pthread_getspecific()</td>
<td>thr_getspecific()</td>
</tr>
<tr>
<td></td>
<td>pthread_key_delete()</td>
<td>---</td>
</tr>
<tr>
<td>Signal</td>
<td>pthread_sigmask()</td>
<td>thr_sigsetmask()</td>
</tr>
<tr>
<td></td>
<td>pthread_kill()</td>
<td>thr_kill()</td>
</tr>
<tr>
<td>ID</td>
<td>pthread_self()</td>
<td>thr_self()</td>
</tr>
<tr>
<td></td>
<td>pthread_equal()</td>
<td>---</td>
</tr>
<tr>
<td></td>
<td>thr_main()</td>
<td>---</td>
</tr>
<tr>
<td>Scheduling</td>
<td>---</td>
<td>thr_yield()</td>
</tr>
<tr>
<td></td>
<td>---</td>
<td>thr_suspend()</td>
</tr>
<tr>
<td></td>
<td>---</td>
<td>thr_continue()</td>
</tr>
<tr>
<td></td>
<td>---</td>
<td>thr_setconcurrency()</td>
</tr>
<tr>
<td></td>
<td>---</td>
<td>thr_getconcurrency()</td>
</tr>
<tr>
<td></td>
<td>pthread_setschedparam()</td>
<td>thr_setprio()</td>
</tr>
<tr>
<td></td>
<td>pthread_getschedparam()</td>
<td>thr_getprio()</td>
</tr>
</tbody>
</table>

modified 8 May 1997

SunOS 5.6
Cancellation
- pthread_cancel()
- pthread_setcancelstate()
- pthread_setcanceltype()
- pthread_testcancel()
- pthread_cleanup_pop()
- pthread_cleanup_push()

Mutex
- pthread_mutex_init()
- pthread_mutexattr_init()
- pthread_mutexattr_setpshared()
- pthread_mutexattr_getpshared()
- pthread_mutexattr_setprotocol()
- pthread_mutexattr_getprotocol()
- pthread_mutexattr_setprioceiling()
- pthread_mutexattr_getprioceiling()
- pthread_mutexattr_destroy()
- pthread_mutex_lock()
- pthread_mutex_trylock()
- pthread_mutex_unlock()
- pthread_mutex_destroy()

Condition Variable
- pthread_cond_init()
- pthread_condattr_init()
- pthread_condattr_setpshared()
- pthread_condattr_getpshared()
- pthread_condattr_destroy()
- pthread_cond_wait()
- pthread_cond_timedwait()
- pthread_cond_signal()
- pthread_cond_broadcast()
- pthread_cond_destroy()

Reader/Writer
- rwlock_init()
- rw_rdlock()
- rw_trrdlock()
- rw_wrlock()
- rw_tryrwlock()
- rw_unlock()
- rwlock_destroy()

Semaphore
- sem_init()
- sem_open()
- sem_close()
- sem_wait()

SunOS 5.6  modified 8 May 1997
Multi-threaded behavior is asynchronous, and therefore, optimized for concurrent and parallel processing. Since threads, always from within the same process and sometimes from multiple processes, share global data with each other, they are not guaranteed exclusive access to the shared data at any point in time. Securing mutually exclusive access to shared data requires synchronization among the threads. Solaris implements four synchronization mechanisms:

- **mutex**
- **condition variable**
- **reader/writer locking** (optimized frequent-read occasional-write mutex)
- **semaphore**

POSIX implements all but reader/writer locking.

Synchronizing multiple threads diminishes their concurrency. The coarser the grain of synchronization, that is, the larger the block of code that is locked, the lesser the concurrency.

**MT fork()** If a multi-threaded program calls `fork(2)`, it implicitly calls `fork1(2)`, which replicates only the calling thread. Should there be any outstanding mutexes throughout the process, the application should call `pthread_atfork(3T)`, to wait for and acquire those mutexes, prior to calling `fork()`.
ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe, Fork1-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO fork(2), pthread_atfork(3T), pthread_create(3T), attributes(5), standards(5)

ERRORS

In a multi-threaded application, linked with libpthread or libthread, EINTR may be returned whenever another thread calls fork(2), which calls fork1(2) instead.
NAME    thr_main – identify the main thread

SYNOPSIS  cc [ flag ...] file ... -lthread [ library ...]
           #include <thread.h>
           int thr_main(void);

DESCRIPTION  thr_main() returns:
              1 if the calling thread is the main thread.
              0 if the calling thread is not the main thread.
              −1 if libthread is not linked in or thread initialization has not completed.

FILES  /lib/libthread

ATTRIBUTES  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  thr_self(3T), attributes(5)
NAME  thr_min_stack – returns the minimum-allowable size for a thread’s stack

SYNOPSIS

cc [ flag ... ] file ... -lthread [ library ... ]

#include <thread.h>

size_t thr_min_stack(void);

DESCRIPTION

When a thread is created with a user-supplied stack, the user must reserve enough space to run this thread. In a dynamically linked execution environment, it is very hard to know what the minimum stack requirements are for a thread. The function thr_min_stack() returns the amount of space needed to execute a null thread. This is a thread that was created to execute a null procedure. A thread that does something useful should have a stack size that is thr_min_stack() + <some increment>.

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.

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.

thr_min_stack() will return the unsigned int THR_MIN_STACK, which is the minimum-allowable size for a thread’s stack.

In this implementation the default size for a user-thread’s stack is one mega-byte. If the second argument to thr_create(3T) is NULL, then the default stack size for the newly-created thread will be used. Otherwise, you may specify a stack-size that is at least THR_MIN_STACK, yet less than the size of your machine’s virtual memory.

It is recommended that the default stack size be used.

To determine the smallest-allowable size for a thread’s stack, execute the following:

```
/* cc thisfile.c -lthread */
#define _REENTRANT
#include <thread.h>
#include <stdio.h>

main() {
    printf("thr_min_stack() returns %u\n",thr_min_stack());
}
```

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

3T-1616 SunOS 5.6 modified 8 May 1997
The POSIX threads implementation, pthreads, does not have a corresponding function to `thr_min_stack()`, but it does implement a minimum stack size, whose value is `PTHREAD_STACK_MIN`, which may be ascertained as follows:

```c
#include <stdio.h>
#include <pthread.h>

#define _REENTRANT

int main() {
    printf("minimum POSIX stack size is %u\n", PTHREAD_STACK_MIN);
}
```

Although the POSIX threads implementation, pthreads, does not have a corresponding function to `thr_min_stack()`, it does implement a minimum stack size, whose value is `PTHREAD_STACK_MIN`, which may be ascertained as follows:

```c
#include <stdio.h>
#include <pthread.h>

#define _REENTRANT

int main() {
    printf("minimum POSIX stack size is %u\n", PTHREAD_STACK_MIN);
}
```
NAME  

thr_setconcurrency, thr_getconcurrency – get/set thread concurrency level

SYNOPSIS  

#include <thread.h>

int thr_setconcurrency(int new_level);
int thr_getconcurrency(void);

DESCRIPTION  

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 new_level, 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)).

If new_level 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.

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.

RETURN VALUES  

thr_setconcurrency() returns zero when successful. A non-zero value indicates an error code.

thr_getconcurrency() always returns the current value for the desired concurrency level.

ERRORS  

If any of the following conditions are detected, thr_setconcurrency() fails and returns the corresponding value:

EAGAIN    the specified concurrency level would cause a system resource to be exceeded.
EINVAL    new_level is negative.

ATTRIBUTES  

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  

thr_create(3T), attributes(5), standards(5)

NOTES  

The Solaris threads set/get concurrency functionality described on this man page is not implemented in the POSIX threads interface.
NAME

thr_stksegment – get thread stack bottom and stack size

SYNOPSIS

cc [ flag ... ] file ... -lthread [ library ... ]
#include <thread.h>
#include <sys/signal.h>

int thr_stksegment(stack_t *);

DESCRIPTION

The stack information provided by thr_stksegment() is typically used by debuggers, garbage collectors, and similar applications. Most applications should not require such information. The bottom of the thread stack returned by thr_stksegment() points to a part of the stack which may contain data maintained by libthread. The user’s thread stack starts at a point below the bottom of the stack as returned by thr_stksegment().

RETURN VALUES

thr_stksegment() returns 0 if both the thread stack bottom and stack size were successfully retrieved; otherwise, it returns a non-zero error code.

ERRORS

If any of the following conditions are detected, thr_stksegment() fails and returns the corresponding value:

EFAULT A system call used to get the stack information failed because a bad address was passed to it.
EAGAIN The stack information for the thread is not available because the thread’s initialization is not yet complete, or the thread is an internal thread.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO

thr_create(3T), attributes(5)
NAME  thr_suspend, thr_continue – suspend or continue thread execution

SYNOPSIS  
#include <thread.h>

int thr_suspend(thread_t target_thread);

int thr_continue(thread_t target_thread);

DESCRIPTION  
The thr_suspend() function immediately suspends the execution of the thread specified by target_thread. 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.

The thr_continue() function resumes the execution of a suspended thread. Once a suspended thread is continued, subsequent calls to thr_continue() have no effect.

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().

RETURN VALUES  
The thr_suspend() and thr_continue() functions return 0 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  target_thread cannot be found in the current process.

ECANCELED  target_thread was not suspended because a subsequent thr_continue() occurred before the suspend completed.

EINVAL  When thr_continue() returns EINVAL, target_thread has died and thr_join() must be called on it to reclaim its resources.

If the following condition is detected, thr_suspend() fails and returns the corresponding value:

EDEADLK  Suspending target_thread will cause all threads in the process to be suspended.

ATTRIBUTES  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  
thr_create(3T), thr_join(3T), attributes(5), standards(5)

NOTES  
The are no POSIX counterparts to the Solaris threads suspend and continue functionality described on this man page.
NAMEthr_yield – thread yield to another thread

SYNOPSIScc [ flag ...] file ... -lthread [ library ...]
#include <thread.h>
void thr_yield(void);

DESCRIPTIONthr_yield() causes the current thread to yield its execution in favor of another thread with the same or greater priority.

RETURN VALUETHR_yield() returns nothing and does not set errno.

ATTRIBUTESSee attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSOsched_yield(3R), thr_setprio(3T), attributes(5), standards(5)

NOTESThere is a POSIX real-time function, sched_yield(3R), that provides the same functionality as thr_yield(). For Solaris, sched_yield(3R) does nothing more than return an error message indicating that the system call is not supported.
NAME  

tigetflag, tigetnum, tigetstr, tparm — return the value of a terminfo capability

SYNOPSIS  

#include <term.h>

int tigetflag (char *capname);
int tigetnum (char *capname);
char *tigetstr (char *capname);
char *tparm (char *cap, long p1, long p2, long p3,
       long p4, long p5, long p6, long p7,
       long p8, long p9);

ARGUMENTS  

capname  Is the name of the terminfo capability for which the value is required.
cap  Is a pointer to a string capability.
p1...p9  Are the parameters to be instantiated.

DESCRIPTION  

The tigetflag(), tigetnum(), and tigetstr() functions return values for terminfo capabilities passed to them.
The following null-terminated arrays contain the capnames, the termcap codes and full C names for each of the terminfo variables.

char *boolnames, *boolcodes, *boolfnames
char *numnames, *numcodes, *numfnames
char *strnames, *strcodes, *strfnames

The tparm() function instantiates a parameterized string using nine arguments. The string is suitable for output processing by tputs().

RETURN VALUES  

On success, the tigetflag(), tigetnum(), and tigetstr() functions return the specified terminfo capability.
tigetflag() returns −1 if capname is not a Boolean capability.
tigetnum() returns −2 if capname is not a numeric capability.
tigetstr() returns (char *)−1 if capname is not a string capability.
On success, the tparm() function returns cap in a static buffer with the parameterization resolved. Otherwise, it returns a null pointer.

ERRORS  

None.

SEE ALSO  

tgetent(3XC), terminfo(4)
NAME  
timer_create – create a timer

SYNOPSIS  
cc [ flag ... ] file ... -lpthread [ 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 sigval sigev_value; /* signal value */
};
union sigval {
  int sival_int; /* integer value */
  void *sival_ptr; /* pointer value */
};

DESCRIPTION  
timer_create() creates a timer 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().
The timer may be created per-LWP or per-process. Expiration signals for a per-LWP timer
will be sent to the creating LWP. Expiration signals for a per-process timer will be sent to
the process. A per-LWP timer will be automatically deleted when the creating LWP exits.
By default, timers are created per-LWP. If the symbol
_POSIX_PER_PROCESS_TIMER_SOURCE is defined or the symbol _POSIX_C_SOURCE is
defined to have a value greater than 199500L before the inclusion of <time.h>, timers will
be created per-process.
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 con-
tains the signal number and the application specific data value to be sent to the pro-
cess. 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 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.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe with exceptions</td>
</tr>
</tbody>
</table>

SEE ALSO

exec(2), fork(2), time(2), clock_settime(3R), signal(3C), timer_delete(3R), timer_settime(3R), attributes(5)

NOTES

Timers are not inherited by a child process across a fork(2) and can be disarmed and deleted by an exec(2).

Due to the way that signals are handled, if two timers expire at approximately the same time, the signal handler might not detect both of them.

In a future release, the ability to create per-LWP timers will be removed, and all calls to timer_create() will result in per-process timers.
NAME       timer_delete – delete a per-LWP timer

SYNOPSIS  cc [ flag ... ] file ... -lposix4 [ library ... ]
           #include <time.h>
           int timer_delete(timer_t timerid);

DESCRIPTION timer_delete() deletes the specified timer, timerid, 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    timerid does not refer to a valid timer.
            ENOSYS    timer_delete() is not supported by this implementation.

ATTRIBUTES See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe with exceptions</td>
</tr>
</tbody>
</table>

SEE ALSO    timer_create(3R), attributes(5)
NAME  
timer_settime, timer_gettime, timer_getoverrun – high-resolution timer operations

SYNOPSIS  
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 */
};

DESCRIPTION  
If value->it_value is non-zero, timer_settime() arms the timer, timerid, to next expire after
the time designated by value->it_value. Upon expiration, an application-specified
notification (see timer_create(3R)) or the default signal, SIGALRM, is queued for the cal-
calling LWP. If timerid was already armed when timer_settime() is called, this call resets
the time until the next expiration to the value of value->it_value. If
value->it_value is zero, then the timer is disarmed.

value->it_value may be expressed as either an absolute or relative time. If flags is set to
TIMER_RELTIME, then the timer will initially expire relative to when the call is made.
If flags 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.

If value->it_interval is non-zero, then timerid, will be a “periodic” timer, to be reloaded
to expire every value->it_interval seconds (nanoseconds). Otherwise, if
value->it_interval is zero and value->it_value is non-zero, then timerid is a “one-shot”
timer, which will expire only at the time specified by value->it_value.

If ovalue is not NULL, and timer timerid had previously been used, then timer_settime() will
store the remaining time until the previous timer expires in ovalue->it_value, and the
previous reload interval in ovalue->it_interval. (If the previous timer was disarmed,
ovalue->it_value will be set to zero). The values stored in ovalue by timer_settime() are
the same values that would have been returned by a call to timer_gettime( timerid, ...).

timer_gettime() stores the amount of time until the specified timer, timerid, expires into
value->it_value, and the timer’s reload value into value->it_interval.
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.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Async-Signal-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO

clock_settime(3R), timer_create(3R), timer_delete(3R), attributes(5)
NAME

times – get process times

SYNOPSIS

/usr/ucb/cc [flag ...] file ...
#include <sys/param.h>
#include <sys/types.h>
#include <sys/times.h>

int times(tmsp)
    register struct tms *tmsp;

DESCRIPTION

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>.

A structure with the following members is returned by times():

    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 */

The children’s times are the sum of the children’s process times and their children’s times.

RETURN VALUES

times() returns

0     on success.

-1    on failure.

SEE ALSO

time(1), time(2), wait(2), getrusage(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-threaded applications is unsupported.

times() has been superseded by getrusage(3C).
NAME  t_listen – listen for a connection indication

SYNOPSIS  cc [ flag ... ] file ... -lnsl [ library ... ]
#include <xti.h>
int t_listen(int fd, struct t_call *call);

DESCRIPTION  This routine is part of the XTI interfaces which evolved from the TLI interfaces. XTI represents the future evolution of these interfaces. However, TLI interfaces are supported for compatibility. When using a TLI routine that has the same name as an XTI routine, a different header file, tiuser.h, must be used. Refer to the section, TLI COMPATIBILITY, for a description of differences between the two interfaces.

This function listens for a connection indication from a calling transport user. The parameter fd identifies the local transport endpoint where connection indications arrive, and on return, call contains information describing the connection indication. call points to a t_call structure, which contains the following members:

```
struct netbuf addr;
struct netbuf opt;
struct netbuf udata;
int sequence;
```

The netbuf structure is described in t_connect(3N). In call, addr returns the protocol address of the calling transport user, opt returns protocol-specific parameters associated with the connection indication, udata returns any user data sent by the caller on the connection indication, and sequence is a number that uniquely identifies the returned connection indication. The value of sequence enables the user to listen for multiple connection indications before responding to any of them.

Once t_listen() returns, the value of the addr field of call will be in a format that is usable inside future calls to t_connect(). Note, however that t_connect() may fail for other reasons, for example TADDRBUSY.

This function returns values for the addr, opt, and udata fields of call in accordance with the maxlen (see netbuf in t_connect()) field of each. Their maxlen fields must be set to reflect the maximum size of their associated buffers before t_listen() is called. No attribute information is returned for any call->addr, call->opt, or call->udata buffer for which the maxlen field is initially set to zero. (TLI users should refer to the error description for TBUFOVFLW in the TLI COMPATIBILITY section for important differences.)

By default, t_listen() executes in synchronous mode and waits for a connection indication to arrive before returning to the user. However, if O_NONBLOCK is set using t_open(3N) or fcntl(), t_listen() executes asynchronously, reducing to a poll for existing connection indications. If none are available, it returns -1 and sets t_errno to TNODATA.

VALID STATES  Legitimate states (see t_getstate(3N)) for a call to this routine are:

T_IDLE
T_INCON

modified 10 Feb 1997  SunOS 5.6  3N-1629
**RETURN VALUES**

`t_listen()` returns:

0  On success.

-1  On failure.

On failure, `t_errno` is set to indicate the error, and possibly `errno` is set.

**ERRORS**

On failure, `t_errno` is set to one of the following:

- **TBADF**: The specified file descriptor does not refer to a transport endpoint.
- **TBADQLEN**: The argument `qlen` of the endpoint referenced by `fd` is zero.
- **TBUFOVFLW**: The number of bytes allocated for an incoming argument (`maxlen`) is greater than 0 but not sufficient to store the value of that argument. The provider’s state, as seen by the user, changes to `T_INCON`, and the connection indication information to be returned in `call` is discarded. The value of `sequence` returned can be used to do a `t_snddis(3N)`.
- **TLOOK**: An asynchronous event has occurred on this transport endpoint and requires immediate attention.
- **TNODATA O_NONBLOCK**: was set, but no connection indications had been queued.
- **TNOTSUPPORT**: This function is not supported by the underlying transport provider.
- **TOUTSTATE**: The communications endpoint referenced by `fd` or `resfd` is not in one of the states in which a call to this function is valid.
- **TPROTO**: This error indicates that a communication problem has been detected between XTI and the transport provider for which there is no other suitable XTI `t_errno` value.
- **TQFULL**: The maximum number of outstanding indications has been reached for the endpoint referenced by `fd`.
- **TSYSERR**: A system error has occurred during execution of this function, `errno` will be set to the specific error.

**TLI COMPATIBILITY**

The XTI and TLI interface definitions have common names but use different header files. This, and other semantic differences between the two interfaces are described in the subsections below.

**Interface Header**

The XTI interfaces use the header file, `xti.h`. TLI interfaces should not use this header. They should use the header:

```
#include <tiuser.h>
```

**Error Description Values**

The `t_errno` values that can be set by the XTI interface and cannot be set by the TLI interface are:

```
TPROTO
```

3N-1630  SunOS 5.6  modified 10 Feb 1997
TBADQLEN
TQFULL

A t_errno value that this routine can return under different circumstances than its XTI counterpart is TBUFOVFLW. It can be returned even when the maxlen field of the corresponding buffer has been set to zero.

Option Buffers

The format of the options in an opt buffer is dictated by the transport provider. Unlike the XTI interface, the TLI interface does not fix the buffer format.

For more information refer to the Transport Interfaces Programming Guide.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO

fcntl(2), t_accept(3N), t_alloc(3N), t_bind(3N), t_connect(3N), t_getstate(3N), t_open(3N), t_optmgmt(3N), t_rcvconnect(3N), t_snddis(3N), attributes(5)

Transport Interfaces Programming Guide

NOTES

Some transport providers do not differentiate between a connection indication and the connection itself. If this is the case, a successful return of t_listen() indicates an existing connection.

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.
NAME
  t_look – look at the current event on a transport endpoint

SYNOPSIS
  cc [ flag ...] file ... -lnsl [ library ... ]
  #include <xti.h>
  int t_look(int fd);

DESCRIPTION
  This routine is part of the XTI interfaces which evolved from the TLI interfaces. XTI
  represents the future evolution of these interfaces. However, TLI interfaces are supported
  for compatibility. When using a TLI routine that has the same name as an XTI routine, a
  different header file, tiuser.h, must be used. Refer to the section, TLI COMPATIBILITY,
  for a description of differences between the two interfaces.

  This function returns the current event that is awaiting acknowledgement and that ori-
  ginated on the transport endpoint specified by fd.

  This function enables a transport provider to notify a transport user of an asynchronous
  event when the user is calling functions in synchronous mode. Those events that require
  immediate notification of the user are indicated by the error value TLOOK on the current
  or next function to be executed. Such events block the progress of communications until
  acknowledgement is given.

  Details on events which cause functions to produce the failure indication TLOOK may be
  found in Section 5.6 of X/Open CAE Specification: Networking Services, Issue 4.

  This function also permits a transport user to poll a transport endpoint periodically for
  asynchronous events.

VALID STATES
  Legitimate states (see t_getstate(3N)) for a call to this routine are every one except
  T_UNINIT.

RETURN VALUES
  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:

  T_CONNECT    Connect confirmation received
  T_DATA       Normal data received
  T_DISCONNECT Disconnect received
  T_EXDATA     Expedited data received
  T_GODATA     Flow control restrictions on normal data flow that led to a TFLOW
                error have been lifted. Normal data may be sent again.
  T_GOEXDATA   Flow control restrictions on expedited data flow that led to a
                TFLOW error have been lifted. Expedited data may be sent again.
  T_LISTEN     Connection indication received
  T_ORDREL     Orderly release indication
  T_UDERR      Datagram error indication
On failure, –1 is returned, t_errno is set to indicate the error, and possibly errno is set.

**ERRORS**

On failure, t_errno is set to one of the following:

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TBADF</td>
<td>The specified file descriptor does not refer to a transport endpoint.</td>
</tr>
<tr>
<td>TPROTO</td>
<td>This error indicates that a communication problem has been detected between XTI and the transport provider for which there is no other suitable XTI t_errno value.</td>
</tr>
<tr>
<td>T SYSERR</td>
<td>A system error has occurred during execution of this function, errno will be set to the specific error.</td>
</tr>
</tbody>
</table>

**TLI COMPATIBILITY**

The XTI and TLI interface definitions have common names but use different header files. This, and other semantic differences between the two interfaces are described in the subsections below.

**Interface Header**

The XTI interfaces use the header file, xti.h. TLI interfaces should not use this header. They should use the header:

```c
#include <tiuser.h>
```

**Return Values**

The return values that are defined by the XTI interface and cannot be returned by the TLI interface are:

- T_GODATA
- T_GOEXDATA

**Error Description Values**

The t_errno value that can be set by the XTI interface and cannot be set by the TLI interface is:

- TPROTO

For more information refer to the *Transport Interfaces Programming Guide*.

**ATTRIBUTES**

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**SEE ALSO**

t_getstate(3N), t_open(3N), t_snd(3N), t_sndudata(3N), attributes(5)

*Transport Interfaces Programming Guide*

---

modified 10 Feb 1997 SunOS 5.6 3N-1633
NAME	tmpfile – create a temporary file

SYNOPSIS	#include <stdio.h>

FILE *tmpfile(void);

DESCRIPTION	The tmpfile() function creates a temporary file and opens a corresponding stream. The
file will automatically be deleted when all references to the file are closed. The file is
opened as in fopen(3S) for update (w+).

The largest value that can be represented correctly in an object of type off_t will be esta-
blished as the offset maximum in the open file description.

RETURN VALUES	Upon successful completion, tmpfile() returns a pointer to the stream of the file that is
created. Otherwise, it returns a null pointer and sets errno to indicate the error.

ERRORS	The tmpfile() function will fail if:

EINTR	A signal was caught during tmpfile().

EMFILE	OPEN_MAX file descriptors are currently open in the calling process.

ENFILE	The maximum allowable number of files is currently open in the system.

ENOSPC	The directory or file system which would contain the new file cannot be
expanded.

The tmpfile() function may fail if:

EMFILE	FOPEN_MAX streams are currently open in the calling process.

ENOMEM	Insufficient storage space is available.

USAGE	The stream refers to a file which is unlinked. If the process is killed in the period between
file creation and unlinking, a permanent file may be left behind.

tmpfile() has an explicit 64-bit equivalent. See interface64(5).

SEE ALSO	unlink(2), fopen(3S), tmpnam(3S), interface64(5)
NAME

tmpnam, tmpnam_r, tempnam – create a name for a temporary file

SYNOPSIS

```c
#include <stdio.h>

char *tmpnam(char *s);
char *tmpnam_r(char *s);
char *tempnam(const char *dir, const char *pfx);
```

DESCRIPTION

These functions generate file names that can safely be used for a temporary file.

`tmpnam()` always generates a file name using the path-prefix defined as `P_tmpdir` in the `<stdio.h>` header. If `s` 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 `s` 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 `s`.

`tmpnam_r()` has the same functionality as `tmpnam()` except that if `s` is a NULL pointer, the function returns NULL.

`tempnam()` allows the user to control the choice of a directory. The argument `dir` points to the name of the directory in which the file is to be created. If `dir` 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.

Many applications prefer their temporary files to have certain favorite initial letter sequences in their names. Use the `pfx` 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.

`tempnam()` uses `malloc(3C)` to get space for the constructed file name, and returns a pointer to this area. Thus, any pointer value returned from `tempnam()` may serve as an argument to `free(3C)` (see `malloc(3C)`). If `tempnam()` 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.

`tempnam()` fails if there is not enough space.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>See NOTES below.</td>
</tr>
</tbody>
</table>
SEE ALSO  
creat(2), unlink(2), fopen(3S), free(3C), malloc(3C), mktemp(3C), tmpfile(3S), attributes(5)

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 multi-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  tnfctl_buffer_alloc, tnfctl_buffer_dealloc – allocate or deallocate a buffer for trace data

SYNOPSIS  

cc [ flag ... ] file ... −ltnfctl [ library ... ]
#include <tnf/tnfctl.h>

tnfctl_errcode_t tnfctl_buffer_alloc(tnfctl_handle_t *hndl,
const char *trace_file_name, size_t trace_buffer_size);

tnfctl_buffer_dealloc(tnfctl_handle_t *hndl);

DESCRIPTION  

tnfctl_buffer_alloc() allocates a buffer to which trace events are logged. When tracing a process using a tnfctl handle returned by tnfctl_pid_open(3X), tnfctl_exec_open(3X), tnfctl_indirect_open(3X), and tnfctl_internal_open(3X), trace_file_name is the name of the trace file to which trace events should be logged. It can be an absolute path specification or a relative path specification. If it is relative, the current working directory of the process that is calling tnfctl_buffer_alloc() is prefixed to trace_file_name. If the named trace file already exists, it is overwritten. For kernel tracing, that is, for a tnfctl handle returned by tnfctl_kernel_open(3X), trace events are logged to a trace buffer in memory; therefore, trace_file_name is ignored. Use tnfextract(1) to extract a kernel buffer into a file.

trace_buffer_size is the size in bytes of the trace buffer that should be allocated. An error is returned if an attempt is made to allocate a buffer when one already exists.

tnfctl_buffer_alloc() affects the trace attributes; use tnfctl_traceAttrs_get(3X) to get the latest trace attributes after a buffer is allocated.

tnfctl_buffer_dealloc() is used to deallocate a kernel trace buffer that is no longer needed. hndl must be a kernel handle, returned by tnfctl_kernel_open(3X). A process’s trace file cannot be deallocated using tnfctl_buffer_dealloc(). Instead, once the trace file is no longer needed for analysis and after the process being traced exits, use rm(1) to remove the trace file. Do not remove the trace file while the process being traced is still alive. tnfctl_buffer_dealloc() affects the trace attributes; use tnfctl_traceAttrs_get(3X) to get the latest trace attributes after a buffer is deallocated.

For a complete discussion of tnf tracing, see tracing(3X).

RETURN VALUES  

tnfctl_buffer_alloc() and tnfctl_buffer_dealloc() return TNFCTL_ERR_NONE upon success.

ERRORS  

The following error codes apply to tnfctl_buffer_alloc():

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TNFCTL_ERR_BUFEXISTS</td>
<td>A buffer already exists.</td>
</tr>
<tr>
<td>TNFCTL_ERR_ACCES</td>
<td>Permission denied; could not create a trace file.</td>
</tr>
<tr>
<td>TNFCTL_ERR_SIZEETOOSMALL</td>
<td>The trace_buffer_size requested is smaller than the minimum trace buffer size needed. Use trace_min_size of trace attributes in tnfctl_traceAttrs_get(3X) to determine the minimum size of the buffer.</td>
</tr>
</tbody>
</table>

modified 4 Mar 1997       SunOS 5.6       3X-1637
The requested trace file size is too big.

 Trace file name is NULL or the absolute path name is longer than MAXPATHLEN.

 A memory allocation failure occurred.

 An internal error occurred.

 The following error codes apply to tnfctl_buffer_dealloc():

 TNFCTL_ERR_BADARG

 TNFCTL_ERR_NOBUF

 Cannot deallocate a trace buffer unless tracing is stopped. Use tnfctl_trace_state_set(3X) to stop tracing.

 TNFCTL_ERR_INTERNAL

 An internal error occurred.

 ATTRIBUTES

 See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Availability</td>
<td>SUNWtnfc</td>
</tr>
<tr>
<td>MT Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

 SEE ALSO

 prex(1), rm(1), tnfxtract(1), TNF_PROBE(3X), libtnfctl(3X), tnfctl_exec_open(3X), tnfctl_indirect_open(3X), tnfctl_internal_open(3X), tnfctl_kernel_open(3X), tnfctl_pid_open(3X), tnfctl_trace_attrs_get(3X), tracing(3X), attributes(5)
NAME  tnfctl_close – close a tnfctl handle

SYNOPSIS  
```c
cc [ flag ... ] file ... -ltnfctl [ library ... ]
#include <tnf/tnfctl.h>

tnfctl_errcode_t tnfctl_close(tnfctl_handle_t *hndl,
    tnfctl_targ_op_t action);
```

DESCRIPTION  tnfctl_close() is used to close a tnfctl handle and to free up the memory associated with the handle. When the handle is closed, the tracing state and the states of the probes are not changed. tnfctl_close() can be used to close handles in any mode, that is, whether they were created by tnfctl_internal_open(3X), tnfctl_pid_open(3X), tnfctl_exec_open(3X), tnfctl_indirect_open(3X), or tnfctl_kernel_open(3X).

The `action` argument is only used in direct mode, that is, if `hndl` was created by tnfctl_exec_open(3X) or tnfctl_pid_open(3X). In direct mode, `action` specifies whether the process will proceed, be killed, or remain suspended. `action` may have the following values:

- **TNFCTL_TARG_DEFAULT**  Kills the target process if `hndl` was created with tnfctl_exec_open(3X), but lets it continue if it was created with tnfctl_pid_open(3X).
- **TNFCTL_TARG_KILL**  Kills the target process.
- **TNFCTL_TARG_RESUME**  Allows the target process to continue.
- **TNFCTL_TARG_SUSPEND**  Leaves the target process suspended. This is not a job control suspend. It is possible to attach to the process again with a debugger or with the tnfctl_pid_open(3X) interface. The target process can also be continued with prun(1).

RETURN VALUES  tnfctl_close() returns TNFCTL_ERR_NONE upon success.

ERRORS  The following error codes apply to tnfctl_close():

- **TNFCTL_ERR_BADARG**  A bad argument was sent in `action`.
- **TNFCTL_ERR_INTERNAL**  An internal error occurred.

ATTRIBUTES  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Availability</td>
<td>SUNWtnfc</td>
</tr>
<tr>
<td>MT Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

modified 4 Mar 1997  SunOS 5.6  3X-1639
SEE ALSO
prex(1), prun(1), TNF_PROBE(3X), libtnfctl(3X), tnfctl_exec_open(3X),
tnfctl_indirect_open(3X), tnfctl_kernel_open(3X), tnfctl_pid_open(3X), tracing(3X),
attributes(5)

*Programming Utilities Guide*
NAME

tnfctl_indirect_open, tnfctl_check_libs – control probes of another process where caller provides /proc functionality

SYNOPSIS

cc [ flag ...] file ... -ltncfl [ library ...]
#include <tnf/tnfctl.h>

tnfctl_errcode_t tnfctl_indirect_open(void *prochandle,
          tnfctl_ind_config_t *config,
          tnfctl_handle_t **ret_val);

tnfctl_errcode_t tnfctl_check_libs(tnfctl_handle_t *hndl);

DESCRIPTION

The interfaces tnfctl_indirect_open() and tnfctl_check_libs() are used to control probes in another process where the libtnfctl(3X) client has already opened proc(4) on the target process. An example of this is when the client is a debugger. Since these clients already use /proc on the target, libtnfctl(3X) cannot use /proc directly. Therefore, these clients must provide callback functions that can be used to inspect and to update the target process. The target process must load libtnfprobe.so.1 (defined in <tnf/tnfctl.h> as macro TNFCTL_LIBTNFPROBE).

The first argument prochandle is a pointer to an opaque structure that is used in the callback functions that inspect and update the target process. This structure should encapsulate the state that the caller needs to use /proc on the target process (the /proc file descriptor). The second argument, config, is a pointer to

typedef struct tnfctl_ind_config {
    int (*p_read)(void *prochandle, paddr_t addr, char *buf,
                  size_t size);
    int (*p_write)(void *prochandle, paddr_t addr, char *buf,
                   size_t size);
    pid_t (*p_getpid)(void *prochandle);
    int (*p_obj_iter)(void *prochandle, tnfctl_ind_obj_f *func,
                      void *client_data);
} tnfctl_ind_config_t;

The first field p_read is the address of a function that can read size bytes at address addr in the target image into the buffer buf. The function should return 0 upon success. The second field p_write is the address of a function that can write size bytes at address addr in the target image from the buffer buf. The function should return 0 upon success. The third field p_getpid is the address of a function that should return the process id of the target process (prochandle). The fourth field p_obj_iter is the address of a function that iterates over all load objects and the executable by calling the callback function func with client_data. If func returns 0, p_obj_iter should continue processing link objects. If func returns any other value, p_obj_iter should stop calling the callback function and return that value. p_obj_iter should return 0 if it iterates over all load objects.

modified 4 Mar 1997               SunOS 5.6
                             3X-1641
If a failure is returned by any of the functions in `config`, the error is propagated back as `PREX_ERR_INTERNAL` by the `libtnfctl` interface that called it.

The definition of `tnfctl_ind_obj_f` is:

```c
typedef int tnfctl_ind_obj_f(void *prochandle,
    const struct tnfctl_ind_obj_info *obj
    void *client_data);
```

```c
typedef struct tnfctl_ind_obj_info {
    int objfd; /* -1 indicates fd not available */
    paddr_t text_base; /* virtual addr of text segment */
    paddr_t data_base; /* virtual addr of data segment */
    const char *objname; /* null-term. pathname to loadobj */
} tnfctl_ind_obj_info_t;
```

`objfd` should be the file descriptor of the load object or executable. If it is −1, then `objname` should be an absolute pathname to the load object or executable. If `objfd` is not closed by `libtnfctl`, it should be closed by the load object iterator function. `text_base` and `data_base` are the addresses where the text and data segments of the load object are mapped in the target process.

Whenever the target process opens or closes a dynamic object, the set of available probes may change. See `dlopen(3X)` and `dlclose(3X)`. In indirect mode, call `tnfctl_check_libs()` when such events occur to make `libtnfctl` aware of any changes. In other modes this is unnecessary but harmless. It is also harmless to call `tnfctl_check_libs()` when no such events have occurred.

**RETURN VALUES**

`tnfctl_indirect_open()` and `tnfctl_check_libs()` return `TNFCTL_ERR_NONE` upon success.

**ERRORS**

The following error codes apply to `tnfctl_indirect_open()`:

- `TNFCTL_ERR_ALLOCFAIL` A memory allocation failure occurred.
- `TNFCTL_ERR_BUSY` Internal tracing is being used.
- `TNFCTL_ERR_NOLIBTNFPHONE` `libtnfprobe.so.1` is not loaded in the target process.
- `TNFCTL_ERR_INTERNAL` An internal error occurred.

The following error codes apply to `tnfctl_check_libs()`:

- `TNFCTL_ERR_ALLOCFAIL` A memory allocation failure occurred.
- `TNFCTL_ERR_INTERNAL` An internal error occurred.
ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Availability</td>
<td>SUNWtnfc</td>
</tr>
<tr>
<td>MT Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO

prex(1), TNF_PROBE(3X), dlclose(3X), dlopen(3X), libtnfctl(3X),
tnctl_probe_enable(3X), tnfctl_probe_trace(3X), tracing(3X), proc(4), attributes(5)

NOTES

tnfctl_indirect_open() should only be called after the dynamic linker has mapped in all the libraries (rtld sync point) and called only after the process is stopped. Indirect process probe control assumes the target process is stopped whenever any libtnfctl interface is used on it. For example, when used for indirect process probe control, tnfctl_probe_enable(3X) and tnfctl_probe_trace(3X) should be called only for a process that is stopped.
tnfctl_internal_open (3X)  Miscellaneous Library Functions

NAME  tnfctl_internal_open – create handle for internal process probe control

SYNOPSIS  cc [ flag ... ] file ... -ltnfctl [ library ... ]
#include <tnf/tnfctl.h>

tnfctl_errcode_t tnfctl_internal_open(
    tnfctl_handle_t **ret_val);

DESCRIPTION  tnfctl_internal_open() returns in ret_val a pointer to an opaque handle that can be used
to control probes in the same process as the caller (internal process probe control). The
process must have libtnfprobe.so.1 loaded. Probes in libraries that are brought in by
dlopen(3X) will be visible after the library has been opened. Probes in libraries closed by
a dlclose(3X) will not be visible after the library has been disassociated. See the NOTES
section for more details.

RETURN VALUES  tnfctl_internal_open() returns TNFCTL_ERR_NONE upon success.

ERRORS  
TNFCTL_ERR_ALLOCFAIL  A memory allocation failure occurred.
TNFCTL_ERR_BUSY  Another client is already tracing this program (internally or externally).
TNFCTL_ERR_NOLIBTNFPROBE  libtnfprobe.so.1 is not linked in the target process.
TNFCTL_ERR_INTERNAL  An internal error occurred.

ATTRIBUTES  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Availability</td>
<td>SUNWtnfc</td>
</tr>
<tr>
<td>MT Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  ld(1), prex(1), TNF_PROBE(3X), dlopen(3X), dlclose(3X), libtnfctl(3X), tracing(3X),
attributes(5)

Programming Utilities Guide
Linker and Libraries Guide

NOTES  libtnfctl interposes on dlopen(3X) and dlclose(3X) in order to be notified of libraries
being dynamically opened and closed. This interposition is necessary for internal process
probe control to update its list of probes. In these interposition functions, a lock is
acquired to synchronize on traversal of the library list maintained by the runtime linker.
To avoid deadlocking on this lock, tnfctl_internal_open() should not be called from
within the init section of a library that can be opened by dlopen(3X).

Since interposition does not work as expected when a library is opened dynamically,
tnfctl_internal_open() should not be used if the client opened libtnfctl through
dlopen(3X). In this case, the client program should be built with a static dependency on

3X-1644  SunOS 5.6  modified 4 Mar 1997
libtnfctl. Also, if the client program is explicitly linking in `-ldl`, it should link `-ltnfctl` before `-ldl`.

Probes in filtered libraries (see `ld(1)`) will not be seen because the filtee (backing library) is loaded lazily on the first symbol reference and not at process startup or `dlopen(3X)` time. A workaround is to call `tnfctl_check_libs(3X)` once the caller is sure that the filtee has been loaded.
NAME tnfctl_kernel_open – create handle for kernel probe control

SYNOPSIS cc [flag ...] file ... -ltntctl [ library ... ]
#include <tnf/tnfctl.h>

tnfctl_errcode_t tnfctl_kernel_open(
    tnfctl_handle_t **ret_val);

DESCRIPTION tnfctl_kernel_open() starts a kernel tracing session and returns in ret_val an opaque handle that can be used to control tracing and probes in the kernel. Only one kernel tracing session is possible at a time on a given machine. An error code of T NFCTL_ERR_BUSY is returned if there is another process using kernel tracing. Use the command

    fuser -f /dev/tnfctl

to print the process id of the process currently using kernel tracing. Only a superuser may use tnfctl_kernel_open(). An error code of T NFCTL_ERR_ACCES is returned if the caller does not have the necessary privileges.

RETURN VALUES tnfctl_kernel_open returns T NFCTL_ERR_NONE upon success.

ERRORS T NFCTL_ERR_ACCES  Permission denied. Superuser privileges are needed for kernel tracing.
T NFCTL_ERR_BUSY  Another client is currently using kernel tracing.
T NFCTL_ERRALLOCFAIL  Memory allocation failed.
T NFCTL_ERR_FILENOTFOUND  /dev/tnfctl not found.
T NFCTL_ERR_INTERNAL  Some other failure occurred.

ATTRIBUTES See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Availability</td>
<td>SUNWtnfc</td>
</tr>
<tr>
<td>MT Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO prex(1), fuser(1M), TNF_PROBE(3X), libtnfctl(3X), tracing(3X), tnf_kernel_probes(4), attributes(5)
NAME

tnfctl_pid_open, tnfctl_exec_open, tnfctl_continue – interfaces for direct probe and process control for another process

SYNOPSIS

cc [flag ...] file ... -ltnfctl [ library ...]

#include <tnf/tnfctl.h>

tnfctl_errcode_t tnfctl_pid_open(pid_t pid, tnfctl_handle_t **ret_val);

tnfctl_errcode_t tnfctl_exec_open(
    const char *pgm_name,
    char * const *argv,
    char * const *envp,
    const char *libnfprobe_path,
    tnfctl_handle_t **ret_val);

tnfctl_errcode_t tnfctl_continue(tnfctl_handle_t *hndl, tnfctl_event_t *evt, tnfctl_handle_t **child_hdl);

DESCRIPTION

tnfctl_pid_open(), tnfctl_exec_open(), and tnfctl_continue() are the interfaces used to create handles to control probes in another process (direct process probe control). Either tnfctl_pid_open() or tnfctl_exec_open() will return a handle in ret_val that can be used for probe control. On return of these calls, the process is stopped. tnfctl_continue() allows the process specified by hndl to continue execution.

tnfctl_pid_open() attaches to a running process with process id of pid. The process is stopped on return of this call. tnfctl_pid_open() returns an error message if pid is the same as the calling process. See tnfctl_internal_open(3X) for information on internal process probe control. A pointer to an opaque handle is returned in ret_val, which can be used to control the process and the probes in the process. The target process must have libtnfprobe.so.1 (defined in <tnf/tnfctl.h> as macro TNFCTL_LIBTNFPROBE) linked in for probe control to work.

tnfctl_exec_open() is used to exec(2) a program and obtain a probe control handle. For probe control to work, the process image to be exec’d must load libtnfprobe.so.1. The interface tnfctl_exec_open() makes it simple for the library to be loaded at process start up time. pgm_name is the command to exec. If pgm_name is not an absolute path, then the $PATH environment variable is used to find the pgm_name. argv is a null-terminated argument pointer, that is, it is a null-terminated array of pointers to null-terminated strings. These strings constitute the argument list available to the new process image. argv must have at least one member, and it should point to a string that is the same as pgm_name. See execve(2). libnfprobe_path is an optional argument, and if set, it should be the path to the directory that contains libtnfprobe.so.1. There is no need for a trailing ”/” in this argument. This argument is useful if libtnfprobe.so.1 is not installed in /usr/lib. ld_preload is a space-separated list of libraries to preload into the target program. This string should follow the syntax guidelines of the LD_PRELOAD environment variable.

modified 4 Mar 1997

SunOS 5.6

3X-1647
See *ld.so.1*(1). The following illustrates how strings are concatenated to form the `LD_PRELOAD` environment variable in the new process image:

```
<current value of $LD_PRELOAD> + <space> +
libtnfprobe_path + "/libtnfprobe.so.1" + <space> +
ld_preload
```

This option is useful for preloading interposition libraries that have probes in them.

`envp` is an optional argument, and if set, it is used for the environment of the target program. It is a null-terminated array of pointers to null-terminated strings. These strings constitute the environment of the new process image. See *execve*(2). If `envp` is set, it overrides `ld_preload`. In this case, it is the caller's responsibility to ensure that `libtnfprobe.so.1` is loaded into the target program. If `envp` is not set, the new process image inherits the environment of the calling process, except for `LD_PRELOAD`.

`ret_val` is the return argument which is the handle that can be used to control the process and the probes within the process. Upon return, the process is stopped before any user code, including `.init` sections, has been executed.

*tnfctl_continue()* is a blocking call and lets the target process referenced by `hndl` continue running. It can only be used on handles returned by *tnfctl_pid_open()* and *tnfctl_exec_open()* (direct process probe control). It returns when the target stops; the reason that the process stopped is returned in `evt`. This call is interruptible by signals. If it is interrupted, the process is stopped, and `TNFCTL_EVENT_EINTR` is returned in `evt`.

The client of this library will have to decide which signal implies a stop to the target and catch that signal. Since a signal interrupts *tnfctl_continue()* , it will return, and the caller can decide whether or not to call *tnfctl_continue()* again.

*tnfctl_continue()* returns with an event of `TNFCTL_EVENT_DLOPEN`, `TNFCTL_EVENT_DLCLOSE`, `TNFCTL_EVENT_EXEC`, `TNFCTL_EVENT_FORK`, `TNFCTL_EVENT_EXIT`, or `TNFCTL_EVENT_TARGGONE`, respectively, when the target program does a `dlopen(3X)`, `dlclose(3X)`, any flavor of `exec(2)`, `fork(2)` (or `fork1(2)`), `exit(2)`, or terminates unexpectedly. If the target program did an `exec(2)`, then the client needs to call *tnfctl_close(3X)* on the current handle leaving the target resumed, suspended, or killed (second argument to *tnfctl_close(3X)*). No other `libtnfctl` interface call can be used on the existing handle. If the client wants to control the `exec`'ed image, it should leave the old handle suspended, and use *tnfctl_pid_open()* to reattach to the same process. This new handle can then be used to control the `exec`'ed image. See EXAMPLES below for sample code. If the target process did a `fork(2)` or `fork1(2)`, and if control of the child process is not needed, then `child_hndl` should be NULL. If control of the child process is needed, then `child_hndl` should be set. If it is set, a pointer to a handle that can be used to control the child process is returned in `child_hndl`. The child process is stopped at the end of the `fork()` system call. See EXAMPLES for an example of this event.

**RETURN VALUES**

*tnfctl_pid_open()* , *tnfctl_exec_open()* , and *tnfctl_continue()* return `TNFCTL_ERR_NONE` upon success.
The following error codes apply to `tnfctl_pid_open()`:

- **TNFCTL_ERR_BADARG**: The *pid* specified is the same process. Use `tnfctl_internal_open(3X)` instead.
- **TNFCTL_ERR_ACCES**: Permission denied. No privilege to connect to a setuid process.
- **TNFCTL_ERR_ALLOCFAIL**: A memory allocation failure occurred.
- **TNFCTL_ERR_BUSY**: Another client is already using `/proc` to control this process or internal tracing is being used.
- **TNFCTL_ERR_NOTDYNAMIC**: The process is not a dynamic executable.
- **TNFCTL_ERR_NOPROCESS**: No such target process exists.
- **TNFCTL_ERR_NOLIBTNFPROBE**: `libtnfprobe.so.1` is not linked in the target process.
- **TNFCTL_ERR_INTERNAL**: An internal error occurred.

The following error codes apply to `tnfctl_exec_open()`:

- **TNFCTL_ERR_ACCES**: Permission denied.
- **TNFCTL_ERR_ALLOCFAIL**: A memory allocation failure occurred.
- **TNFCTL_ERR_NOTDYNAMIC**: The target is not a dynamic executable.
- **TNFCTL_ERR_NOLIBTNFPROBE**: `libtnfprobe.so.1` is not linked in the target process.
- **TNFCTL_ERR_FILENOTFOUND**: The program is not found.
- **TNFCTL_ERR_INTERNAL**: An internal error occurred.

The following error codes apply to `tnfctl_continue()`:

- **TNFCTL_ERR_BADARG**: Bad input argument. `hndl` is not a direct process probe control handle.
- **TNFCTL_ERR_INTERNAL**: An internal error occurred.
- **TNFCTL_ERR_NOPROCESS**: No such target process exists.

These examples do not include any error-handling code. Only the initial example includes the declaration of the variables that is used for all the examples.

The following example shows how to preload `libtnfprobe.so.1` from the normal location and inherit the parent’s environment:

```c
const char *pgm;
char * const *argv;
tnfctl_handle_t *hndl, *new_hndl, *child_hndl;
tnfctl_errcode_t err;
char * const *envptr;
extern char **environ;
tnfctl_event_t evt;
int pid;

/* assuming argv has been allocated */
argv[0] = pgm;
```
This example shows how to preload two user-supplied libraries **libc_probe.so.1** and **libthread_probe.so.1**. They interpose on the corresponding **libc.so** and **libthread.so** interfaces and have probes for function entry and exit. **libtnfprobe.so.1** is preloaded from the normal location and the parent’s environment is inherited.

```c
err = tnfctl_exec_open(pgm, argv, NULL, NULL, NULL, &hndl);
```

This example preloads an interposition library **libc_probe.so.1**, and specifies a different location from which to preload **libtnfprobe.so.1**.

```c
err = tnfctl_exec_open(pgm, argv, NULL, NULL, NULL, &hndl);
```

To set up the environment explicitly for probe control to work, the target process must link **libtnfprobe.so.1**. If using **envp**, it is the caller’s responsibility to do so.

```c
err = tnfctl_exec_open(pgm, argv, envptr, NULL, NULL, &hndl);
```

Use this example to resume a process that does an `exec(2)` without controlling it.

```c
err = tnfctl_continue(hndl, &evt, NULL);
switch (evt) {
    case TNFCTL_EVENT_EXEC:
        /* let target process continue without control */
        err = tnfctl_close(hndl, TNFCTL_TARG_RESUME);
        ...
        break;
    }
```

Alternatively, use the next example to control a process that does an `exec(2)`.

```c
*/
  * assume the pid variable has been set by calling
  * tnfctl_trace_attrs_get()
  */
  err = tnfctl_continue(hndl, &evt, NULL);
switch (evt) {
    case TNFCTL_EVENT_EXEC:
```
To let fork’ed children continue without control, use NULL as the last argument to tnfctl_continue().

```c
err = tnfctl_continue(hndl, &evt, NULL);
```

The next example is how to control child processes that `fork(2)` or `fork1(2)` create.

```c
err = tnfctl_continue(hndl, &evt, &child_hndl);
switch (evt) {
    case TNFCTL_EVENT_FORK:
        /* spawn a new thread or process to control child_hndl */
        ...
        break;
}
```

## ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Availability</td>
<td>SUNWtnfc</td>
</tr>
<tr>
<td>MT Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

## SEE ALSO

ld(1), prex(1), proc(1), exec(2), execve(2), exit(2), fork(2), TNF_PROBE(3X), dlclose(3X), dlopen(3X), libtnfctl(3X), tnfctl_close(3X), tnfctl_internal_open(3X), tracing(3X) attributes(5)

Programming Utilities Guide  
Linker and Libraries Guide

## NOTES

After a tnfctl_continue() returns, a client should use tnfctl_trace_attrs_get(3X) to check the trace_buf_state member of the trace attributes and make sure that there is no internal error in the target.
NAME
   tnfctl_probe_apply, tnfctl_probe_apply_ids – iterate over probes

SYNOPSIS
   cc [ flag . . . ] file . . . -ltnfctl [ library . . . ]
   
   #include <tnf/tnfctl.h>

   tnfctl_errcode_t tnfctl_probe_apply(
      tnfctl_handle_t *hndl,
      tnfctl_probe_op_t probe_op,
      void *clientdata);

   tnfctl_errcode_t tnfctl_probe_apply_ids(
      tnfctl_handle_t *hndl,
      ulong_t probe_count,
      ulong_t *probe_ids,
      tnfctl_probe_op_t probe_op,
      void *clientdata);

DESCRIPTION
   tnfctl_probe_apply() is used to iterate over the probes controlled by
   hndl. For every probe, the probe_op function is called:

   typedef tnfctl_errcode_t (*tnfctl_probe_op_t)(
      tnfctl_handle_t *hndl,
      tnfctl_probe_t *probe_hndl,
      void *clientdata);

   Several predefined functions are available for use as probe_op. These functions are
   described in tnfctl_probe_state_get(3X).

   The clientdata supplied in
   tnfctl_probe_apply() is passed in as the last argument of probe_op. The probe_hndl in the
   probe operation function can be used to query or change the state of the probe. See
   tnfctl_probe_state_get(3X). The probe_op function should return TNSCTL_ERR_NONE
   upon success. It can also return an error code, which will cause tnfctl_probe_apply() to
   stop processing the rest of the probes and return with the same error code. Note that
   there are five (5) error codes reserved that the client can use for its own semantics. See
   ERRORS.

   The lifetime of probe_hndl is the same as the lifetime of hndl. It is good until hndl is closed
   by tnfctl_close(3X). Do not confuse a probe_hndl with hndl. The probe_hndl refers to a
   particular probe, while hndl refers to a process or the kernel. If probe_hndl is used in
   another libtnfctl(3X) interface, and it references a probe in a library that has been dynam-
   ically closed (see dlclose(3X)), then the error code TNSCTL_ERR_INVALIDPROBE will be
   returned by that interface.

   tnfctl_probe_apply_ids() is very similar to tnfctl_probe_apply(). The difference is that
   probe_op is called only for probes that match a probe id specified in the array of integers
   referenced by probe_ids. The number of probe ids in the array should be specified in
   probe_count. Use tnfctl_probe_state_get() to get the probe_id that corresponds to the
   probe_hndl.
**RETURN VALUES**

`tnfctl_probe_apply()` and `tnfctl_probe_apply_ids()` return `TNFCTL_ERR_NONE` upon success.

**ERRORS**

The following errors apply to both `tnfctl_probe_apply()` and `tnfctl_probe_apply_ids()`:

- **TNFCTL_ERR_INTERNAL**: An internal error occurred.
- **TNFCTL_ERR_USR1**: Error code reserved for user.
- **TNFCTL_ERR_USR2**: Error code reserved for user.
- **TNFCTL_ERR_USR3**: Error code reserved for user.
- **TNFCTL_ERR_USR4**: Error code reserved for user.
- **TNFCTL_ERR_USR5**: Error code reserved for user.

`tnfctl_probe_apply()` and `tnfctl_probe_apply_ids()` also return any error returned by the callback function `probe_op`.

The following errors apply only to `tnfctl_probe_apply_ids()`:

- **TNFCTL_ERR_INVALIDPROBE**: The probe handle is no longer valid. For example, the probe is in a library that has been closed by `dlclose(3X)`.

**EXAMPLES**

To enable all probes:

```c
tnfctl_probe_apply(hndl, tnfctl_probe_enable, NULL);
```

To disable the probes that match a certain pattern in the probe attribute string:

```c
/* To disable all probes that contain the string "vm" */
tnfctl_probe_apply(hndl, select_disable, "vm");
```

```c
static tnfctl_errcode_t
select_disable(tnfctl_handle_t *hndl, tnfctl_probe_t *probe_hndl, void *client_data)
{
    char *pattern = client_data;
    tnfctl_probe_state_t probe_state;

    tnfctl_probe_state_get(hndl, probe_hndl, &probe_state);
    if (strstr(probe_state.attr_string, pattern)) {
        tnfctl_probe_disable(hndl, probe_hndl, NULL);
    }
}
```

Note that these examples do not have any error handling code.

---

modified 4 Mar 1997

SunOS 5.6

3X-1653
ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Availability</td>
<td>SUNWtnfc</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
prex(1), TNF_PROBE(3X), dlclose(3X), dlopen(3X), libtnfctl(3X), tnfctl_close(3X), tnfctl_probe_state_get(3X), tracing(3X), tnf_kernel_probes(4), attributes(5)

Programming Utilities Guide
Linker and Libraries Guide
NAME

tnfctl_probe_state_get, tnfctl_probe_enable, tnfctl_probe_disable, tnfctl_probe_trace,
tnfctl_probe_untrace, tnfctl_probe_connect, tnfctl_probe_disconnect_all – interfaces to
query and to change the state of a probe

SYNOPSIS

cc [ flag ...] file ... -ltnfctl [ library ...]
#include <tnf/tnfctl.h>

tnfctl_errcode_t tnfctl_probe_state_get(
    tnfctl_handle_t *hndl, tnfctl_probe_t *probe_hndl,
    tnfctl_probe_state_t *state);

tnfctl_errcode_t tnfctl_probe_enable(tnfctl_handle_t *hndl,
    tnfctl_probe_t *probe_hndl, void *ignored);

tnfctl_errcode_t tnfctl_probe_disable(tnfctl_handle_t *hndl,
    tnfctl_probe_t *probe_hndl, void *ignored);

tnfctl_errcode_t tnfctl_probe_trace(tnfctl_handle_t *hndl,
    tnfctl_probe_t *probe_hndl, void *ignored);

tnfctl_errcode_t tnfctl_probe_untrace(tnfctl_handle_t *hndl,
    tnfctl_probe_t *probe_hndl, void *ignored);

tnfctl_errcode_t tnfctl_probe_disconnect_all(
    tnfctl_handle_t *hndl, tnfctl_probe_t *probe_hndl,
    void *ignored);

tnfctl_errcode_t tnfctl_probe_connect(tnfctl_handle_t *hndl,
    tnfctl_probe_t *probe_hndl, const char *lib_base_name,
    const char *func_name);

DESCRIPTION

tnfctl_probe_state_get() returns the state of the probe specified by probe_hndl in the pro-
cess or kernel specified by hndl. The user will pass these in to an apply iterator. The
caller must also allocate state and pass in a pointer to it. The semantics of the individual
members of state are:

id

The unique integer assigned to this probe. This number does not change over the lifetime of this probe. A probe_hndl can be
obtained by using the calls tnfctl_apply(), tnfctl_apply_ids(),
or tnfctl_register_funcs().

attr_string

A string that consists of attribute value pairs separated by semi-
colons. For the syntax of this string, see the syntax of the detail
argument of the TNF_PROBE(3X) macro. The attributes name,
slots, keys, file, and line are defined for every probe. Additional
user-defined attributes can be added by using the detail argu-
ment of the TNF_PROBE(3X) macro. An example of attr_string
follows:

"name pageout;slots vnode pages_pageout;
keys vm pageio io;file vm.c;line 25;"

modified 4 Mar 1997
SunOS 5.6
3X-1655
<table>
<thead>
<tr>
<th>enabled</th>
<th>B_TRUE if the probe is enabled, or B_FALSE if the probe is disabled. Probes are disabled by default. Use <code>tnfctl_probe_enable()</code> or <code>tnfctl_probe_disable()</code> to change this state.</th>
</tr>
</thead>
<tbody>
<tr>
<td>traced</td>
<td>B_TRUE if the probe is traced, or B_FALSE if the probe is not traced. Probes in user processes are traced by default. Kernel probes are untraced by default. Use <code>tnfctl_probe_trace()</code> or <code>tnfctl_probe_untrace()</code> to change this state.</td>
</tr>
<tr>
<td>new_probe</td>
<td>B_TRUE if this is a new probe brought in since the last change in libraries. See <code>dlopen(3X)</code> or <code>dlclose(3X)</code>. Otherwise, the value of new_probe will be B_FALSE. This field is not meaningful for kernel probe control.</td>
</tr>
<tr>
<td>obj_name</td>
<td>The name of the shared object or executable in which the probe is located. This string can be freed, so the client should make a copy of the string if it needs to be saved for use by other <code>libtnfctl</code> interfaces. In kernel mode, this string is always NULL.</td>
</tr>
<tr>
<td>func_names</td>
<td>A null-terminated array of pointers to strings that contain the names of functions connected to this probe. Whenever an enabled probe is encountered at runtime, these functions are executed. This array also will be freed by the library when the state of the probe changes. Use <code>tnfctl_probe_connect()</code> or <code>tnfctl_probe_disconnect_all()</code> to change this state.</td>
</tr>
<tr>
<td>func_addrs</td>
<td>A null-terminated array of pointers to addresses of functions in the target image connected to this probe. This array also will be freed by the library when the state of the probe changes.</td>
</tr>
<tr>
<td>client_registered_data</td>
<td>Data that was registered by the client for this probe by the creator function in <code>tnfctl_register_funcs(3X)</code>.</td>
</tr>
</tbody>
</table>

`tnfctl_probe_enable()`, `tnfctl_probe_disable()`, `tnfctl_probe_trace()`, `tnfctl_probe_untrace()`, and `tnfctl Probe_disconnect_all()` ignore the last argument. This convenient feature permits these functions to be used in the `probe_op` field of `tnfctl_probe_apply(3X)` and `tnfctl_probe_apply_ids(3X)`. `tnfctl_probe_enable()` enables the probe specified by `probe_hndl`. This is the master switch on a probe. A probe does not perform any action until it is enabled. `tnfctl Probe_disable()` disables the probe specified by `probe_hndl`. `tnfctl_probe_trace()` turns on tracing for the probe specified by `probe_hndl`. Probes emit a trace record only if the probe is traced. `tnfctl Probe_untrace()` turns off tracing for the probe specified by `probe_hndl`. This is useful if you want to connect probe functions to a probe without tracing it. `tnfctl Probe_connect()` connects the function `func_name` which exists in the library `lib_base_name`, to the probe specified by `probe_hndl`. `tnfctl Probe_connect()` returns an error code if used on a kernel tnfctl handle. `lib_base_name` is the base name (not a path) of the library. If it is NULL, and multiple functions in the target process match `func_name`, etc.
one of the matching functions is chosen arbitrarily. A probe function is a function that is in the target’s address space and is written to a certain specification. The specification is not currently published.

`tnf_probe_debug()` is one function exported by `libtnfprobe.so.1` and is the debug function that `prex(1)` uses. When the debug function is executed, it prints out the probe arguments and the value of the `sunw%debug` attribute of the probe to `stderr`.

`tnfctl_probe_disconnect_all()` disconnects all probe functions from the probe specified by `probe_hndl`.

Note that no `libtnfctl` call returns a probe handle (`tnfctl_probe_t`), yet each of the routines described here takes a `probe_hndl` as an argument. These routines may be used by passing them to one of the `tnfctl_probe_apply(3X)` iterators as the "op" argument. Alternatively, probe handles may be obtained and saved by a user’s "op" function, and they can be passed later as the `probe_hndl` argument when using any of the functions described here.

**RETURN VALUES**

- `tnfctl_probe_state_get()`, `tnfctl_probe_enable()`, `tnfctl_probe_disable()`, `tnfctl_probe_trace()`, `tnfctl_probe_untrace()`, `tnfctl_probe_disconnect_all()` and `tnfctl_probe_connect()` return `TNFCTL_ERR_NONE` upon success.

**ERRORS**

The following error codes apply to `tnfctl_probe_state_get()`:

- `TNFCTL_ERR_INVALIDPROBE` `probe_hndl` is no longer valid. The library that the probe was in could have been dynamically closed by `dlclose(3X)`.

The following error codes apply to `tnfctl_probe_enable()`, `tnfctl_probe_disable()`, `tnfctl_probe_trace()`, `tnfctl_probe_untrace()`, and `tnfctl_probe_disconnect_all()`:

- `TNFCTL_ERR_INVALIDPROBE` `probe_hndl` is no longer valid. The library that the probe was in could have been dynamically closed by `dlclose(3X)`.

- `TNFCTL_ERR_BUFBROKEN` Cannot do probe operations because tracing is broken in the target.

- `TNFCTL_ERR_NOBUF` Cannot do probe operations until a buffer is allocated. See `tnfctl_buffer_alloc(3X)`. This error code does not apply to kernel probe control.

The following error codes apply to `tnfctl_probe_connect()`:

- `TNFCTL_ERR_INVALIDPROBE` `probe_hndl` is no longer valid. The library that the probe was in could have been dynamically closed by `dlclose(3X)`.

- `TNFCTL_ERR_BADARG` The handle is a kernel handle, or `func_name` could not be found.

- `TNFCTL_ERR_BUFBROKEN` Cannot do probe operations because tracing is broken.
TNFCTL_ERR_NOBUF Cannot do probe operations until a buffer is allocated. See `tnfctl_buffer_alloc(3X)`.

ATTRIBUTES

See attributes(5) for description of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Availability</td>
<td>SUNWtnfc</td>
</tr>
<tr>
<td>MT Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO

`prex(1), TNF_PROBE(3X), libtnfctl(3X), tnfctl_check_libs(3X), tnfctl_continue(3X), tnfctl_probe_apply(3X), tnfctlProbe_apply_ids(3X), tracing(3X), tnf_kernel_probes(4), attributes(5)`

*Programming Utilities Guide*
NAME

tnfctl_register_funcs – register callbacks for probe creation and destruction

SYNOPSIS

```c
#include <tnf/tnfctl.h>

tnfctl_errcode_t tnfctl_register_funcs(tnfctl_handle_t *hndl,
    void (*create_func)(tnfctl_handle_t *, tnfctl_probe_t *),
    void (*destroy_func)(void *));
```

The function `tnfctl_register_funcs()` is used to store client-specific data on a per-probe basis. It registers a creator and a destructor function with `hndl`, either of which can be NULL. The creator function is called for every probe that currently exists in `hndl`. Every time a new probe is discovered, that is brought in by `dlopen(3X)`, `create_func` is called.

The return value of the creator function is stored as part of the probe state and can be retrieved by `tnfctl_probe_state_get(3X)` in the member field `client_registered_data`.

`destroy_func` is called for every probe handle that is freed. This does not necessarily happen at the time `dlclose(3X)` frees the shared object. The probe handles are freed only when `hndl` is closed by `tnfctl_close(3X)`. If `tnfctl_register_funcs()` is called a second time for the same `hndl`, then the previously registered destructor function is called first for all of the probes.

RETURN VALUES

`tnfctl_register_funcs()` returns `TNFCTL_ERR_NONE` upon success.

ERRORS

`TNFCTL_ERR_INTERNAL` An internal error occurred.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Availability</td>
<td>SUNWtnfc</td>
</tr>
<tr>
<td>MT Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO

`prex(1), T NF_PROBE(3X), dlclose(3X), dlopen(3X), libtnfctl(3X), tnfctl_close(3X), tnfctl_probe_state_get(3X), tracing(3X), tnf_kernel_probes(4), attributes(5)`

*Programming Utilities Guide*

*Linker and Libraries Guide*
tnfctl_strerror (3X)  Miscellaneous Library Functions

NAME  tnfctl_strerror – map a tnfctl error code to a string

SYNOPSIS  cc [ flag ...] file ... -ltnfctl [ library ...]
           #include <tnf/tnfctl.h>
           const char * tnfctl_strerror(tnfctl_errcode_t errcode);

DESCRIPTION  tnfctl_strerror() maps the error number in errcode to an error message string, and it returns a pointer to that string. The returned string should not be overwritten or freed.

ERRORS  tnfctl_strerror() returns the string "unknown libtnfctl.so error code" if the error number is not within the legal range.

ATTRIBUTES  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Availability</td>
<td>SUNWtnfc</td>
</tr>
<tr>
<td>MT Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  prex(1), TNF_PROBE(3X), libtnfctl(3X), tracing(3X), attributes(5)

Programming Utilities Guide
NAME
tnfctl_trace_attrs_get – get the trace attributes from a tnfctl handle

SYNOPSIS
cc [ flag ...] file ... -ltnfctl [ library ... ]
#include <tnf/tnfctl.h>

tnfctl_errcode_t tnfctl_trace_attrs_get(
    tnfctl_handle_t *hndl,
    tnfctl_trace_attrs_t *attrs);

DESCRIPTION
tnfctl_trace_attrs_get() returns the trace attributes associated with hndl in attrs. The
trace attributes can be changed by some of the other interfaces in libtnfctl(3X). It is the
client’s responsibility to use tnfctl_trace_attrs_get() to get the new trace attributes after
use of interfaces that change them. Typically, a client will use tnfctl_trace_attrs_get() after a call to
tnfctl_continue(3X) in order to make sure that tracing is still working. See
the discussion of trace_buf_state that follows.

Trace attributes are represented by the struct tnfctl_trace_attrs structure defined in
<tnf/tnfctl.h>:

    struct tnfctl_trace_attrs {
        pid_t targ_pid; /* not kernel mode */
        const char *trace_file_name; /* not kernel mode */
        size_t trace_buf_size;
        size_t trace_min_size;
        tnfctl_bufstate_t trace_buf_state;
        boolean_t trace_state;
        boolean_t filter_state; /* kernel mode only */
        long pad;
    };

The semantics of the individual members of attrs are:

targ_pid The process id of the target process. This is not valid for kernel
tracing.

trace_file_name The name of the trace file to which the target writes.

trace_buf_size The size of the trace buffer or file in bytes.

trace_min_size The minimum size in bytes of the trace buffer that can be allocated
by using the tnfctl_buffer_alloc(3X) interface.

trace_buf_state The state of the trace buffer. TNFCTL_BUF_OK indicates that a
trace buffer has been allocated. T NFCTL_BUF_NONE indicates that
no buffer has been allocated. T NFCTL_BUF_BROKEN indicates that

modified 4 Mar 1997 SunOS 5.6 3X-1661
there is an internal error in the target for tracing. The target will continue to run correctly, but no trace records will be written. To fix tracing, restart the process. For kernel tracing, deallocate the existing buffer with \texttt{tnfctl_buffer_dealloc(3X)} and allocate a new one with \texttt{tnfctl_buffer_alloc(3X)}.

**trace\_state**

The global tracing state of the target. Probes that are enabled will not write out data unless this state is on. This state is off by default for the kernel and can be changed by \texttt{tnfctl_trace_state_set(3X)}.

For a process, this state is on by default and can only be changed by \texttt{tnf_process_disable(3X)} and \texttt{tnf_process_enable(3X)}.

**filter\_state**

The state of process filtering. For kernel probe control, it is possible to select a set of processes for which probes are enabled. See \texttt{tnfctl_filter_list_get(3X)}, \texttt{tnfctl_filter_list_add(3X)}, and \texttt{tnfctl_filter_list_delete(3X)}. No trace output will be written when other processes traverse these probe points. By default process filtering is off, and all processes cause the generation of trace records when they hit an enabled probe. Use \texttt{tnfctl_filter_state_set(3X)} to change the filter state.

**RETURN VALUES**

\texttt{tnfctl_trace_attrs_get()} returns \texttt{TNFCTL\_ERR\_NONE} upon success.

**ERRORS**

The following error codes apply to \texttt{tnfctl_trace_attrs_get()}

- \texttt{TNFCTL\_ERR\_INTERNAL} An internal error occurred.

**ATTRIBUTES**

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Availability</td>
<td>SUNWtnfc</td>
</tr>
<tr>
<td>MT Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**SEE ALSO**

prex(1), \texttt{TNF\_PROBE(3X)}, \texttt{libtnfctl(3X)}, \texttt{tnfctl_buffer_alloc(3X)}, \texttt{tnfctl_continue(3X)}, \texttt{tnfctl_filter_list_get(3X)}, \texttt{tnf\_process\_disable(3X)}, \texttt{tracing(3X)}, attributes(5)
## NAME

*tnfctl_trace_state_set*, *tnfctl_filter_state_set*, *tnfctl_filter_list_get*, *tnfctl_filter_list_add*, *tnfctl_filter_list_delete* – control kernel tracing and process filtering

## SYNOPSIS

```c
cc [flag ...] file ... -ltnfctl [library ...]
#include <tnf/tnfctl.h>

tnfctl_errcode_t tnfctl_trace_state_set(
    tnfctl_handle_t *hndl, boolean_t trace_state);

tnfctl_errcode_t tnfctl_filter_state_set(
    tnfctl_handle_t *hndl, boolean_t filter_state);

tnfctl_errcode_t tnfctl_filter_list_get(
    tnfctl_handle_t *hndl,
    pid_t **pid_list, int *pid_count);

tnfctl_errcode_t tnfctl_filter_list_add(
    tnfctl_handle_t *hndl, pid_t pid_to_add);

tnfctl_errcode_t tnfctl_filter_list_delete(
    tnfctl_handle_t *hndl, pid_t pid_to_delete);
```

## DESCRIPTION

The interfaces to control kernel tracing and process filtering are used only with kernel handles, handles created by *tnfctl_kernel_open*(3X). These interfaces are used to change the tracing and filter states for kernel tracing.

*tnfctl_trace_state_set()* sets the kernel global tracing state to "on" if *trace_state* is *B_TRUE*, or to "off" if *trace_state* is *B_FALSE*. For the kernel, *trace_state* is off by default. Probes that are enabled will not write out data unless this state is on. Use *tnfctl_trace_attrs_get*(3X) to retrieve the current tracing state.

*tnfctl_filter_state_set()* sets the kernel process filtering state to "on" if *filter_state* is *B_TRUE*, or to "off" if *filter_state* is *B_FALSE*. *filter_state* is off by default. If it is on, only probe points encountered by processes in the process filter set by *tnfctl_filter_list_add()* will generate trace points. Use *tnfctl_trace_attrs_get*(3X) to retrieve the current process filtering state.

*tnfctl_filter_list_get()* returns the process filter list as an array in *pid_list*. The count of elements in the process filter list is returned in *pid_count*. The caller should use *free*(3C) to free memory allocated for the array *pid_list*.

*tnfctl_filter_list_add()* adds *pid_to_add* to the process filter list. The process filter list is maintained even when the process filtering state is off, but it has no effect unless the process filtering state is on.

*tnfctl_filter_list_delete()* deletes *pid_to_delete* from the process filter list. It returns an error if the process does not exist or is not in the filter list.

## RETURN VALUES

The interfaces *tnfctl_trace_state_set*, *tnfctl_filter_state_set*, *tnfctl_filter_list_add*, *tnfctl_filter_list_delete*, and *tnfctl_filter_list_get* return *TNFCTL_ERR_NONE* upon success.

---

*modified 4 Mar 1997*

*SunOS 5.6*

*3X-1663*
The following error codes apply to `tnfctl_trace_state_set`:
- **TNFCTL_ERR_BADARG**: The handle is not a kernel handle.
- **TNFCTL_ERR_NOBUF**: Cannot turn on tracing without a buffer being allocated.
- **TNFCTL_ERR_BUFBROKEN**: Tracing is broken in the target.
- **TNFCTL_ERR_INTERNAL**: An internal error occurred.

The following error codes apply to `tnfctl_filter_state_set`:
- **TNFCTL_ERR_BADARG**: The handle is not a kernel handle.
- **TNFCTL_ERR_INTERNAL**: An internal error occurred.

The following error codes apply to `tnfctl_filter_list_add`:
- **TNFCTL_ERR_BADARG**: The handle is not a kernel handle.
- **TNFCTL_ERR_NOPROCESS**: No such process exists.
- **TNFCTL_ERR_ALLOCFAIL**: A memory allocation failure occurred.
- **TNFCTL_ERR_INTERNAL**: An internal error occurred.

The following error codes apply to `tnfctl_filter_list_delete`:
- **TNFCTL_ERR_BADARG**: The handle is not a kernel handle.
- **TNFCTL_ERR_NOPROCESS**: No such process exists.
- **TNFCTL_ERR_INTERNAL**: An internal error occurred.

The following error codes apply to `tnfctl_filter_list_get`:
- **TNFCTL_ERR_BADARG**: The handle is not a kernel handle.
- **TNFCTL_ERR_ALLOCFAIL**: A memory allocation failure occurred.
- **TNFCTL_ERR_INTERNAL**: An internal error occurred.

**ATTRIBUTES**

See [attributes](5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Availability</td>
<td>SUNWtnfc</td>
</tr>
<tr>
<td>MT Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**SEE ALSO**

`prex(1)`, `TNF_PROBE(3X)`, `free(3C)`, `libtnfctl(3X)`, `tnfctl_kernel_open(3X)`, `tnfctl_trace_attrs_get(3X)`, `tracing(3X)`, `tnf_kernel_probes(4)`, [attributes](5)

*Programming Utilities Guide*
NAME
TNF_DECLARE_RECORD, TNF_DEFINE_RECORD_1, TNF_DEFINE_RECORD_2,
TNF_DEFINE_RECORD_3, TNF_DEFINE_RECORD_4, TNF_DEFINE_RECORD_5 –
TNF type extension interface for probes

SYNOPSIS
cc [ flag ... ] file ... [-ltnfprobe ] [ library ... ]
#include <tnf/probe.h>
TNF_DECLARE_RECORD(c_type, tnf_type);
TNF_DEFINE_RECORD_1(c_type, tnf_type,
        tnf_member_type_1, c_member_name_1)
TNF_DEFINE_RECORD_2(c_type, tnf_type,
        tnf_member_type_1, c_member_name_1,
        tnf_member_type_2, c_member_name_2)
TNF_DEFINE_RECORD_3(c_type, tnf_type,
        tnf_member_type_1, c_member_name_1,
        tnf_member_type_2, c_member_name_2,
        tnf_member_type_3, c_member_name_3)
TNF_DEFINE_RECORD_4(c_type, tnf_type,
        tnf_member_type_1, c_member_name_1,
        tnf_member_type_2, c_member_name_2,
        tnf_member_type_3, c_member_name_3,
        tnf_member_type_4, c_member_name_4)
TNF_DEFINE_RECORD_5(c_type, tnf_type,
        tnf_member_type_1, c_member_name_1,
        tnf_member_type_2, c_member_name_2,
        tnf_member_type_3, c_member_name_3,
        tnf_member_type_4, c_member_name_4,
        tnf_member_type_5, c_member_name_5)

DESCRIPTION
This macro interface is used to extend the TNF (Trace Normal Form) types that can be
used in TNF_PROBE(3X).
There should be only one TNF_DECLARE_RECORD and one TNF_DEFINE_RECORD
per new type being defined. The TNF_DECLARE_RECORD should precede the
TNF_DEFINE_RECORD. It can be in a header file that multiple source files share if those
source files need to use the tnf_type being defined. The TNF_DEFINE_RECORD should
only appear in one of the source files.
The TNF_DEFINE_RECORD macro interface defines a function as well as a couple of
data structures. Hence, this interface has to be used in a source file (.c or .cc file) at file
scope and not inside a function.
Note that there is no semicolon after the TNF_DEFINE_RECORD interface. Having one
will generate a compiler warning.

modified 31 Dec 1996 SunOS 5.6 3X-1665
Compiling with the preprocessor option \texttt{-DNPROBE} (see \texttt{cc(1B)}), or with the preprocessor control statement \texttt{#define NPROBE} ahead of the \texttt{#include <tnf/probe.h>} statement, will stop the TNF type extension code from being compiled into the program.

\textbf{c\_type} \hspace{1cm} \textit{c\_type} must be a C struct type. It is the template from which the new \textit{tnf\_type} is being created. Not all elements of the C struct need be provided in the TNF type being defined.

\textbf{tnf\_type} \hspace{1cm} \textit{tnf\_type} is the name being given to the newly created type. Use of this interface uses the name space prefixed by \textit{tnf\_type}. So, if a new type called \texttt{"xxx\_type"} is defined by a library, then the library should not use \texttt{"xxx\_type"} as a prefix in any other symbols it defines. The policy on managing the type name space is the same as managing any other name space in a library i.e., prefix any new TNF types by the unique prefix that the rest of the symbols in the library use. This would prevent name space collisions when linking multiple libraries that define new TNF types. For example, if a library \texttt{libpalloc.so} uses the prefix \texttt{"pal"} for all symbols it defines, then it should also use the prefix \texttt{"pal"} for all new TNF types being defined.

\textbf{tnf\_member\_type\_n} \hspace{1cm} \textit{tnf\_member\_type\_n} is the TNF type of the $n$th provided member of the C structure.

\textbf{tnf\_member\_name\_n} \hspace{1cm} \textit{tnf\_member\_name\_n} is the name of the $n$th provided member of the C structure.

\textbf{EXAMPLES} \hspace{1cm} This example shows how a new TNF type is defined and used in a probe. This code is assumed to be part of a fictitious library called \texttt{"libpalloc.so"} which uses the prefix \texttt{"pal"} for all it’s symbols.

\texttt{#include <tnf/probe.h>}

\begin{verbatim}
typedef struct pal_header {
    long size;
    char * descriptor;
    struct pal_header *next;
} pal_header_t;

TNF_DECLARE_RECORD(pal_header_t, pal_tnf_header);
TNF_DEFINE_RECORD_2(pal_header_t, pal_tnf_header,
    tnf_long, size,
    tnf_string, descriptor)

/*
 * Note: name space prefixed by pal_tnf_header should not be used by this
 * client anymore.
 */

void pal_free(pal_header_t *header_p)
{
}
\end{verbatim}
int state;

TNF_PROBE_2(pal_free_start, "palloc pal_free",
   "sunw%debug entering pal_free",
   tnf_long,   state_var, state,
   pal_tnf_header, header_var, header_p);

...
NAME
TNF_PROBE_0, TNF_PROBE_1, TNF_PROBE_2, TNF_PROBE_3, TNF_PROBE_4,
TNF_PROBE_5, TNF_PROBE_0_DEBUG, TNF_PROBE_1_DEBUG,
TNF_PROBE_2_DEBUG, TNF_PROBE_3_DEBUG, TNF_PROBE_4_DEBUG,
TNF_PROBE_5_DEBUG, TNF_DEBUG – probe insertion interface

SYNOPSIS
cc [ flag ... ] [ -DTNF_DEBUG ] file ... [ -lttnfprobe ] [ library ... ]
#include <tnf/probe.h>
TNF_PROBE_0(name, keys, detail);
TNF_PROBE_1(name, keys, detail,
    arg_type_1, arg_name_1, arg_value_1);
TNF_PROBE_2(name, keys, detail,
    arg_type_1, arg_name_1, arg_value_1,
    arg_type_2, arg_name_2, arg_value_2);
TNF_PROBE_3(name, keys, detail,
    arg_type_1, arg_name_1, arg_value_1,
    arg_type_2, arg_name_2, arg_value_2,
    arg_type_3, arg_name_3, arg_value_3);
TNF_PROBE_4(name, keys, detail,
    arg_type_1, arg_name_1, arg_value_1,
    arg_type_2, arg_name_2, arg_value_2,
    arg_type_3, arg_name_3, arg_value_3,
    arg_type_4, arg_name_4, arg_value_4);
TNF_PROBE_5(name, keys, detail,
    arg_type_1, arg_name_1, arg_value_1,
    arg_type_2, arg_name_2, arg_value_2,
    arg_type_3, arg_name_3, arg_value_3,
    arg_type_4, arg_name_4, arg_value_4,
    arg_type_5, arg_name_5, arg_value_5);
TNF_PROBE_0_DEBUG(name, keys, detail);
TNF_PROBE_1_DEBUG(name, keys, detail,
    arg_type_1, arg_name_1, arg_value_1);
TNF_PROBE_2_DEBUG(name, keys, detail,
    arg_type_1, arg_name_1, arg_value_1,
    arg_type_2, arg_name_2, arg_value_2);
TNF_PROBE_3_DEBUG(name, keys, detail,
    arg_type_1, arg_name_1, arg_value_1,
    arg_type_2, arg_name_2, arg_value_2,
    arg_type_3, arg_name_3, arg_value_3);
TNF_PROBE 4 DEBUG(name, keys, detail,
    arg_type_1, arg_name_1, arg_value_1,
    arg_type_2, arg_name_2, arg_value_2,
    arg_type_3, arg_name_3, arg_value_3,
    arg_type_4, arg_name_4, arg_value_4);

TNF_PROBE 5 DEBUG(name, keys, detail,
    arg_type_1, arg_name_1, arg_value_1,
    arg_type_2, arg_name_2, arg_value_2,
    arg_type_3, arg_name_3, arg_value_3,
    arg_type_4, arg_name_4, arg_value_4,
    arg_type_5, arg_name_5, arg_value_5);

DESCRIPTION

This macro interface is used to insert probes into C or C++ code for tracing. See tracing(3X) for a discussion of the Solaris tracing architecture, including example source code that uses it.

You can place probes anywhere in C and C++ programs including .init sections, .fini sections, multi-threaded code, shared objects, and shared objects opened by dlopen(3X).

Use probes to generate trace data for performance analysis or to write debugging output to stderr. Probes are controlled at runtime by prex(1).

The trace data is logged to a trace file in Trace Normal Form (TNF). The interface for the user to specify the name and size of the trace file is described in prex(1). Think of the trace file as the least recently used circular buffer. Once the file has been filled, newer events will overwrite the older ones.

Use TNF_PROBE 0 through TNF_PROBE 5 to create production probes. These probes are compiled in by default. Developers are encouraged to embed such probes strategically, and to leave them compiled within production software. Such probes facilitate on-site analysis of the software.

Use TNF_PROBE 0_DEBUG through TNF_PROBE 5_DEBUG to create debug probes. These probes are compiled out by default. If you compile the program with the preprocessor option −DTNF_DEBUG (see cc(1B)), or with the preprocessor control statement #define TNF_DEBUG ahead of the #include <tnf/probe.h> statement, the debug probes will be compiled into the program. When compiled in, debug probes differ in only one way from the equivalent production probes. They contain an additional "debug" attribute which may be used to distinguish them from production probes at runtime, for example, when using prex(1). Developers are encouraged to embed any number of probes for debugging purposes. Disabled probes have such a small runtime overhead that even large numbers of them do not make a significant impact.

If you compile with the preprocessor option −DNPROBE (see cc(1B)), or place the preprocessor control statement #define NPROBE ahead of the #include <tnf/probe.h> statement, no probes will be compiled into the program.
name

The name of the probe should follow the syntax guidelines for identifiers in ANSI C. The use of name declares it, hence no separate declaration is necessary. This is a block scope declaration, so it does not affect the name space of the program.

keys

keys is a string of space-separated keywords that specify the groups that the probe belongs to. Semicolons, single quotation marks, and the equal character (=) are not allowed in this string. If any of the groups are enabled, the probe is enabled. keys cannot be a variable. It must be a string constant.

detail

detail is a string that consists of <attribute> <value> pairs that are each separated by a semicolon. The first word (up to the space) is considered to be the attribute and the rest of the string (up to the semicolon) is considered the value. Single quotation marks are used to denote a string value. Besides quotation marks, spaces separate multiple values. The value is optional. Although semicolons or single quotation marks generally are not allowed within either the attribute or the value, when text with embedded spaces is meant to denote a single value, use single quotes surrounding this text.

Use detail for one of two reasons. First, use detail to supply an attribute that a user can type into prex(1) to select probes. For example, if a user defines an attribute called color, then prex(1) can select probes based on the value of color. Second, use detail to annotate a probe with a string that is written out to a trace file only once. prex(1) uses spaces to tokenize the value when searching for a match. Spaces around the semicolon delimiter are allowed. detail cannot be a variable; it must be a string constant. For example, the detail string:

"XYZ%debug 'entering function A'; XYZ%exception 'no file'; XYZ%func_entry; XYZ%color red blue"

consists of 4 units:

<table>
<thead>
<tr>
<th>attribute</th>
<th>value</th>
<th>values that prex matches on</th>
</tr>
</thead>
<tbody>
<tr>
<td>XYZ%debug</td>
<td>'entering function A'</td>
<td>'entering function A'</td>
</tr>
<tr>
<td>XYZ%exception</td>
<td>'no file'</td>
<td>'no file'</td>
</tr>
<tr>
<td>XYZ%func_entry</td>
<td>/.*/</td>
<td>(regular expression)</td>
</tr>
<tr>
<td>XYZ%color</td>
<td>red blue</td>
<td>red &lt;or&gt; blue</td>
</tr>
</tbody>
</table>

Attribute names must be prefixed by the vendor stock symbol followed by the ‘%’ character. This avoids conflicts in the attribute name space. All attributes that do not have a ‘%’ character are reserved. The following attributes are predefined:

<table>
<thead>
<tr>
<th>attribute</th>
<th>semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name of probe</td>
</tr>
<tr>
<td>keys</td>
<td>keys of the probe (value is space-separated tokens)</td>
</tr>
<tr>
<td>file</td>
<td>file name of the probe</td>
</tr>
<tr>
<td>line</td>
<td>line number of the probe</td>
</tr>
<tr>
<td>slots</td>
<td>slot names of the probe event (arg_name_n)</td>
</tr>
<tr>
<td>object</td>
<td>the executable or shared object that this probe is in.</td>
</tr>
<tr>
<td>debug</td>
<td>distinguishes debug probes from production probes</td>
</tr>
</tbody>
</table>
arg_type_n

This is the type of the \textit{n}th argument. The following are predefined TNF types:

<table>
<thead>
<tr>
<th>tnf type</th>
<th>associated C type (and semantics)</th>
</tr>
</thead>
<tbody>
<tr>
<td>tnf_int</td>
<td>int</td>
</tr>
<tr>
<td>tnf_uint</td>
<td>unsigned int</td>
</tr>
<tr>
<td>tnf_long</td>
<td>long</td>
</tr>
<tr>
<td>tnf_ulong</td>
<td>unsigned long</td>
</tr>
<tr>
<td>tnf_longlong</td>
<td>long long (if implemented in compilation system)</td>
</tr>
<tr>
<td>tnf_ulonglong</td>
<td>unsigned long long (if implemented in compilation system)</td>
</tr>
<tr>
<td>tnf_float</td>
<td>float</td>
</tr>
<tr>
<td>tnf_double</td>
<td>double</td>
</tr>
<tr>
<td>tnf_string</td>
<td>char *</td>
</tr>
<tr>
<td>tnf_opaque</td>
<td>void *</td>
</tr>
</tbody>
</table>

To define new TNF types that are records consisting of the predefined TNF types or references to other user defined types, use the interface specified in \texttt{TNF_DECLARE_RECORD}(3X).

arg_name_n

\textit{arg\_name\_n} is the name that the user associates with the \textit{n}th argument. Do not place quotation marks around \textit{arg\_name\_n}. Follow the syntax guidelines for identifiers in ANSI C. The string version of \textit{arg\_name\_n} is stored for every probe and can be accessed as the attribute "slots".

arg_value_n

\textit{arg\_value\_n} is evaluated to yield a value to be included in the trace file. A read access is done on any variables that are in mentioned in \textit{arg\_value\_n}. In a multi-threaded program, it is the user’s responsibility to place locks around the \texttt{TNF\_PROBE} macro if \textit{arg\_value\_n} contains a variable that should be read protected.

EXAMPLES

See \texttt{tracing}(3X) for complete examples showing debug and production probes in source code.

ATTRIBUTES

See \texttt{attributes}(5) for descriptions of the following attributes:

\begin{tabular}{|c|c|}
\hline
ATTRIBUTE TYPE & ATTRIBUTE VALUE \\
\hline
Availability & SUNWtnfd \\
MT Level & MT-Safe \\
\hline
\end{tabular}

SEE ALSO

\texttt{cc}(1B), \texttt{id}(1), \texttt{prex}(1), \texttt{tnfdump}(1), \texttt{libthread}(3T), \texttt{libtnfctl}(3X), \texttt{TNF\_DECLARE\_RECORD}(3X), \texttt{dlopen}(3X), \texttt{libtnfctl}(3X), \texttt{tnf\_process\_disable}(3X), \texttt{tracing}(3X), \texttt{attributes}(5)

Programmer Utilities Guide

NOTES

If attaching to a running program with \texttt{prex}(1) to control the probes, compile the program with \texttt{–ltnfprobe} or start the program with the environment variable \texttt{LD\_PRELOAD} set to \texttt{libtnfprobe.so.1}. See \texttt{id}(1). If \texttt{libtnfprobe} is explicitly linked into the program, it must be before \texttt{libthread} on the link line.

modified 4 Mar 1997 SunOS 5.6 3X-1671
NAME  tnf_process_disable, tnf_process_enable, tnf_thread_disable, tnf_thread_enable – probe control internal interface

SYNOPSIS  cc [ flag ... ] file ... -ltntfprobe [ library ... ]
#include <tnf/probe.h>
void tnf_process_disable(void);
void tnf_process_enable(void);
void tnf_thread_disable(void);
void tnf_thread_enable(void);

DESCRIPTION  There are three levels of granularity for controlling tracing and probe functions (called probing from here on) — probing for the entire process, a particular thread, and the probe itself can be disabled/enabled. The first two (process and thread) are controlled by this interface. The probe is controlled via the application prex(1).

tnf_process_disable() turns off probing for the process. The default process state is to have probing enabled. tnf_process_enable() turns on probing for the process.

tnf_thread_disable() turns off probing for the currently running thread. Threads are "born" or created with this state enabled. tnf_thread_enable() turns on probing for the currently running thread. If the program is a non-threaded program, these two thread interfaces disable or enable probing for the process.

ATTRIBUTES  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Availability</td>
<td>SUNWtnfd</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  prex(1), tnfdump(1), TNF_DECLARE_RECORD(3X), TNF_PROBE(3X), attributes(5)

NOTES  A probe is considered enabled only if:

- prex(1) has enabled the probe AND
- the process has probing enabled — which is the default or could be set via tnf_process_enable() AND
- the thread that hits the probe has probing enabled — which is every thread’s default or could be set via tnf_thread_enable().

There is a run time cost associated with determining that the probe is disabled. To reduce the performance effect of probes, this cost should be minimized. The quickest way that a probe can be determined to be disabled is by the enable control that prex(1) uses. Therefore, to disable all the probes in a process use the disable command in prex(1) rather than tnf_process_disable().
\texttt{tnf\_process\_disable()} and \texttt{tnf\_process\_enable()} should only be used to toggle probing based on some internal program condition. \texttt{tnf\_thread\_disable()} should be used to turn off probing for threads that are uninteresting.
NAME  toascii – translate integer to a 7-bit ASCII character

SYNOPSIS  #include <ctype.h>
            int toascii(int c);

DESCRIPTION  The toascii() function converts its argument into a 7-bit ASCII character.

RETURN VALUES  The toascii() function returns the value (c & 0x7f).

ERRORS  No errors are returned.

ATTRIBUTES  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
<tr>
<td>CSI</td>
<td>Enabled</td>
</tr>
</tbody>
</table>

SEE ALSO  isascii(3C), attributes(5)
NAME  _tolower – transliterate upper-case characters to lower-case

SYNOPSIS  #include <ctype.h>
            int _tolower(int c);

DESCRIPTION  The _tolower() macro is equivalent to tolower(3C) except that the argument c must be an upper-case letter.

RETURN VALUES  On successful completion, _tolower() returns the lower-case letter corresponding to the argument passed.

ERRORS  No errors are defined.

ATTRIBUTES  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
<tr>
<td>CSI</td>
<td>Enabled</td>
</tr>
</tbody>
</table>

SEE ALSO  isupper(3C), tolower(3C), attributes(5)
NAME
tolower – transliterate upper-case characters to lower-case

SYNOPSIS
#include <ctype.h>
int tolower(int c);

DESCRIPTION
The tolower() function has as a domain a type int, the value of which is representable as
an unsigned char or the value of EOF. If the argument has any other value, the argument
is returned unchanged. If the argument of tolower() represents an upper-case letter, and
there exists a corresponding lower-case letter (as defined by character type information in
the program locale category LC_CTYPE), the result is the corresponding lower-case letter.
All other arguments in the domain are returned unchanged.

RETURN VALUES
On successful completion, tolower() returns the lower-case letter corresponding to the
argument passed. Otherwise, it returns the argument unchanged.

ERRORS
No errors are defined.

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
<tr>
<td>CSI</td>
<td>Enabled</td>
</tr>
</tbody>
</table>

SEE ALSO
_tolower(3C), setlocale(3C), attributes(5)
NAME

`t_open` – establish a transport endpoint

SYNOPSIS

```c
cc [ flag ...] file ... -lnsl [ library ... ]
#include <xti.h>
#include <fcntl.h>
int t_open(const char *name, int oflag, struct t_info *info);
```

DESCRIPTION

This routine is part of the XTI interfaces which evolved from the TLI interfaces. XTI represents the future evolution of these interfaces. However, TLI interfaces are supported for compatibility. When using a TLI routine that has the same name as an XTI routine, a different header file, `tiuser.h`, must be used. Refer to the section, **TLI COMPATIBILITY**, for a description of differences between the two interfaces.

The `t_open()` function must be called as the first step in the initialization of a transport endpoint. This function establishes a transport endpoint by taking a supplied protocol identifier, `name`, and by returning a file descriptor that identifies that endpoint.

The argument `name` points to a transport provider identifier and `oflag` identifies any open flags (as in `open(2)`). The argument `oflag` can be specified using `O_RDWR` or using a bitwise inclusive-OR operation with the required value `O_RDWR` and the optional value `O_NONBLOCK`. 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.

This function also returns various default characteristics of the underlying transport protocol by setting fields in a `t_info` structure.

A `t_info` contains the following members:

- `long addr; /* max size of the transport protocol address */`
- `long options; /* max number of bytes of /* protocol-specific options */`
- `long tsdu; /* max size of a transport service data /* unit (TSDU) */`
- `long etsdu; /* max size of an 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()` and `t_rcvdis()` functions */`
- `long servtype; /* service type supported by the /* transport provider */`
- `long flags; /* other info about the transport provider */`

modified 10 Feb 1997

SunOS 5.6

3N-1677
The values of the fields have the following meanings:

**addr**
A value greater than zero (>T_NULL) indicates the maximum size of a transport protocol address and a value of −2 (T_INVALID) specifies that the transport provider does not provide user access to transport protocol addresses.

**options**
A value greater than zero (>T_NULL) indicates the maximum number of bytes of protocol-specific options supported by the provider, and a value of −2(T_INVALID) specifies that the transport provider does not support user-selectable options.

**tsdu**
A value greater than zero (>T_NULL) specifies the maximum size of a transport service data unit (TSDU); a value of zero (T_NULL) specifies that the transport provider does not support the concept of a TSDU value, although it does support the sending of a data stream with no logical boundaries preserved across a connection; a value of −1 (T_INFINITE) specifies that there is no limit to the size of a TSDU; and a value of −2 (T_INVALID) specifies that the transfer of normal data is not supported by the transport provider.

**etsdu**
A value greater than zero (>T_NULL) specifies the maximum size of an expedited transport service data unit (ETSDU); a value of zero (T_NULL) specifies that the transport provider does not support the concept of ETSDU, although it does support the sending of an expedited data stream with no logical boundaries preserved across a connection; a value of −1 (T_INFINITE) specifies that there is no limit on the size of an ETSDU; and a value of −2 (T_INVALID) specifies that the transfer of expedited data is not supported by the transport provider. Note that the semantics of expedited data may be quite different for different transport providers.

**connect**
A value greater than zero (>T_NULL) specifies the maximum amount of data that may be associated with connection establishment functions, and a value of −2 (T_INVALID) specifies that the transport provider does not allow data to be sent with connection establishment functions.

**discon**
a value greater than zero (>T_NULL) specifies the maximum amount of data that may be associated with the t_snddis() and t_rcvdis() functions, and a value of −2 (T_INVALID) specifies that the transport provider does not allow data to be sent with the abortive release functions.

**servtype**
This field specifies the service type supported by the transport provider, as described below.

**flags**
This is a bit field used to specify other information about the communications provider. If the T_SENDZERO bit is set in flags, this indicates the underlying transport provider supports the sending of zero-length TSDUs.

If a transport user is concerned with protocol independence, the above sizes may be accessed to determine how large the buffers must be to hold each piece of information. Alternatively, the t_alloc(3N) function may be used to allocate these buffers. An error will result if a transport user exceeds the allowed data size on any function.
The **servtype** field `info` specifies one of the following values on return:

- **T_COTS** The transport provider supports a connection-mode service but does not support the optional orderly release facility.
- **T_COTS_ORD** The transport provider supports a connection-mode service with the optional orderly release facility.
- **T_CLTS** The transport provider supports a connectionless-mode service. For this service type, `t_open()` will return −2 (T_INVALID) for `etsdu`, `connect`, and `discon`.

A single transport endpoint may support only one of the above services at one time. If `info` is set to a null pointer by the transport user, no protocol information is returned by `t_open()`.

**VALID STATES**

The only legitimate state (see `t_getstate(3N)`) for a call to this routine is the conceptual state T_UNINIT.

**RETURN VALUES**

- **t_open()** returns:
  - A **Valid File Descriptor** On success.
  - −1 On failure.

On failure, `t_errno` is set to indicate the error, and possibly `errno` is set.

**ERRORS**

On failure, `t_errno` is set to the following:

- **TBADFLAG** An invalid flag is specified.
- **TBADNAME** Invalid transport provider name.
- **TPROTO** This error indicates that a communication problem has been detected between XTI and the transport provider for which there is no other suitable XTI `t_errno` value.
- **TSYSERR** A system error has occurred during execution of this function, `errno` will be set to the specific error.

**TLI COMPATIBILITY**

The XTI and TLI interface definitions have common names but use different header files. This, and other semantic differences between the two interfaces are described in the subsections below.

**Interface Header**

The XTI interfaces use the header file, `xti.h`. TLI interfaces should not use this header. They should use the header:

```c
#include <tiuser.h>
```

**Error Description Values**

The `t_errno` values that can be set by the XTI interface and cannot be set by the TLI interface are:

- **TPROTO**
- **TBADNAME**
Notes

For TLI, the t_info structure referenced by info lacks the following structure member:

```c
long flags;
/* other info about the transport provider */
```

This member was added to struct t_info in the XTI interfaces.

When a value of −1 is observed as the return value in various t_info structure members, it signifies that the transport provider can handle an infinite length buffer for a corresponding attribute, such as address data, option data, TSDU (octet size), ETSDU (octet size), connection data, and disconnection data. The corresponding structure members are `addr`, `options`, `tsdu`, `estdu`, `connect`, and `discon`, respectively.

For more information refer to the `Transport Interfaces Programming Guide`.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO t_alloc(3N), open(2), t_getinfo(3N), t_getstate(3N), attributes(5)

`Transport Interfaces Programming Guide`
NAME
t_optmgmt – manage options for a transport endpoint

SYNOPSIS
cc [ flag ... ] file ... -lnsl [ library ... ]
#include <xti.h>
int t_optmgmt(int fd, const struct t_optmgmt *req, struct t_optmgmt *ret);

DESCRIPTION
This routine is part of the XTI interfaces which evolved from the TLI interfaces. XTI represents the future evolution of these interfaces. However, TLI interfaces are supported for compatibility. When using a TLI routine that has the same name as an XTI routine, a different header file, tuser.h, must be used. Refer to the section, TLI COMPATIBILITY, for a description of differences between the two interfaces.

The t_optmgmt() function enables a transport user to retrieve, verify, or negotiate protocol options with the transport provider. The argument fd identifies a transport endpoint. The req and ret arguments point to a t_optmgmt structure containing the following members:

struct netbuf opt;
long flags;

The opt field identifies protocol options and the flags field is used to specify the action to take with those options.

The options are represented by a netbuf structure (see t_connect(3N)) in a manner similar to the address in t_bind(3N).

The argument req is used to request a specific action of the provider and to send options to the provider. The len field specifies the number of bytes in the options, the buf field points to the options buffer, and the maxlen field has no meaning for the req argument. The transport provider may return options and flag values to the user through ret. For ret, the maxlen field specifies the maximum size of the options buffer and the buf field points to the buffer where the options are to be placed. If the maxlen field in ret is set to zero, no options values are returned. (TLI users should refer to the error description for TBUFOVFLW in the TLI COMPATIBILITY section for important differences.) On return, the len field specifies the number of bytes of options returned. The value in the maxlen field has no meaning for the req argument, but must be set in the ret argument to specify the maximum number of bytes the options buffer can hold.

Options are conveyed in two buffers. Access to these buffers is moderated through two instances of the structure type netbuf, which in turn are referenced through the opt field of the t_optmgmt structures that are referenced by ret and req.

The following text graphic illustrates the layout of the option buffer. Note: TLI users please notice the option buffer differences mentioned in the TLI COMPATIBILITY section. XTI users please read the Warnings subsection.
The **opt** buffers can hold various options, each of which is an instance of the structure type **t_opthdr** followed by a variable-length option value, the meaning of which is context-dependent.

The values of fields in the **t_opthdr** determine the context for a particular option value:

<table>
<thead>
<tr>
<th>len</th>
<th>level</th>
<th>name</th>
<th>status</th>
<th>value...</th>
<th>~ ~ ~</th>
<th>len...------------------------------------- - - ----------------</th>
</tr>
</thead>
<tbody>
<tr>
<td>32bit</td>
<td>32bit</td>
<td>32bit</td>
<td>32bit</td>
<td>^ 32bit</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

alignment bytes

The **opt** buffers can hold various options, each of which is an instance of the structure type **t_opthdr** followed by a variable-length option value, the meaning of which is context-dependent.

The values of fields in the **t_opthdr** determine the context for a particular option value:

- **len** Specifies the total length that the option value occupies in the buffer, not counting any padding bytes for boundary-alignment purposes. It is the sum of the length of the header (**t_opthdr**) and the length of the option value that usually follows.
- **level** Identifies the protocol or API associated with the option.
- **name** Identifies a particular option applicable to the protocol or API corresponding to **level**.
- **status** Indicates the success or failure of a negotiation, one of many possible types of actions that the **flags** field can specify.

Several options can be concatenated into a single **opt** buffer. However, in doing so, the transport user has to ensure that each option starts at a 32-bit or **uint32_t** boundary (same as long word boundary in ILP32 representation).

Each option in both the input (**req**) and output (**ret**) option buffers must start at a 32-bit boundary.

The macro **OPT_NEXTHDR(pbuf, buflen, poption)** can help dispatch these alignment requirements. This function macro can be an aid for both writing to and reading from the option buffers. It returns a pointer to the position of the next option or returns a null pointer if the option buffer is exhausted. In calls to this function macro, the parameter **pbuf** denotes a pointer to an option buffer **opt.buf**, and **buflen** is its length. The parameter **poption** points to the current option in the option buffer.

If the transport user specifies several options on input, all options must address the same level.

If any option in the options buffer does not indicate the same level as the first option, or the level specified is unsupported, then the **t_optmgmt()** request will fail with **TBADOPT**. If the error is detected, some options have possibly been successfully negotiated. The transport user can check the current status by calling **t_optmgmt()** with the **T_CURRENT** flag set.

The **flags** field of **req** can specify one of the following actions:

**T_NEGOTIATE** This action enables the transport user to negotiate option values.

The user specifies the options of interest and their values in the buffer.
specified by `req->opt.buf` and `req->opt.len`. The negotiated option values are returned in the buffer pointed to by `ret->opt.buf`. The `status` field of each returned option is set to indicate the result of the negotiation. The value is `T_SUCCESS` if the proposed value was negotiated, `T_PARTSUCCESS` if a degraded value was negotiated, `T_FAILURE` if the negotiation failed (according to the negotiation rules), `T_NOTSUPPORT` if the transport provider does not support this option or illegally requests negotiation of a privileged option, and `T_READONLY` if modification of a read-only option was requested. If the status is `T_SUCCESS`, `T_FAILURE`, `T_NOTSUPPORT`, or `T_READONLY`, the returned option value is the same as the one requested on input.

The overall result of the negotiation is returned in `ret->flags`

This field contains the worst single result, whereby the rating is done according to the order `T_NOTSUPPORT`, `T_READONLY`, `T_FAILURE`, `T_PARTSUCCESS`, `T_SUCCESS`. The value `T_NOTSUPPORT` is the worst result and `T_SUCCESS` is the best.

For each level, the option `T_ALLOPT` (see below) can be requested on input. No value is given with this option; only the `t_opthdr` part is specified. This input requests to negotiate all supported options of this level to their default values. The result is returned option by option in `ret->opt.buf`. (Note that depending on the state of the transport endpoint, not all requests to negotiate the default value may be successful.)

**T_CHECK**

This action enables the user to verify whether the options specified in `req` are supported by the transport provider.

If an option is specified with no option value (it consists only of a `t_opthdr` structure), the option is returned with its `status` field set to `T_SUCCESS` if it is supported, `T_NOTSUPPORT` if it is not or needs additional user privileges, and `T_READONLY` if it is read-only (in the current XTI state). No option value is returned.

If an option is specified with an option value, the `status` field of the returned option has the same value, as if the user had tried to negotiate this value with `T_NEGOTIATE`. If the status is `T_SUCCESS`, `T_FAILURE`, `T_NOTSUPPORT`, or `T_READONLY`, the returned option value is the same as the one requested on input.

The overall result of the option checks is returned in `ret->flags`. This field contains the worst single result of the option checks, whereby the rating is the same as for `T_NEGOTIATE`.

Note that no negotiation takes place. All currently effective option values remain unchanged.

**T_DEFAULT**

This action enables the transport user to retrieve the default option values. The user specifies the options of interest in `req->opt.buf`. The option values are irrelevant and will be ignored; it is sufficient to specify the `t_opthdr` part of an option only. The default values are then

modified 10 Feb 1997
SunOS 5.6
3N-1683
The status field returned is T_NOTSUPPORT if the protocol level does not support this option or the transport user illegally requested a privileged option, T_READONLY if the option is read-only, and set to T_SUCCESS in all other cases. The overall result of the request is returned in ret→flags. This field contains the worst single result, whereby the rating is the same as for T_NEGOTIATE.

For each level, the option T_ALLOPT (see below) can be requested on input. All supported options of this level with their default values are then returned. In this case, ret→opt.maxlen must be given at least the value info→options before a call to t_optmgmt() (see t_getinfo(3N) or t_open(3N)).

**T_CURRENT**

This action enables the transport user to retrieve the currently effective option values. The user specifies the options of interest in req→opt.buf. The option values are irrelevant and will be ignored; it is sufficient to specify the t_opthdr part of an option only. The currently effective values are then returned in ret→opt.buf.

The status field returned is T_NOTSUPPORT if the protocol level does not support this option or the transport user illegally requested a privileged option, T_READONLY if the option is read-only, and set to T_SUCCESS in all other cases. The overall result of the request is returned in ret→flags. This field contains the worst single result, whereby the rating is the same as for T_NEGOTIATE.

For each level, the option T_ALLOPT (see below) can be requested on input. All supported options of this level with their currently effective values are then returned.

The option T_ALLOPT can only be used with t_optmgmt() and the actions T_NEGOTIATE, T_DEFAULT, and T_CURRENT. It can be used with any supported level and addresses all supported options of this level. The option has no value; it consists of a t_opthdr only. Since in a t_optmgmt() call only options of one level may be addressed, this option should not be requested together with other options. The function returns as soon as this option has been processed.

Options are independently processed in the order they appear in the input option buffer. If an option is multiply input, it depends on the implementation whether it is multiply output or whether it is returned only once.

Transport providers may not be able to provide an interface capable of supporting T_NEGOTIATE and/or T_CHECK functionalities. When this is the case, the error TNOTSUPPORT is returned.

The function t_optmgmt() may block under various circumstances and depending on the implementation. The function will block, for instance, if the protocol addressed by the call resides on a separate controller. It may also block due to flow control constraints; that is, if data sent previously across this transport endpoint has not yet been fully processed. If the function is interrupted by a signal, the option negotiations that have been
done so far may remain valid. The behaviour of the function is not changed if
O_NONBLOCK is set.

**VALID STATES**
Legitimate states (see `t_getstate(3N)` for a call to this routine are every one except
T_UNINIT.

**RETURN VALUES**
`t_optmgmt()` returns:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>On success.</td>
</tr>
<tr>
<td>−1</td>
<td>On failure.</td>
</tr>
</tbody>
</table>

On failure, `t_errno` is set to indicate the error, and possibly `errno` is set.

**ERRORS**
On failure, `t_errno` is set to one of the following:

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TBADF</td>
<td>The specified file descriptor does not refer to a transport endpoint.</td>
</tr>
<tr>
<td>TBADFLAG</td>
<td>An invalid flag was specified.</td>
</tr>
<tr>
<td>TBADOPT</td>
<td>The specified protocol options were in an incorrect format or contained illegal information.</td>
</tr>
<tr>
<td>TBUFOVFLW</td>
<td>The number of bytes allowed (<code>maxlen</code>) for an incoming argument is greater than 0 but still insufficient to store the value of that argument. The information to be returned in <code>ret</code> will be discarded.</td>
</tr>
<tr>
<td>TNOTSUPPORT</td>
<td>This action is not supported by the transport provider.</td>
</tr>
<tr>
<td>TOUTSTATE</td>
<td>The function was issued in the wrong sequence. the communications endpoint referenced by <code>fd</code> or <code>resfd</code> is not in one of the states in which a call to this function is valid.</td>
</tr>
<tr>
<td>TPROTO</td>
<td>This error indicates that a communication problem has been detected between XTI and the transport provider for which there is no other suitable XTI <code>t_errno</code> value.</td>
</tr>
<tr>
<td>TSYSErrR</td>
<td>A system error has occurred during execution of this function; <code>errno</code> will be set to the specific error.</td>
</tr>
</tbody>
</table>

**Warning**
Using OPT_NEXTHDR, and not using knowledge of 32-bit boundary layouts, is recommended for applications. Future versions of LP64 architectures will provide more macros. This area of specification is evolving. For maximal portability, using only one option in a buffer and multiple `t_optmgmt()` calls is recommended, instead of packing multiple option requests in a buffer.

**TLI COMpatibility**
The XTI and TLI interface definitions have common names but use different header files. This, and other semantic differences between the two interfaces are described in the subsections below.

**Interface Header**
The XTI interfaces use the header file, `xti.h`. TLI interfaces should not use this header. They should use the header:

---

modified 10 Feb 1997

SunOS 5.6

3N-1685
#include <tiuser.h>

### Error Description Values

The `t_errno` value that can be set by the XTI interface and cannot be set by the TLI interface is:

- **TPROTO**

The `t_errno` values that this routine can return under different circumstances than its XTI counterpart are **TACCES** and **TBUFOVFLW**.

- **TACCES** can be returned to indicate that the user does not have permission to negotiate the specified options.
- **TBUFOVFLW** can be returned even when the `maxlen` field of the corresponding buffer has been set to zero.

### Option Buffers

The format of the options in an `opt` buffer is dictated by the transport provider. Unlike the XTI interface, the TLI interface does not fix the buffer format.

### Actions

The semantic meaning of various action values for the `flags` field of `req` differs between the TLI and XTI interfaces. TLI interface users should heed the following descriptions of the actions:

- **T_NEGOTIATE** This action enables the user to negotiate the values of the options specified in `req` with the transport provider. The provider will evaluate the requested options and negotiate the values, returning the negotiated values through `ret`.

- **T_CHECK** This action enables the user to verify whether the options specified in `req` are supported by the transport provider. On return, the `flags` field of `ret` will have either **T_SUCCESS** or **T_FAILURE** set to indicate to the user whether the options are supported. These flags are only meaningful for the **T_CHECK** request.

- **T_DEFAULT** This action enables a user to retrieve the default options supported by the transport provider into the `opt` field of `ret`. In `req`, the `len` field of `opt` must be zero and the `buf` field may be `NULL`.

### Connectionlessness

If issued as part of the connectionless-mode service, `t_optmgmt()` may block due to flow control constraints. The function will not complete until the transport provider has processed all previously sent data units.

For more information refer to the *Transport Interfaces Programming Guide*.

### ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

---

3N-1686 SunOS 5.6 modified 10 Feb 1997
SEE ALSO

t_accept(3N), t_alloc(3N), t_bind(3N), t_connect(3N), t_getinfo(3N), t_getstate(3N),
t_listen(3N), t_open(3N), t_rcvconnect(3N), attributes(5)

Transport Interfaces Programming Guide
NAME  _toupper – transliterate lower-case characters to upper-case

SYNOPSIS  #include <ctype.h>
            int _toupper(int c);

DESCRIPTION  The _toupper() macro is equivalent to toupper(3C) except that the argument c must be a lower-case letter.

RETURN VALUES  On successful completion, _toupper() returns the upper-case letter corresponding to the argument passed.

ERRORS  No errors are defined.

ATTRIBUTES  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
<tr>
<td>CSI</td>
<td>Enabled</td>
</tr>
</tbody>
</table>

SEE ALSO  islower(3C), toupper(3C), attributes(5)
NAME
toupper – transliterate lower-case characters to upper-case

SYNOPSIS
#include <ctype.h>
int toupper(int c);

DESCRIPTION
The toupper() function has as a domain a type int, the value of which is representable as an
unsigned char or the value of EOF. If the argument has any other value, the argument is returned unchanged. If the argument of toupper() represents a lower-case letter, and there exists a corresponding upper-case letter (as defined by character type information in the program locale category LC_CTYPE), the result is the corresponding upper-case letter. All other arguments in the domain are returned unchanged.

RETURN VALUES
On successful completion, toupper() returns the upper-case letter corresponding to the argument passed.

ERRORS
No errors are defined.

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
<tr>
<td>CSI</td>
<td>Enabled</td>
</tr>
</tbody>
</table>

SEE ALSO
toupper(3C), setlocale(3C), attributes(5)
NAME       towctrans – wide-character mapping

SYNOPSIS   #include <wctype.h>
            wint_t towctrans(wint_t wc, wctrans_t desc);

DESCRIPTION The towctrans() function maps the wide character wc using the mapping described by desc. The current setting of the LC_CTYPE category shall be the same as during the call to wctrans() that returned the value desc.

            towctrans(wc, wctrans("tolower")) behaves the same as towlower(wc).
            towctrans(wc, wctrans("toupper")) behaves the same as towupper(wc).

RETURN VALUES The towctrans() function returns the mapped value of wc, using the mapping described by desc; otherwise, it returns wc unchanged.

ATTRIBUTES See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe with exceptions</td>
</tr>
<tr>
<td>CSI</td>
<td>Enabled</td>
</tr>
</tbody>
</table>

SEE ALSO       setlocale(3C), wctrans(3C), attributes(5)
NAME
towlower – transliterate upper-case wide-character code to lower-case

SYNOPSIS
#include <wchar.h>

wint_t towlower(wint_t wc);

DESCRIPTION The towlower() function has as a domain a type wint_t, the value of which must be a
character representable as a wchar_t, and must be a wide-character code corresponding
to a valid character in the current locale or the value of WEOF. If the argument has any
other value, the argument is returned unchanged. If the argument of towlower() represents an upper-case wide-character code, and there exists a corresponding lower-
case wide-character code (as defined by character type information in the program locale
category LC_CTYPE), the result is the corresponding lower-case wide-character code. All
other arguments in the domain are returned unchanged.

RETURN VALUES On successful completion, towlower() returns the lower-case letter corresponding to the
argument passed. Otherwise, it returns the argument unchanged.

ERRORS No errors are defined.

ATTRIBUTES See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
<tr>
<td>CSI</td>
<td>Enabled</td>
</tr>
</tbody>
</table>

SEE ALSO iswalpha(3C), setlocale(3C), towupper(3C), attributes(5)
NAME
towupper – transliterate lower-case wide-character code to upper-case

SYNOPSIS
#include <wchar.h>

wint_t towupper(wint_t wc);

DESCRIPTION
The towupper() function has as a domain a type wint_t, the value of which must be a character representable as a wchar_t, and must be a wide-character code corresponding to a valid character in the current locale or the value of WEOF. If the argument has any other value, the argument is returned unchanged. If the argument of towupper() represents a lower-case wide-character code (as defined by character type information in the program locale category LC_CTYPE), the result is the corresponding upper-case wide-character code. All other arguments in the domain are returned unchanged.

RETURN VALUES
Upon successful completion, towupper() returns the upper-case letter corresponding to the argument passed. Otherwise, it returns the argument unchanged.

ERRORS
No errors are defined.

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
<tr>
<td>CSI</td>
<td>Enabled</td>
</tr>
</tbody>
</table>

SEE ALSO
iswalpha(3C), setlocale(3C), tolower(3C), attributes(5)
NAME  tracing – overview of tnf tracing system

DESCRIPTION  tnf tracing is a set of programs and API’s that can be used to present a high-level view of the performance of an executable, a library, or part of the kernel. tracing is used to analyze a program’s performance and identify the conditions that produced a bug.

The core elements of tracing are:

TNF_PROBE_∗()  The TNF_PROBE_∗() macros define ”probes” to be placed in code which, when enabled and executed, cause information to be added to a trace file. See TNF_PROBE(3X). If there are insufficient TNF_PROBE_∗ macros to store all the data of interest for a probe, data may be grouped into records. See TNF_DECLARE_RECORD(3X).

prex  Displays and controls probes in running software. See prex(1).

kernel probes  A set of probes built into the Solaris kernel which capture information about system calls, multithreading, page faults, swapping, memory management, and I/O. You can use these probes to obtain detailed traces of kernel activity under your application workloads. See tnf_kernel_probes(4).

tnfxtract  A program that extracts the trace data from the kernel’s in-memory buffer into a file. See tnfextract(1).

tnfdump  A program that displays the information from a trace file. See tnfdump(1).

libtnfctl  A library of interfaces that controls probes in a process. See libtnfctl(3X). prex(1) also utilizes this library. Other tools and processes use the libtnfctl interfaces to exercise fine control over their own probes.

tnf_process_enable()  A routine called by a process to turn on tracing and probe functions for the current process. See tnf_process_enable(3X).

tnf_process_disable()  A routine called by a process to turn off tracing and probe functions for the current process. See tnf_process_disable(3X).

tnf_thread_enable()  A routine called by a process to turn on tracing and probe functions for the currently running thread. See tnf_thread_enable(3X).

tnf_thread_disable()  A routine called by a process to turn off tracing and probe functions for the currently running thread. See tnf_thread_disable(3X).

EXAMPLES  The two examples shown here illustrate tracing within a process and within the kernel.
Tracing a Process

The following function in some daemon process accepts job requests of various types, queueing them for later execution. There are two "debug probes" and one "production probe." Note that probes which are intended for debugging will not be compiled into the final version of the code; however, production probes are compiled into the final product.

```c
/*
 * To compile in all probes (for development):
 *   cc -DTNF_DEBUG ...
 *
 * To compile in only production probes (for release):
 *   cc ...
 *
 * To compile in no probes at all:
 *   cc -DNPROBE ...
 */

#include <tnf/probe.h>

void work(long, char *);
enum work_request_type { READ, WRITE, ERASE, UPDATE };
static char *work_request_name[] = {"read", "write", "erase", "update"};

main()
{
  long i;
  for (i = READ; i <= UPDATE; i++)
    work(i, work_request_name[i]);
}

void work(long request_type, char *request_name)
{

  static long q_length;

  TNF_PROBE_2_DEBUG(work_start, "work",
    "XYZ%debug 'in function work'",
    tnf_long, request_type_arg, request_type,
    tnf_string, request_name_arg, request_name);

  /* assume work request is queued for later processing */
  q_length++;

  TNF_PROBE_1(work_queue, "work queue",
    "XYZ%work_load heavy",
    tnf_long, queue_length, q_length);
```

3X-1694 SunOS 5.6 modified 4 Mar 1997
The production probe "work_queue," which remains compiled in the code, will, when enabled, log the length of the work queue each time a request is received.

The debug probes "work_start" and "work_end," which are compiled only during the development phase, track entry to and exit from the `work()` function and measure how much time is spent executing it. Additionally, the debug probe "work_start" logs the value of the two incoming arguments `request_type` and `request_name`. The runtime overhead for disabled probes is low enough that one can liberally embed them in the code with little impact on performance.

For debugging, the developer would compile with `-DTNF_DEBUG`, run the program under control of `prex(1)`, enable the probes of interest (in this case, all probes), continue the program until exit, and dump the trace file:

```
% cc -DTNF_DEBUG -o daemon daemon.c # compile in all probes
% prex daemon # run program under prex control
```

```
Target process stopped
Type "continue" to resume the target, "help" for help ...
prex> list probes $all # list all probes in program
<probe list output here>
prex> enable $all # enable all probes
prex> continue # let target process execute
<program output here>
prex: target process finished
% ls /tmp/trace-* # trace output is in trace-<pid>
/tmp/trace-4194
% tnfdump /tmp/trace-4194 # get ascii output of trace file
<trace records output here>
```

For the production version of the system, the developer simply compiles without `-DTNF_DEBUG`.

### Tracing the Kernel

Kernel tracing is similar to tracing a process; however, there are some differences. For instance, to trace the kernel, you need superuser privileges. The following example uses `prex(1)` and traces the probes in the kernel that capture system call information.

**Allocate kernel trace buffer and capture trace data:**

```
root# prex -k
Type "help" for help ...
prex> buffer alloc 2m # allocate kernel trace buffer
Buffer of size 2097152 bytes allocated
prex> list probes $all # list all kernel probes
<probe list output here>
prex> list probes syscall # list syscall probes
```

---

modified 4 Mar 1997  
SunOS 5.6  
3X-1695
<syscall probes list output here>  # (keys=syscall)
prex> enable syscall  # enable only syscall probes
prex> ktrace on  # turn on kernel tracing

<Run your application in another window at this point>

prex> ktrace off  # turn off kernel tracing
prex> quit  # exit prex

Extract the kernel’s trace buffer into a file:

root# tnfxtract /tmp/ktrace  # extract kernel trace buffer

Reset kernel tracing:

root# prex -k
prex> disable $all  # disable all probes
prex> untrace $all  # untrace all probes
prex> buffer dealloc  # deallocate kernel trace buffer
prex> quit

CAUTION: Do not deallocate the trace buffer until you have extracted it into a trace file. Otherwise, you will lose the trace data that you collected from your experiment!

Examine the kernel trace file:

root# tnfdump /tmp/ktrace  # get ascii dump of trace file

prex can also attach to a running process, list probes, and perform a variety of other tasks. For more detailed examples and a more thorough discussion of tracing under Solaris, see the chapter entitled "Tracing Program Execution with the TNF Utilities" in the Programming Utilities Guide.

ATTRIBUTES  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Availability</td>
<td>SUNWtnfd</td>
</tr>
<tr>
<td>MT Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

3X-1696 SunOS 5.6 modified 4 Mar 1997
SEE ALSO

prex(1), tnfdump(1), tnfxtract(1), TNF_DECLARE_RECORD(3X), TNF_PROBE(3X),
libtnfctl(3X), tnf_process_disable(3X), tnf_kernel_probes(4), attributes(5)

Programming Utilities Guide
NAME
t_rcv – receive data or expedited data sent over a connection

SYNOPSIS
cc [flag ...] file ... -lnsl [library ...]
#include <xti.h>
int t_rcv(int fd, void *buf, unsigned int nbytes, int *flags);

DESCRIPTION
This routine is part of the XTI interfaces which evolved from the TLI interfaces. XTI
represents the future evolution of these interfaces. However, TLI interfaces are supported
for compatibility. When using a TLI routine that has the same name as an XTI routine, a
different header file, tiuser.h, must be used. Refer to the section, TLI COMPATIBILITY,
for a description of differences between the two interfaces.

This function receives either normal or expedited data. fd identifies the local transport
endpoint through which data will arrive, buf points to a receive buffer where user data
will be placed, and nbytes specifies the size of the receive buffer. flags may be set on
return from t_rcv() and specifies optional flags as described below.

By default, t_rcv() operates in synchronous mode and will wait for data to arrive if none
is currently available. However, if O_NONBLOCK is set using t_open(3N) or fcntl(),
t_rcv() will execute in asynchronous mode and will fail if no data is available. (See
TNO-DATA below.)

On return from the call, T_MORE may be set in flags. This indicates that there is more
data and the current transport service data unit (TSDU) or expedited transport service
data unit (ETSDU) awaiting reception in multiple t_rcv() calls.

In the asynchronous mode, or under unusual conditions (for example, the arrival of a sig-
nal or T_EXDATA event), the T_MORE flag may be set on return from the t_rcv() call even
when the number of bytes received is less than the size of the receive buffer specified.

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 T_MORE not set. If the transport provider does not support the concept
of a TSDU as indicated in the info argument on return from t_open() or t_getinfo(3N), the
T_MORE flag is not meaningful and should be ignored.

If nbytes is greater than zero on the call to t_rcv(), t_rcv() will return 0 only if the end of a
TSDU is being returned to the user.

On return, the data is expedited data if T_EXPEDITED is set in flags.

If T_MORE is also set, it indicates that the number of expedited bytes exceeded nbytes,
that a signal has interrupted the call, or that an entire ETSDU was not available (only for
transport protocols that support fragmentation of ETSDUs). The rest of the ETSDU will
be returned by subsequent calls to t_rcv() which will return with T_EXPEDITED set in
flags.

The end of the ETSDU is identified by the return of a t_rcv() call with T_EXPEDITED set
and T_MORE flag cleared.
If the entire ETSDU is not available, it is possible for normal data fragments to be returned between the initial and final fragments of an ETSDU.

If a signal arrives, \texttt{t_rcv()} returns, giving the user any data currently available. If no data is available, \texttt{t_rcv()} returns \(-1\), sets \texttt{t_errno} to TSYSERR and \texttt{errno} to EINTR. If some data is available, \texttt{t_rcv()} returns the number of bytes received and T_MORE is set in flags.

In synchronous mode, the only way for the user to be notified of the arrival of normal or expedited data is to issue this function or check for the T_DATA or T_EXDATA events using the \texttt{t_look}(3N) function. Additionally, the process can arrange to be notified through the EM interface, possibly through \texttt{poll}(2).

### VALID STATES

Legitimate states (see \texttt{t_getstate}(3N)) for a call to this routine are:

- T_DATAXFER
- T_OUTREL

### RETURN VALUES

\texttt{t_rcv()} \texttt{t_accept} returns:

- **Number of Bytes Received**
  - On success.
  - \(-1\) on failure.

On failure, \texttt{t_errno} is set to indicate the error, and possibly \texttt{errno} is set.

### ERRORS

On failure, \texttt{t_errno} is set to one of the following:

- **TBADF**
  - The specified file descriptor does not refer to a transport endpoint.

- **TLOOK**
  - An asynchronous event has occurred on this transport endpoint and requires immediate attention.

- **TNODATA**
  - O_NONBLOCK was set, but no data is currently available from the transport provider.

- **TNOTSUPPORT**
  - This function is not supported by the underlying transport provider.

- **TPROTO**
  - This error indicates that a communication problem has been detected between XTI and the transport provider for which there is no other suitable XTI \texttt{t_errno} value.

- **TSYSERR**
  - A system error has occurred during execution of this function, \texttt{errno} will be set to the specific error.

### TLI COMPATIBILITY

The XTI and TLI interface definitions have common names but use different header files. This, and other semantic differences between the two interfaces are described in the subsections below.

**Interface Header**

The XTI interfaces use the header file, \texttt{xti.h}. TLI interfaces should not use this header. They should use the header:

SunOS 5.6

3N-1699
include <tiuser.h>

Error Description

Values

The t_errno value that can be set by the XTI interface and cannot be set by the TLI interface is:

TPROTO

For more information refer to the Transport Interfaces Programming Guide.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO

fcntl(2), poll(2), t_getinfo(3N), t_getstate(3N), t_look(3N), t_open(3N), t_snd(3N), attributes(5)

Transport Interfaces Programming Guide
NAME
t_rcvconnect – receive the confirmation from a connection request

SYNOPSIS
cc [ flag ...] file ... -lnsl [ library ...]
#include <xti.h>
int t_rcvconnect(int fd, struct t_call *call);

DESCRIPTION
This routine is part of the XTI interfaces which evolved from the TLI interfaces. XTI represents the future evolution of these interfaces. However, TLI interfaces are supported for compatibility. When using a TLI routine that has the same name as an XTI routine, a different header file, tiuser.h, must be used. Refer to the section, TLI COMPATIBILITY, for a description of differences between the two interfaces.

This function enables a calling transport user to determine the status of a previously sent connection request and is used in conjunction with t_connect(3N) to establish a connection in asynchronous mode. The t_rcvconnect() function can also be used to complete a synchronous t_connect() call that was interrupted by a signal. The connection will be established on successful completion of this function.

The argument fd identifies the local transport endpoint where communication will be established, and call contains information associated with the newly established connection. The argument call points to a t_call structure which contains the following members:

struct netbuf addr;
struct netbuf opt;
struct netbuf udata;
int sequence;

The netbuf structure is described in t_connect().

In call, addr returns the protocol address associated with the responding transport endpoint, opt presents any protocol-specific options 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.

The maxlen (see netbuf in t_connect(3N)) field of each netbuf structure in call must be set before issuing this function to indicate the maximum size of those buffers. However, call can be set to a null pointer, in which case no information is passed to the user after t_rcvconnect() returns. Likewise, setting maxlen to zero for a particular netbuf structure disables the return of information for that particular netbuf buffer.

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.

If O_NONBLOCK is set (using t_open(3N) or fcntl(2)), t_rcvconnect() executes in asynchronous mode, and reduces to a poll for existing connection confirmations. If none are available, t_rcvconnect() fails and returns immediately without waiting for the connection to be established. (See TNODATA below.) In this case, t_rcvconnect() must be called again at a later time to complete the connection establishment phase and retrieve the

modified 10 Feb 1997

SunOS 5.6

3N-1701
information returned in call.

VALID STATES
The only legitimate state (see \texttt{t\_getstate(3N)}) for a call to this routine is \texttt{T\_OUTCON}.

RETURN VALUES
\texttt{t\_rcvconnect()} returns:

- 0 On success.
- \text{-1} On failure.

On failure, \texttt{t\_errno} is set to indicate the error, and possibly \texttt{errno} is set.

ERRORS
On failure, \texttt{t\_errno} will be set to one of the following:

- \texttt{TBADF} The specified file descriptor does not refer to a transport endpoint.
- \texttt{TBUFOVFLW} The number of bytes allocated for an incoming argument (\texttt{maxlen}) is greater than zero but is still insufficient to store the value of that argument. Accordingly, the connection information normally returned in \texttt{call} is discarded. The provider’s state, as seen by the user, will be changed to \texttt{T\_DATA\_XFER}.
- \texttt{TNODATA} \texttt{O\_NONBLOCK} was set, but a connection confirmation has not arrived.
- \texttt{TLOOK} An asynchronous event has occurred on this transport connection and requires immediate attention.
- \texttt{TNOTSUPPORT} This function is not supported by the underlying transport provider.
- \texttt{TOUTSTATE} The communications endpoint referenced by \texttt{fd} or \texttt{resfd} is not in one of the states in which a call to this function is valid.
- \texttt{TPROTO} This error indicates that a communication problem has been detected between XTI and the transport provider for which there is no other suitable XTI \texttt{t\_errno} value.
- \texttt{T_SYSERR} A system error has occurred during execution of this function. Accordingly, \texttt{errno} will have been set to the specific error.

TLI COMPATIBILITY
The XTI and TLI interface definitions have common names but use different header files. This, and other semantic differences between the two interfaces are described in the subsections below.

Interface Header
The XTI interfaces use the header file, \texttt{xti.h}. TLI interfaces should \textit{not} use this header. They should use the header:
\begin{verbatim}
#include <tiuser.h>
\end{verbatim}

Error Description Values
The \texttt{t\_errno} value that can be set by the XTI interface and cannot be set by the TLI interface is:

- \texttt{TPROTO}
A t_errno value that this routine can return under different circumstances than its XTI counterpart is TBUFOVFLW. It can be returned even when the maxlen field of the corresponding buffer has been set to zero.

For more information refer to the Transport Interfaces Programming Guide.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO

fcntl(2), t_accept(3N), t_alloc(3N), t_bind(3N), t_connect(3N), t_listen(3N), t_open(3N), t_optmgmt(3N), attributes(5)

Transport Interfaces Programming Guide
NAME  
t_rcvdis – retrieve information from disconnect

SYNOPSIS  
cc [ flag . . . ] file . . . -lnsl [ library . . . ]
#include <xti.h>
int t_rcvdis(int fd, struct t_discon *discon);

DESCRIPTION  
This routine is part of the XTI interfaces which evolved from the TLI interfaces. XTI represents the future evolution of these interfaces. However, TLI interfaces are supported for compatibility. When using a TLI routine that has the same name as an XTI routine, a different header file, tiuser.h, must be used. Refer to the section, TLI COMPATIBILITY, for a description of differences between the two interfaces.

This function is used to identify the cause of a disconnection, and to retrieve any user data sent with the disconnection. fd identifies the local transport endpoint where the connection existed, and discon points to a t_discon structure containing the following members:

   struct netbuf udata;
   int reason;
   int sequence;

reason specifies the reason for the disconnection through a protocol-dependent reason code, udata identifies any user data that was sent with the disconnection, and sequence may identify an outstanding connection indication with which the disconnection 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 connection indications. If a disconnection indication occurs, sequence can be used to identify which of the outstanding connection indications is associated with the disconnection.

The maxlen field of udata may be set to zero to indicate that the user does not care about incoming data (see TLI COMPATIBILITY for different TLI behavior). Furthermore, a user may not care if there is incoming data and may not need to know the value of reason or sequence. In such cases, supplying a null pointer for discon causes any user data associated with the disconnection to be discarded. However, if a user has retrieved more than one outstanding connection indication (using t_listen()) and discon is a null pointer, the user will be unable to identify with which connection indication the disconnection is associated.

VALID STATES  
Legitimate states (see t_getstate(3N)) for a call to this routine are:

T_DATAXFER
T_INCON  with outstanding connection count (ocnt) greater than zero
T_INREL
T_OUTCON
T_OUTREL
t_rcvdis returns:

0  On success.

−1  On failure.

On failure, t_errno is set to indicate the error, and possibly errno is set.

On failure, t_errno is set to one of the following:

TBADF  The specified file descriptor does not refer to a transport endpoint.

TBUFOVFLW  The number of bytes allocated for incoming data (maxlen in the udata portion of discon) is greater than 0 but not sufficient to store the data. If fd is a passive endpoint with ocnt>1, it remains in state T_INCON; otherwise, the endpoint state is set to T_IDLE.

TNODIS  No disconnection indication currently exists on the specified transport endpoint.

TNOTSUPPORT  This function is not supported by the underlying transport provider.

TOUTSTATE  The communications endpoint referenced by fd or resfd is not in one of the states in which a call to this function is valid.

TPROTO  This error indicates that a communication problem has been detected between XTI and the transport provider for which there is no other suitable XTI t_errno.

TSYSERR  A system error has occurred during execution of this function, errno will be set to the specific error.

TLI COMPATIBILITY  The XTI and TLI interface definitions have common names but use different header files. This, and other semantic differences between the two interfaces are described in the subsections below.

Interface Header  The XTI interfaces use the header file, xti.h. TLI interfaces should not use this header. They should use the header:

#include <tiuser.h>

Error Description Values  The t_errno values that can be set by the XTI interface and cannot be set by the TLI interface are:

TPROTO

TOUTSTATE

A failure return, and a t_errno value that this routine can set under different circumstances than its XTI counterpart is TBUFOVFLW. It can be returned even when the maxlen field of the corresponding buffer has been set to zero.
For more information refer to the *Transport Interfaces Programming Guide*.

**ATTRIBUTES**

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**SEE ALSO**

t_alloc(3N), t_connect(3N), t_listen(3N), t_open(3N), t_snddis(3N), attributes(5)  
*Transport Interfaces Programming Guide*
NAME  
t_rcvrel – acknowledge receipt of an orderly release indication

SYNOPSIS  
```
c [ flag ...] file ... -lnsl [ library ... ]

#include <xti.h>

int t_rcvrel(int fd);
```

DESCRIPTION  
This routine is part of the XTI interfaces which evolved from the TLI interfaces. XTI represents the future evolution of these interfaces. However, TLI interfaces are supported for compatibility. When using a TLI routine that has the same name as an XTI routine, a different header file, tiuser.h, must be used. Refer to the section, TLI COMPATIBILITY, for a description of differences between the two interfaces.

This function is used to acknowledge receipt of an orderly release indication. fd 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(3N) 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(3N) or t_getinfo(3N). Any user data that may be associated with the orderly release indication is discarded when t_rcvrel() is issued.

VALID STATES  
Legitimate states (see t_getstate(3N)) for a call to this routine are:

- T_DATAFER
- T_OUTREL

RETURN VALUES  
t_rcvrel() returns:

- 0  On success.
- −1  On failure.

On failure, t_errno is set to indicate the error, and possibly errno is set.

ERRORS  
On failure, t_errno is set to one of the following:

- TBADF  The specified file descriptor does not refer to a transport endpoint.
- TLOOK  An asynchronous event has occurred on this transport endpoint and requires immediate attention.
- TNOREL  No orderly release indication currently exists on the specified transport endpoint.
- TNOTSUPP  This function is not supported by the underlying transport provider.
- TOUTSTATE  The communications endpoint referenced by fd or resfd is not in one of the states in which a call to this function is valid.
- TPROTO  This error indicates that a communication problem has been
detected between XTI and the transport provider for which there is no other suitable XTI t_errno value.

**TSYSERR**
A system error has occurred during execution of this function; errno will be set to the specific error.

**TLI COMPATIBILITY**
The XTI and TLI interface definitions have common names but use different header files. This, and other semantic differences between the two interfaces are described in the subsections below.

**Interface Header**
The XTI interfaces use the header file, xti.h. TLI interfaces should not use this header. They should use the header:
```
#include <tiuser.h>
```

**Error Description Values**
The t_errno values that can be set by the XTI interface and cannot be set by the TLI interface are:

- **TPROTO**
- **TOUTSTATE**

For more information refer to the *Transport Interfaces Programming Guide*.

**ATTRIBUTES**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**SEE ALSO**
t_getinfo(3N), t_open(3N), t_sndrel(3N), attributes(5)

*Transport Interfaces Programming Guide*
NAME  t_rcvudata — receive a data unit

SYNOPSIS  cc [ flag ... ] file ... -lnsl [ library ... ]
            #include <xti.h>
            int t_rcvudata(int fd, struct t_unitdata *unitdata, int *flags);

DESCRIPTION  This routine is part of the XTI interfaces which evolved from the TLI interfaces. XTI
            represents the future evolution of these interfaces. However, TLI interfaces are supported
            for compatibility. When using a TLI routine that has the same name as an XTI routine, a
            different header file, tiuser.h, must be used. Refer to the section, TLI COMPATIBILITY,
            for a description of differences between the two interfaces.

            This function is used in connectionless mode to receive a data unit from another tran-
            sport user.  fd identifies the local transport endpoint through which data will be received,
            unitdata holds information associated with the received data unit, and flags is set on
            return to indicate that the complete data unit was not received.  unitdata points to a
            t_unitdata structure containing the following members:

            struct netbuf addr;
            struct netbuf opt;
            struct netbuf udata;

            (maxlen and other members of the netbuf structure are shown in t_connect(3N).)  The
            maxlen field of addr, opt, and udata must be set before issuing this function to indicate
            the maximum size of the buffer for each.  If the maxlen field of addr or opt is set to zero,
            no information is returned in the buf field of this parameter.

            On return from this call, addr specifies the protocol address of the sending user, opt
            identifies options that were associated with this data unit, and udata specifies the user
            data that was received.

            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_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.

            If the buffer defined in the udata field of unitdata is not large enough to hold the current
            data unit, the buffer will be filled and T_MORE will be set in flags on return to indicate
            that another t_rcvudata() should be issued to retrieve the rest of the data unit.  Subse-
            quent calls to t_rcvudata() will return zero for the length of the address and options until
            the full data unit has been received.  If the call is interrupted, t_rcvudata() will return
            EINTR and no datagrams will have been removed from the endpoint.

VALID STATES  The only legitimate state (see t_getstate(3N)) for a call to this routine is T_IDLE.

RETURN VALUES  t_rcvudata() returns:

            0    On success.
            −1   On failure.

modified 6 Mar 1997            SunOS 5.6            3N-1709
On failure, `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 the incoming protocol address or options (maxlen) is greater than zero but not sufficient to store the information. The unit data information to be returned in `unit_data` will be discarded.

- **TLOOK**
  An asynchronous event has occurred on this transport endpoint and requires immediate attention.

- **TNODATA**
  `O_NONBLOCK` was set, but no data units are currently available from the transport provider.

- **TNOTSUPPORT**
  This function is not supported by the underlying transport provider.

- **TOUTSTATE**
  The communications endpoint referenced by `fd` or `resfd` is not in one of the states in which a call to this function is valid.

- **TPROTO**
  This error indicates that a communication problem has been detected between XTI and the transport provider for which there is no other suitable XTI, `t_errno` value.

- **TSYSERR**
  A system error has occurred during execution of this function. `errno` will be set to the specific error.

**TLI COMPATIBILITY**

The XTI and TLI interface definitions have common names but use different header files. This, and other semantic differences between the two interfaces are described in the subsections below.

**Interface Header**

The XTI interfaces use the header file, `xti.h`. TLI interfaces should not use this header. They should use the header:

```c
#include <tiuser.h>
```

**Error Description Values**

The `t_errno` values that can be set by the XTI interface and cannot be set by the TLI interface are:

- **TPROTO**
- **TOUTSTATE**

A `t_errno` value that this routine can return under different circumstances than its XTI counterpart is **TBUFOVFLW**. It can be returned even when the `maxlen` field of the corresponding buffer has been set to zero.

**Option Buffers**

The format of the options in an `opt` buffer is dictated by the transport provider. Unlike the XTI interface, the TLI interface does not fix the buffer format.
For more information refer to the *Transport Interfaces Programming Guide*.

**ATTRIBUTES**

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**SEE ALSO**

`fcntl(2), t_connect(3N), t_getstate(3N), t_open(3N), t_rcvuderr(3N), t_sndudata(3N), attributes(5)`

*Transport Interfaces Programming Guide*
**NAME**

`t_rcvuderr` – receive a unit data error indication

**SYNOPSIS**

```c
cc [ flag ... ] file ... -lnsl [ library ... ]
#include <xti.h>
int t_rcvuderr(int fd, struct t_uderr *uderr);
```

**DESCRIPTION**

This routine is part of the XTI interfaces which evolved from the TLI interfaces. XTI represents the future evolution of these interfaces. However, TLI interfaces are supported for compatibility. When using a TLI routine that has the same name as an XTI routine, a different header file, `tiuser.h`, must be used. Refer to the section, TLI COMPATIBILITY, for a description of differences between the two interfaces.

This function is used in connectionless mode to receive information concerning an error on a previously sent data unit, and should only be issued following a unit data error indication. It informs the transport user that a data unit with a specific destination address and protocol options produced an error. `fd` identifies the local transport endpoint through which the error report will be received, and `uderr` points to a `t_uderr` structure containing the following members:

```c
struct netbuf addr;
struct netbuf opt;
long error;
```

(maxlen and other members of `netbuf` are described in `t_connect(3N)`.) The `maxlen` field of `addr` and `opt` must be set before issuing this function to indicate the maximum size of the buffer for each. If this field is set to zero for `addr` or `opt`, no information is returned in the `buf` field of this parameter.

On return from this call, the `addr` structure specifies the destination protocol address of the erroneous data unit, the `opt` structure identifies options that were associated with the data unit, and `error` specifies a protocol-dependent error code.

If the user does not care to identify the data unit that produced an error, `uderr` may be set to a null pointer and `t_rcvuderr()` will simply clear the error indication without reporting any information to the user.

**VALID STATES**

The only legitimate state (see `t_getstate(3N)`) for a call to this routine is `T_IDLE`.

**RETURN VALUES**

`t_rcvuderr()` returns:

- 0 On success.
- -1 On failure.

On failure, `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 the incoming protocol address or
Network Functions

options (maxlen) is greater than 0 but not sufficient to store the information. The unit data error information to be returned in uderr will be discarded.

TNOTSUPPORT This function is not supported by the underlying transport provider.

TNODERR No unit data error indication currently exists on the specified transport endpoint.

TOUTSTATE The communications endpoint referenced by fd or resfd is not in one of the states in which a call to this function is valid.

TPROTO This error indicates that a communication problem has been detected between XTI and the transport provider for which there is no other suitable XTI t_errno value.

TSYSERR A system error has occurred during execution of this function, errno will be set to the specific error.

TLI COMPATIBILITY

The XTI and TLI interface definitions have common names but use different header files. This, and other semantic differences between the two interfaces are described in the subsections below.

Interface Header

The XTI interfaces use the header file, xti.h. TLI interfaces should not use this header. They should use the header:

```
#include <tiuser.h>
```

Error Description Values

The t_errno values that can be set by the XTI interface and cannot be set by the TLI interface are:

TPROTO

TOUTSTATE

A t_errno value that this routine can return under different circumstances than its XTI counterpart is TBUFOVFLW. It can be returned even when the maxlen field of the corresponding buffer has been set to zero.

Option Buffers

The format of the options in an opt buffer is dictated by the transport provider. Unlike the XTI interface, the TLI interface does not fix the buffer format.

For more information refer to the *Transport Interfaces Programming Guide*.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

modified 10 Feb 1997

SunOS 5.6

3N-1713
SEE ALSO  

`t_connect(3N), t_rcvudata(3N), t_sndudata(3N), attributes(5)`

*Transport Interfaces Programming Guide*
NAME

truncate, ftruncate – set a file to a specified length

SYNOPSIS

#include <unistd.h>

int truncate(const char *path, off_t length);

int ftruncate(int fildes, off_t length);

DESCRIPTION

The **truncate()** function causes the regular file named by **path** to have a size of **length** bytes.

The **ftruncate()** function causes the regular file referenced by **fildes** to have a size of **length** bytes.

The effect of **ftruncate()** and **truncate()** on other types of files is unspecified. If the file previously was larger than **length**, the extra data is lost. If it was previously shorter than **length**, bytes between the old and new lengths are read as zeroes. With **ftruncate()**, the file must be open for writing; for **truncate()**, the process must have write permission for the file.

If the request would cause the file size to exceed the soft file size limit for the process, the request will fail and the implementation will generate the **SIGXFSZ** signal for the process.

These functions do not modify the file offset for any open file descriptions associated with the file. On successful completion, if the file size is changed, these functions will mark for update the **st_ctime** and **st_mtime** fields of the file, and if the file is a regular file, the **S_ISUID** and **S_ISGID** bits of the file mode may be cleared.

RETURN VALUES

Upon successful completion, **ftruncate()** and **truncate()** return 0. Otherwise, −1 is returned and **errno** is set to indicate the error.

ERRORS

The **ftruncate()** and **truncate()** functions will fail if:

- **EINTR** A signal was caught during execution.
- **EINVAL** The **length** argument was less than 0.
- **EFBIG** or **EINVAL** The **length** argument was greater than the maximum file size.
- **EIO** An I/O error occurred while reading from or writing to a file system.

The **truncate()** function will fail if:

- **EACCES** A component of the path prefix denies search permission, or write permission is denied on the file.
- **EFAULT** The **path** argument points outside the process’ allocated address space.
- **EINVAL** The **path** argument is not an ordinary file.
- **EISDIR** The named file is a directory.
- **ELOOP** Too many symbolic links were encountered in resolving **path**.
- **EMFILE** The maximum number of file descriptors available to the process has been reached.
Components of path require hopping to multiple remote machines and file system type does not allow it.

The length of the specified pathname exceeds PATH_MAX bytes, or the length of a component of the pathname exceeds NAME_MAX bytes.

A component of path does not name an existing file or path is an empty string.

Additional space could not be allocated for the system file table.

A component of the path prefix of path is not a directory.

The path argument points to a remote machine and the link to that machine is no longer active.

The length of the specified pathname exceeds PATH_MAX bytes, or the length of a component of the pathname exceeds NAME_MAX bytes.

The named file resides on a read-only file system.

The file exists, mandatory file/record locking is set, and there are outstanding record locks on the file (see chmod(2)).

The fildes argument is not a file descriptor open for writing.

The file is a regular file and length is greater than the offset maximum established in the open file description associated with fildes.

The fildes argument references a file that was opened without write permission.

The fildes argument does not correspond to an ordinary file.

The fildes argument points to a remote machine and the link to that machine is no longer active.

Pathname resolution of a symbolic link produced an intermediate result whose

The truncate() and ftruncate() functions have explicit 64-bit equivalents. See interface64(5).

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

chmod(2), fcntl(2), open(2), attributes(5), interface64(5)
NAME

#include <search.h>

DESCRIPTION

The tsearch(), tfind(), tdelete(), and twalk() functions are routines for manipulating binary search trees. They are generalized from Knuth (6.2.2) Algorithms T and D. 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.

tsearch() is used to build and access the tree. key is a pointer to a datum to be accessed or stored. If there is a datum in the tree equal to *key (the value pointed to by key), a pointer to this found datum is returned. Otherwise, *key is inserted, and a pointer to it returned. Only pointers are copied, so the calling routine must store the data. rootp points to a variable that points to the root of the tree. A null value for the variable pointed to by rootp 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.

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().

tdelete() deletes a node from a binary search tree. The arguments are the same as for tsearch(). The variable pointed to by rootp 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.

twalk() traverses a binary search tree. root 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.) action 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

typedef enum { preorder, postorder, endorder, leaf } VISIT;

(defined in the <search.h> 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 the tree), or whether the node is a leaf. The third argument is the level of the node in the tree.
with the root being level zero.
The pointers to the key and the root of the tree should be of type pointer-to-element, and cast to type pointer-to-character. Similarly, although declared as type pointer-to-character, the value returned should be cast into type pointer-to-element.

**RETURN VALUES**

If the node is found, both `tsearch()` and `tfind()` return a pointer to it. If not, `tfind()` returns a null pointer, and `tsearch()` returns a pointer to the inserted item.

A null pointer is returned by `tsearch()` if there is not enough space available to create a new node.

A null pointer is returned by `tsearch()`, `tfind()` and `tdelete()` if `rootp` is a null pointer on entry.

The `tdelete()` function returns a pointer to the parent of the deleted node, or a null pointer if the node is not found.

The `twalk()` function returns no value.

**ERRORS**

No errors are defined.

**USAGE**

The `root` argument to `twalk()` is one level of indirection less than the `rootp` arguments to `tsearch()` and `tdelete()`.

There are two nomenclatures used to refer to the order in which tree nodes are visited. `tsearch` uses preorder, postorder and endorder to refer respectively to visiting a node before any of its children, after its left child and before its right, and after both its children. The alternate nomenclature uses preorder, inorder and postorder to refer to the same visits, which could result in some confusion over the meaning of postorder.

If the calling function alters the pointer to the root, results are unpredictable.

**EXAMPLES**

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.

```c
#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);
}

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO bsearch(3C), hsearch(3C), lsearch(3C), attributes(5)
NAME

\texttt{t_snd} – send data or expedited data over a connection

SYNOPSIS

\begin{verbatim}
cc [ flag ...] file ... -lnsl [ library ... ]
#include <xti.h>

int \texttt{t_snd}(int \textit{fd}, void *\textit{buf}, unsigned int \textit{nbytes}, int \textit{flags});
\end{verbatim}

DESCRIPTION

This routine is part of the XTI interfaces which evolved from the TLI interfaces. XTI represents the future evolution of these interfaces. However, TLI interfaces are supported for compatibility. When using a TLI routine that has the same name as an XTI routine, a different header file, \texttt{tiuser.h}, must be used. Refer to the section, \texttt{TLI COMPATIBILITY}, for a description of differences between the two interfaces.

This function is used to send either normal or expedited data. \textit{fd} identifies the local transport endpoint over which data should be sent, \textit{buf} points to the user data, \textit{nbytes} specifies the number of bytes of user data to be sent, and \textit{flags} can be specified using bitwise-OR operations with the following values.

\begin{description}
\item[T\_EXPEDITED] Send the data as expedited data. This will be subject to the interpretations of the transport provider.
\item[T\_MORE] Send an indication to the transport provider that the transport service data unit (TSDU) or expedited transport service data unit (ETSDU) is being sent through multiple \texttt{t_snd()} calls. Each \texttt{t_snd()} with the T\_MORE flag set indicates that another \texttt{t_snd()} will follow with more data for the current TSDU or (ETSDU). The end of the TSDU (or ETSDU) is identified by a \texttt{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 \textit{info} argument on return from \texttt{t_open(3N)} or \texttt{t_getinfo(3N)}, the T\_MORE flag is not meaningful and will be ignored if set.

The sending of a zero-length fragment of a TSDU or ETSDU is only permitted where this is used to indicate the end of a TSDU or ETSDU; that is, when the T\_MORE flag is not set. Some transport providers also forbid zero-length TSDUs and ETSDUs.
\item[T\_PUSH] Tells the communication provider to flush all data that is currently in its send buffers. If not set in flags, the behaviour is protocol-specific.
\end{description}

\textbf{Note:} The communications provider is free to collect data in a send buffer until it accumulates as sufficient amount for transmission.

By default, \texttt{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 \texttt{O\_NONBLOCK} was set using \texttt{t_open()} or \texttt{fcntl()}, \texttt{t_snd()} will
execute in asynchronous mode, and will fail immediately if there are flow control restrictions.

The process can arrange to be informed when the flow control restrictions are cleared using either \texttt{t\_look()} or the event management (EM) interface, possibly through \texttt{poll()} (see \texttt{poll(2)}).

On successful completion, \texttt{t\_snd()} returns the number of octets accepted by the transport provider. Normally this will equal the number of bytes specified in \texttt{nbytes}. However, if \texttt{O\_NONBLOCK} is set or the function is interrupted by a signal, it is possible that only part of the data has actually been accepted by the transport provider. In this case, \texttt{t\_snd()} will set \texttt{T\_MORE} for the data that was sent (see below) and will return a value that is less than the value of \texttt{nbytes}.

If \texttt{t\_snd()} is interrupted by a signal before it could transfer data to the communications provider, it returns \texttt{-1} with \texttt{t\_errno} set to \texttt{TSYSERR} and \texttt{errno} set to \texttt{EINTR}.

If \texttt{nbytes} is zero and sending of zero bytes is not supported by the underlying communications service, \texttt{t\_snd()} will return \texttt{-1} with \texttt{t\_errno} set to \texttt{TBADDATA}.

The size of each \texttt{TSDU} or \texttt{ETSDU} must not exceed the limits of the transport provider as specified by the current values in the \texttt{TSDU} or \texttt{ETSDU} fields in the \texttt{info} argument returned by \texttt{t\_getinfo(\)}. The error \texttt{TLOOK} may be returned to inform the process that an event (for example, a disconnection) has occurred.

**VALID STATES**

Legitimate states (see \texttt{t\_getstate(3N)}) for a call to this routine are:

\texttt{T\_DATAXFER}

\texttt{T\_INREL}

**RETURN VALUES**

\texttt{t\_snd()} returns:

- \texttt{number of bytes accepted by the transport provider}
- \texttt{-1} On failure.

On failure, \texttt{t\_errno} is set to indicate the error, and possibly \texttt{errno} is set.

**Note:** If the number of bytes accepted by the communications provider is less than the number of bytes requested, this may either indicate that \texttt{O\_NONBLOCK} is set and the communications provider is blocked due to flow control, or that \texttt{O\_NONBLOCK} is clear and the function was interrupted by a signal.

**ERRORS**

On failure, \texttt{t\_errno} is set to one of the following:

- \texttt{TBADDATA} Illegal amount of data:
  - A single send was attempted specifying a \texttt{TSDU} (\texttt{ETSDU}) or fragment \texttt{TSDU} (\texttt{ETSDU}) greater than that specified by the
current values of the TSDU or ETSDU fields in the info argument.
— A send of a zero byte TSDU (ETSDU) or zero byte fragment of a TSDU (ETSDU) is not supported by provider.
— Multiple sends were attempted resulting in a TSDU (ETSDU) larger than that specified by the current value of the TSDU or ETSDU fields in the info argument – the ability of an XTI implementation to detect such an error case is implementation-dependent (see NOTES).

TBADF  The specified file descriptor does not refer to a transport endpoint.
TBADFLAG An invalid flag was specified.
TFLOW O_NONBLOCK was set, but the flow control mechanism prevented the transport provider from accepting any data at this time.
TLOOK An asynchronous event has occurred on this transport endpoint.
TNOTSUPPORT This function is not supported by the underlying transport provider.
TOUTSTATE The communications endpoint referenced by fd or resfd is not in one of the states in which a call to this function is valid.
TPROTO This error indicates that a communication problem has been detected between XTI and the transport provider for which there is no other suitable XTI t_errno value.
TSYSERR A system error (see intro(2)) has occurred during execution of this function.

TLI COMPATIBILITY

The XTI and TLI interface definitions have common names but use different header files. This, and other semantic differences between the two interfaces are described in the subsections below.

Interface Header
The XTI interfaces use the header file, xti.h. TLI interfaces should not use this header. They should use the header:
#include <tiuser.h>

Error Description Values
The t_errno values that can be set by the XTI interface and cannot be set by the TLI interface are:

TPROTO
TLOOK
TBADFLAG
TOUTSTATE

The t_errno values that this routine can return under different circumstances than its XTI counterpart are:

TBADDATA

In the TBADDATA error cases described above, TBADDATA is

SunOS 5.6

modified 10 Feb 1997
Network Functions

returned, only for illegal zero byte TSDU (ETSDU) send attempts.
For more information refer to the Transport Interfaces Programming Guide.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO

fcntl(2), t_getinfo(3N), t_open(3N), t_rcv(3N), attributes(5)

Transport Interfaces Programming Guide

NOTES

It is important to remember that the transport provider treats all users of a transport endpoint as a single user. Therefore if several processes issue concurrent t_snd() calls then the different data may be intermixed.

Multiple sends which exceed the maximum TSDU or ETSDU size may not be discovered by XTI. In this case an implementation-dependent error will result (generated by the transport provider) perhaps on a subsequent XTI call. This error may take the form of a connection abort, a TSYSERR, a TBADDATA or a TPROTO error.

If multiple sends which exceed the maximum TSDU or ETSDU size are detected by XTI, t_snd() fails with TBADDATA.

modified 10 Feb 1997

SunOS 5.6

3N-1723
NAME t_snddis – send user-initiated disconnection request

SYNOPSIS cc [ flag ... ] file ... -lnsl [ library ... ]
#include <xti.h>
int t_snddis(int fd, const struct t_call  *call);

DESCRIPTION This routine is part of the XTI interfaces which evolved from the TLI interfaces. XTI represents the future evolution of these interfaces. However, TLI interfaces are supported for compatibility. When using a TLI routine that has the same name as an XTI routine, a different header file, tiuser.h, must be used. Refer to the section, TLI COMPATIBILITY, for a description of differences between the two interfaces.

This function is used to initiate an abortive release on an already established connection or to reject a connection request. fd identifies the local transport endpoint of the connection, and call specifies information associated with the abortive release. call points to a t_call structure, which contains the following members:

struct netbuf addr;
struct netbuf opt;
struct netbuf udata;
int sequence;

netbuf is described in t_connect(3N). The values in call have different semantics, depending on the context of the call to t_snddis(). When rejecting a connection request, call must be non-null pointer and must contain a valid value of sequence to uniquely identify the rejected connection indication to the transport provider. The sequence field is only meaningful if the transport connection is in the T_INCON state.

The addr and opt fields of call are ignored. In all other cases, call need only be used when data is being sent with the disconnection 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 call may be a null pointer.

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 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 remote user.

VALID STATES Legitimate states (see t_getstate(3N)) for a call to this routine are:
T_DATAXFER
T_INCON with outstanding connection count (ocnt) greater than zero
T_INREL
T_OUTCON
T_OUTREL

3N-1724 SunOS 5.6 modified 10 Feb 1997
RETURN VALUES

\texttt{t_snddis()} returns:

- 0 On success.
- -1 On failure.

On failure, \texttt{t_errno} is set to indicate the error, and possibly \texttt{errno} is set.

ERRORS

On failure, \texttt{t_errno} is set to one of the following:

- **TBADDATA**: The amount of user data specified was not within the bounds allowed by the transport provider.
- **TBADF**: The specified file descriptor does not refer to a transport endpoint.
- **TBADSEQ**: An invalid sequence number was specified, or a null call pointer was specified when rejecting a connection request.
- **TLOOK**: An asynchronous event has occurred on this transport endpoint, for which handling is required before any progress can be made.
- **TNOTSUPPORT**: This function is not supported by the underlying transport provider.
- **TOUTSTATE**: The communications endpoint referenced by \texttt{fd} or \texttt{resfd} is not in one of the states in which a call to this function is valid.
- **TPROTO**: This error indicates that a communication problem has been detected between XTI and the transport provider for which there is no other suitable XTI \texttt{t_errno} value.
- **TSYSERR**: A system error has occurred during execution of this function, \texttt{errno} will be set to the specific error.

TLI COMPATIBILITY

The XTI and TLI interface definitions have common names but use different header files. This, and other semantic differences between the two interfaces are described in the subsections below.

Interface Header

The XTI interfaces use the header file, \texttt{xti.h}. TLI interfaces should \texttt{not} use this header. They should use the header:

\begin{verbatim}
#include <tiuser.h>
\end{verbatim}

Error Description Values

The \texttt{t_errno} value that can be set by the XTI interface and cannot be set by the TLI interface is:

\begin{verbatim}
TPROTO
\end{verbatim}

Option Buffers

The format of the options in an \texttt{opt} buffer is dictated by the transport provider. Unlike the XTI interface, the TLI interface does not fix the buffer format.

modified 10 Feb 1997

SunOS 5.6

3N-1725
For more information refer to the *Transport Interfaces Programming Guide*.

**ATTRIBUTES**

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**SEE ALSO**

`t_connect(3N), t_getinfo(3N), t_listen(3N), t_open(3N), attributes(5)`

*Transport Interfaces Programming Guide*

**NOTES**

`t_snddis()` is an abortive disconnection. Therefore a `t_snddis()` issued on a connection endpoint may cause data previously sent using `t_snd()`, or data not yet received, to be lost (even if an error is returned).
NAME  t_sndrel – initiate an orderly release

SYNOPSIS  
```
c[flag ...] file ... -lnsl [ library ... ]
#include <xti.h>
int t_sndrel(int fd);
```

DESCRIPTION  This routine is part of the XTI interfaces which evolved from the TLI interfaces. XTI represents the future evolution of these interfaces. However, TLI interfaces are supported for compatibility. When using a TLI routine that has the same name as an XTI routine, a different header file, `tiuser.h`, must be used. Refer to the section, TLI COMPATIBILITY, for a description of differences between the two interfaces.

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. `fd` 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.

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(3N)` or `t_getinfo(3N)`.

VALID STATES  Legitimate states (see `t_getstate(3N)` ) for a call to this routine are:
```
T_DATAxFER
T_INREL
```

RETURN VALUES  `t_sndrel()` returns:
```
0   On success.
-1  On failure.
```

On failure, `t_errno` is set to indicate the error, and possibly `errno` is set.

ERRORS  On failure, `t_errno` is set to one of the following:
```
TBADF  The specified file descriptor does not refer to a transport endpoint.
TFLOW  O_NONBLOCK was set, but the flow control mechanism prevented the transport provider from accepting the function at this time.
TLOOK  An asynchronous event has occurred on this transport endpoint and requires immediate attention.
TNOTSUPPORT  This function is not supported by the underlying transport provider.
TOUTSTATE  The communications endpoint referenced by `fd` or `resfd` is not in one of the states in which a call to this function is valid.
TPROTO  This error indicates that a communication problem has been detected between XTI and the transport provider for which there is no other suitable XTI `t_errno` value.
```

modified 10 Feb 1997  SunOS 5.6  3N-1727
A system error has occurred during execution of this function; 
`errno` will be set to the specific error.

**TLI COMPATIBILITY**

The XTI and TLI interface definitions have common names but use different header files. 
This, and other semantic differences between the two interfaces are described in the subsections below.

**Interface Header**

The XTI interfaces use the header file, `xti.h`. TLI interfaces should not use this header. 
They should use the header:

```c
#include <tiuser.h>
```

**Error Description**

The `t_errno` values that can be set by the XTI interface and cannot be set by the TLI interface are:

- TPROTO
- TLOOK
- TOUTSTATE

**Notes**

Whenever this function fails with `t_errno` set to `TFLOW`, `O_NONBLOCK` must have been set.

For more information refer to the *Transport Interfaces Programming Guide*.

### ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### SEE ALSO

`t_getinfo(3N)`, `t_open(3N)`, `t_rcvrel(3N)`, `attributes(5)`

*Transport Interfaces Programming Guide*
Network Functions  t_sndudata (3N)

NAME  t_sndudata – send a data unit

SYNOPSIS  cc [ flag ... ] file ... -lnsl [ library ... ]

#include <xti.h>

int t_sndudata(int fd, const struct t_unitdata *unitdata);

DESCRIPTION  This routine is part of the XTI interfaces which evolved from the TLI interfaces. XTI represents the future evolution of these interfaces. However, TLI interfaces are supported for compatibility. When using a TLI routine that has the same name as an XTI routine, a different header file, tiuser.h, must be used. Refer to the section, TLI COMPATIBILITY, for a description of differences between the two interfaces.

This function is used in connectionless mode to send a data unit to another transport user. fd identifies the local transport endpoint through which data will be sent, and unitdata points to a t_unitdata structure containing the following members:

- struct netbuf addr;
- struct netbuf opt;
- struct netbuf udata;

netbuf is described in t_connect(3N). In unitdata, addr specifies the protocol address of the destination user, opt identifies 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 uses the option values currently set for the communications endpoint.

If the len field of udata is zero, and the sending of zero octets is not supported by the underlying transport service, t_sndudata() will return −1 with t_errno set to TBADDATA.

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_NONBLOCK is set using t_open(3N) or fcntl(2), t_sndudata() will execute in asynchronous mode and will fail under such conditions.

The process can arrange to be notified of the clearance of a flow control restriction using either t_look(3N) or the EM interface, possibly through poll(2).

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 info argument of t_open() or t_getinfo(), a TBADDATA error will be generated. If t_sndudata() is called before the destination user has activated its transport endpoint (see t_bind(3N)), the data unit may be discarded.

If it is not possible for the transport provider to immediately detect the conditions that cause the errors TBADDADDR and TBADOPT, these errors will alternatively be returned by t_rcvuderr(3N). Therefore, an application must be prepared to receive these errors in both of these ways.

modified 10 Feb 1997  SunOS 5.6  3N-1729
If the call is interrupted, \texttt{t_sndudata()} will return \texttt{EINTR} and the datagram will not be sent.

**VALID STATES**

The only legitimate state (see \texttt{t_getstate(3N)}) for a call to this routine is \texttt{T_IDLE}.

**RETURN VALUES**

\texttt{t_sndudata()} returns:

- \texttt{0} On success.
- \texttt{-1} On failure.

On failure, \texttt{t_errno} is set to indicate the error, and possibly \texttt{errno} is set.

**ERRORS**

On failure, \texttt{t_errno} is set to one of the following:

- \texttt{TBADADDR} The specified protocol address was in an incorrect format or contained illegal information.
- \texttt{TBADDATA} Illegal amount of data. A single send was attempted specifying a TSDU greater than that specified in an earlier \texttt{info} argument (see \texttt{t_open(3N)} and \texttt{t_connect(3N)}), or a send of a zero byte TSDU is not supported by the provider.
- \texttt{TBADF} The specified file descriptor does not refer to a transport endpoint.
- \texttt{TBADOPT} The specified options were in an incorrect format or contained illegal information.
- \texttt{TFLOW} \texttt{O_NONBLOCK} was set, but the flow control mechanism prevented the transport provider from accepting data at this time.
- \texttt{TLOOK} An asynchronous event has occurred on this transport endpoint.
- \texttt{TNOTSUPPORT} This function is not supported by the underlying transport provider.
- \texttt{TOUTSTATE} The communications endpoint referenced by \texttt{fd} or \texttt{resfd} is not in one of the states in which a call to this function is valid.
- \texttt{TPROTO} This error indicates that a communication problem has been detected between XTI and the transport provider for which there is no other suitable XTI \texttt{t_errno} value.
- \texttt{TSYSERR} A system error has occurred during execution of this function, \texttt{errno} will be set to the specific error.

**TLI COMPATIBILITY**

The XTI and TLI interface definitions have common names but use different header files. This, and other semantic differences between the two interfaces are described in the subsections below.

**Interface Header**

The XTI interfaces use the header file, \texttt{xti.h}. TLI interfaces should not use this header. They should use the header:

\texttt{#include <tiuser.h>}

3N-1730 SunOS 5.6 modified 10 Feb 1997
Network Functions

t_sndudata (3N)

Error Description

The `t_errno` values that can be set by the XTI interface and cannot be set by the TLI interface are:

- TPROTO
- TBADADDR
- TBADOPT
- TLOOK
- TOUTSTATE

Notes

Whenever this function fails with `t_errno` set to `TFLOW O_NONBLOCK` must have been set.

Option Buffers

The format of the options in an `opt` buffer is dictated by the transport provider. Unlike the XTI interface, the TLI interface does not fix the buffer format.

For more information refer to the *Transport Interfaces Programming Guide*.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO

fcntl(2), poll(2), t_alloc(3N), t_bind(3N), t_connect(3N), t_getinfo(3N), t_look(3N), t_open(3N), t_rcvudata(3N), t_rcvuderr(3N), attributes(5)

*Transport Interfaces Programming Guide*
NAME  t_strerror – get error message string

SYNOPSIS  cc [ flag ... ] file ... -lnsl [ library ... ]
#include <xti.h>

const char *t_strerror(int errnum);

DESCRIPTION  This routine is part of the XTI interfaces which evolved from the TLI interfaces. XTI represents the future evolution of these interfaces. However, TLI interfaces are supported for compatibility. When using a TLI routine that has the same name as an XTI routine, a different header file, tiuser.h, must be used. Refer to the section, TLI COMPATIBILITY, for a description of differences between the two interfaces.

The t_strerror() function maps the supplied number (errnum) corresponding to a transport-level error to a language-specific 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 it is English, the error message string describing the value in t_errno is identical to the comments following the t_errno codes defined in xti.h. If an error code is unknown and the language is English, t_strerror() returns the string:

"<error>: error unknown"

where <error> is the error number supplied as input. In other languages, an equivalent text is provided.

VALID STATES  Legitimate states (see t_getstate(3N)) for a call to this routine are every one except T_UNINIT.

RETURN VALUES  The function t_strerror() returns a pointer to the generated message string.

TLI COMPATIBILITY  The XTI and TLI interface definitions have common names but use different header files. This, and other semantic differences between the two interfaces are described in the subsections below.

Interface Header  The XTI interfaces use the header file, xti.h. TLI interfaces should not use this header. They should use the header:

#include <tiuser.h>

For more information refer to the Transport Interfaces Programming Guide.

ATTRIBUTES  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>
SEE ALSO
ggettext(3C), perror(3C), setlocale(3C), strerror(3C), t_error(3N), attributes(5)

Transport Interfaces Programming Guide
NAME
t_sync – synchronize transport library

SYNOPSIS
c{ [flag . . .] file . . . -lnsl [ library . . . ]
#include <xti.h>
int t_sync(int fd);

DESCRIPTION
This routine is part of the XTI interfaces which evolved from the TLI interfaces. XTI
represents the future evolution of these interfaces. However, TLI interfaces are supported
for compatibility. When using a TLI routine that has the same name as an XTI routine, a
different header file, tiuser.h, must be used. Refer to the section, TLI COMPATIBILITY,
for a description of differences between the two interfaces.

For the transport endpoint specified by fd, t_sync() synchronizes the data structures
managed by the transport library with information from the underlying transport pro-
vider. In doing so, it can convert an uninitialized file descriptor (obtained using open(2),
dup(2), or as a result of a fork(2) and exec(2) ) to an initialize transport endpoint, assum-
ing that file descriptor referenced a transport provider. This function also allows two
cooperating processes to synchronize their interaction with a transport provider.

For example, if a process issues a fork() for a new process and issues an exec(), 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.

It is important to remember that the transport provider treats all users of a transport end-
point as a single user. If multiple processes are using the same endpoint, they should
coordinate their activities so as not to violate the state of the transport endpoint. t_sync()
returns the current state of the transport endpoint to the user, thereby enabling the user
to verify the state before taking further action. This coordination is only valid among
cooperating processes; it is possible that a process or an incoming event could change the
endpoint’s state after a t_sync() is issued.

If the transport endpoint is undergoing a state transition when t_sync() is called, the
function will fail.

VALID STATES
Legitimate states (see t_getstate(3N)) for a call to this routine are every one except
T_UNINIT.

RETURN VALUES
t_sync() returns:
State of The Transport Provider On success.
−1 On failure.

On failure, t_errno is set to indicate the error, and possibly errno is set.

The state returned may be one of the following:
T_UNBND unbound
T_IDLE idle
Network Functions

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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>
</tbody>
</table>

**ERRORS**

On failure, `t_errno` is set to one of the following:

- **TBADF**
  - The specified file descriptor does not refer to a transport endpoint.
  - This error may be returned when the `fd` has been previously closed or when an erroneous number may have been passed to the call.

- **TPROTO**
  - This error indicates that a communication problem has been detected between XTI and the transport provider for which there is no other suitable XTI `t_errno` value.

- **TSTATECHNG**
  - The transport endpoint is undergoing a state change.

- **TSYSERR**
  - A system error has occurred during execution of this function, `errno` will be set to the specific error.

**TLI COMPatibility**

The XTI and TLI interface definitions have common names but use different header files. This, and other semantic differences between the two interfaces are described in the subsections below.

**Interface Header**

The XTI interfaces use the header file, `xti.h`. TLI interfaces should not use this header. They should use the header:

```c
#include <tiuser.h>
```

**Error Description Values**

The `t_errno` value that can be set by the XTI interface and cannot be set by the TLI interface is:

- **TPROTO**

For more information refer to the *Transport Interfaces Programming Guide*.

**Attributes**

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

dup(2), exec(2), fork(2), open(2), attributes(5)

*Transport Interfaces Programming Guide*

modified 10 Feb 1997

SunOS 5.6

3N-1735
NAME    ttyname, ttyname_r, – find pathname of a terminal

SYNOPSIS    #include <unistd.h>
            char *ttyname(int fd);
            char *ttyname_r(int fd, char *name, int namelen);

POSIX    cc [ flag ...] file ... -D_POSIX_PTHREAD_SEMANTICS [ library ...]
            int ttyname_r(int fd, char *name, size_t namesize);

DESCRIPTION    The ttyname() function returns a pointer to a string containing the null-terminated path
                name of the terminal device associated with file descriptor fd. The return value may
                point to static data whose content is overwritten by each call.

                The ttyname_r() function has the same functionality as ttyname() except that the caller
                must supply a buffer name with length namelen to store the result; this buffer must be at
                least _POSIX_PATH_MAX in size (defined in <limits.h>). The POSIX version (see standards(5))
                of ttyname_r() takes a namesize parameter of type size_t.

RETURN VALUES    Upon successful completion, ttyname() and ttyname_r() return a pointer to a string.
                Otherwise, a null pointer is returned and errno is set to indicate the error.

                The POSIX ttyname_r() returns zero if successful, or the error number upon failure.

ERRORS    The ttyname_r() function will fail if:
            ERANGE  The size of the buffer is smaller than the result to be returned.

            The ttyname() function may fail if:
            EBADF   The fd argument is not a valid file descriptor.
            ENOTTY  The fd argument does not refer to a terminal device.

FILES    /dev/*                device file

ATTRIBUTES    See attributes(5) for descriptions of the following attributes:

            ATTRIBUTE TYPE | ATTRIBUTE VALUE
            MT-Level        | See NOTES below.

SEE ALSO    Intro(3), gettext(3C), setlocale(3C), attributes(5), standards(5)

NOTES    When compiling multithread programs, see Intro(3), Notes On Multithread Applications.

            If the application is linked with -lintl, then messages printed from this function are in the
            native language specified by the LC_MESSAGES locale category; see setlocale(3C).
            The return value points to static data whose content is overwritten by each call.

3C-1736               SunOS 5.6          modified 20 Mar 1997
ttyname() is unsafe in multi-thread applications. ttyname_r() is MT-Safe, and should be used instead.

Solaris 2.4 and earlier releases provided definitions of the ttyname_r() interface as specified in POSIX.1c Draft 6. The final POSIX.1c standard changed the interface as described above. Support for the Draft 6 interface is provided for compatibility only and may not be supported in future releases. New applications and libraries should use the POSIX standard interface.
NAME
ttyslot – find the slot in the utmp file of the current user

SYNOPSIS
#include <stdlib.h>
int ttyslot(void);

DESCRIPTION
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

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
getutent(3C), ttyname(3C), attributes(5)
NAME       t_unbind – disable a transport endpoint

SYNOPSIS   cc [ flag ...] file ... -lnsl [ library ... ]
            #include <xti.h>
            int t_unbind(int fd);

DESCRIPTION This routine is part of the XTI interfaces which evolved from the TLI interfaces. XTI
            represents the future evolution of these interfaces. However, TLI interfaces are supported
            for compatibility. When using a TLI routine that has the same name as an XTI routine, a
            different header file, tuser.h, must be used. Refer to the section, TLI COMPATIBILITY,
            for a description of differences between the two interfaces.

            The t_unbind() function disables the transport endpoint specified by fd which was previ-
            ously bound by t_bind(3N). On completion of this call, no further data or events des-
            tined for this transport endpoint will be accepted by the transport provider. An endpoint
            which is disabled by using t_unbind() can be enabled by a subsequent call to t_bind().

VALID STATES The only legitimate state (see t_getstate(3N)) for a call to this routine is T_IDLE.

RETURN VALUES t_unbind() returns:
            0 On success.
            −1 On failure.

            On failure, t_errno is set to indicate the error, and possibly errno is set.

ERRORS     On failure, t_errno is set to one of the following:
            TBADF The specified file descriptor does not refer to a transport endpoint.
            TLOOK An asynchronous event has occurred on this transport endpoint.
            TOUTSTATE The communications endpoint referenced by fd or resfd is not in one of
            the states in which a call to this function is valid.
            TPROTO This error indicates that a communication problem has been detected
            between XTI and the transport provider for which there is no other suit-
            able XTI t_errno value.
            TSYSERR A system error has occurred during execution of this function, errno
            will be set to the specific error.

TLI       The XTI and TLI interface definitions have common names but use different header files.
COMPATIBILITY This, and other semantic differences between the two interfaces are described in the sub-
            sections below.

Interface Header The XTI interfaces use the header file, xti.h. TLI interfaces should not use this header.
            They should use the header:

modified 10 Feb 1997 SunOS 5.6  3N-1739
#include <tiuser.h>

Error Description

Values

The `t_errno` value that can be set by the XTI interface and cannot be set by the TLI interface is:

TPROTO

For more information refer to the *Transport Interfaces Programming Guide*.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO

t_bind(3N), `attributes(5)`

*Transport Interfaces Programming Guide*
<table>
<thead>
<tr>
<th>NAME</th>
<th>typeahead – check for type-ahead characters</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYNOPSIS</td>
<td>#include &lt;curses.h&gt;</td>
</tr>
</tbody>
</table>

```c
int typeahead(int fd);
```

<table>
<thead>
<tr>
<th>ARGUMENTS</th>
<th>$fd$ Is the file descriptor that is used to check for type-ahead characters.</th>
</tr>
</thead>
</table>

| DESCRIPTION | The `typeahead()` function specifies the file descriptor ($fd$) to use to check for type-ahead characters (characters typed by the user but not yet processed by X/Open Curses). X/Open Curses checks for type-ahead characters periodically while updating the screen. If characters are found, the current update is postponed until the next `refresh(3XC)` or `doupdate(3XC)`. This speeds up response to commands that have been typed ahead. Normally, the input file pointer passed to `newterm(3XC)`, or `stdin` in the case of `initscr(3XC)`, is used for type-ahead checking. If $fd$ is -1, no type-ahead checking is done. |

<table>
<thead>
<tr>
<th>RETURN VALUES</th>
<th>On success, the <code>typeahead()</code> function returns <code>OK</code>. Otherwise, it returns <code>ERR</code>.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>ERRORS</th>
<th>None.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>SEE ALSO</th>
<th><code>doupdate(3XC)</code>, <code>getch(3XC)</code>, <code>initscr(3XC)</code></th>
</tr>
</thead>
</table>
The `ualarm()` function causes the `SIGALRM` signal to be generated for the calling process after the number of real-time microseconds specified by the `useconds` argument has elapsed. When the `interval` argument is non-zero, repeated timeout notification occurs with a period in microseconds specified by the `interval` argument. If the notification signal, `SIGALRM`, is not caught or ignored, the calling process is terminated. Because of scheduling delays, resumption of execution when the signal is caught may be delayed an arbitrary amount of time.

Interactions between `ualarm()` and either `alarm(2)` or `sleep(3C)` are unspecified.

The `ualarm()` function returns the number of microseconds remaining from the previous `ualarm()` call. If no timeouts are pending or if `ualarm()` has not previously been called, `ualarm()` returns 0.

No errors are defined.

The `ualarm()` function is a simplified interface to `setitimer(2)`, and uses the `ITIMER_REAL` interval timer.

`alarm(2)`, `setitimer(2)`, `sighold(3C)`, `signal(3C)`, `sleep(3C)`, `usleep(3C)`
<table>
<thead>
<tr>
<th>NAME</th>
<th>unctrl – convert character to printable form</th>
</tr>
</thead>
</table>
| SYNOPSIS     | `#include <unctrl.h>`
               | `const char *unctrl(chtype c);`          |
| ARGUMENTS    | `c`          | Is a character.                           |
| DESCRIPTION  | The `unctrl()` function converts the character code `c` into a printable form (if unprintable). Control characters are displayed using the `^x` notation where `^` identifies the control key and `x` represents an alphanumerical character that is pressed while the control key is held down. Characters which have their eighth bit set are represented using the meta notation `M-X` where `X` is the byte with eighth bit stripped. This stripped byte will represent either a printable character or a control character. If it is a control character, `X` is actually represented using `^X` notation. For example, `0xCD` in ASCII is `M-^K`. |
| RETURN VALUES| On success, the `unctrl()` function returns the generated string. Otherwise, it returns a null pointer. |
| ERRORS       | None.         |
| SEE ALSO     | `addch(3XC), addstr(3XC), wunctrl(3XC)` |
NAME ungetc – push character back onto input stream

SYNOPSIS

```c
#include <stdio.h>

int ungetc(int c, FILE *stream);
```

DESCRIPTION The `ungetc()` function inserts the character specified by `c` (converted to an `unsigned char`) into the buffer associated with an input stream (see `intro(3)`). That character, `c`, will be returned by the next `getc(3S)` call on that stream. `ungetc()` returns `c`, and leaves the file corresponding to `stream` unchanged. A successful call to `ungetc()` clears the `EOF` indicator for `stream`.

Four bytes of pushback are guaranteed.

The value of the file position indicator for `stream` after reading or discarding all pushed-back characters will be the same as it was before the characters were pushed back.

If `c` equals `EOF`, `ungetc()` does nothing to the buffer and returns `EOF`.

`fseek()`, `rewind()` (both described on `fseek(3S)`), and `fsetpos(3S)` erase the memory of inserted characters for the stream on which they are applied.

RETURN VALUES `ungetc()` returns `EOF` if it cannot insert the character.

ATTRIBUTES See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO `intro(3)`, `fseek(3S)`, `fsetpos(3S)`, `getc(3S)`, `setbuf(3S)`, `stdio(3S)`, `attributes(5)`
NAME | ungetch, unget_wch – push character back onto the input queue

SYNOPSIS | #include <curses.h>
int ungetch(int ch);
int unget_wch(const wchar_t wch);

ARGUMENTS | ch | Is the single byte character to be put back in the input queue for the next call to getch(3XC).
wch | Is the wide character to be put back in the input queue for the next call to get_wch(3XC).

DESCRIPTION | The ungetch() function pushes ch back onto the input queue until the next call to getch(). The unget_wch() function is similar to ungetch() except that ch can be of type wchar_t.

RETURN VALUES | On success, these functions return OK. Otherwise, they return ERR.

ERRORS | None.

SEE ALSO | get_wch(3XC), getch(3XC)
NAME
ungetwc – push wide-character code back into input stream

SYNOPSIS
#include <stdio.h>
#include <wchar.h>
wint_t ungetwc(wint_t wc, FILE *stream);

DESCRIPTION
The ungetwc() function pushes the character corresponding to the wide character code specified by wc back onto the input stream pointed to by stream. The pushed-back characters will be returned by subsequent reads on that stream in the reverse order of their pushing. A successful intervening call (with the stream pointed to by stream) to a file-positioning function (fseek(3S), fsetpos(3S) or rewind(3S)) discards any pushed-back characters for the stream. The external storage corresponding to the stream is unchanged.

One character of push-back is guaranteed. If ungetwc() is called too many times on the same stream without an intervening read or file-positioning operation on that stream, the operation may fail.

If the value of wc equals that of the macro WEOF, the operation fails and the input stream is unchanged.

A successful call to ungetwc() clears the end-of-file indicator for the stream. The value of the file-position indicator for the stream after reading or discarding all pushed-back characters will be the same as it was before the characters were pushed back. The file-position indicator is decremented (by one or more) by each successful call to ungetwc(); if its value was 0 before a call, its value is indeterminate after the call.

RETURN VALUES
Upon successful completion, ungetwc() returns the wide-character code corresponding to the pushed-back character. Otherwise it returns WEOF.

ERRORS
The ungetwc() function may fail if:

EILSEQ An invalid character sequence is detected, or a wide-character code does not correspond to a valid character.

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
read(2), fseek(3S), fsetpos(3S), rewind(3S), setbuf(3S), attributes(5)
NAME
unlockpt – unlock a pseudo-terminal master/slave pair

SYNOPSIS
#include <stdlib.h>
int unlockpt(int fildes);

DESCRIPTION
The unlockpt() function unlocks the slave pseudo-terminal device associated with the
master to which fildes refers.
Portable applications must call unlockpt() before opening the slave side of a pseudo-
terminal device.

RETURN VALUES
Upon successful completion, unlockpt() returns 0. Otherwise, it returns −1 and sets
errno to indicate the error.

ERRORS
The unlockpt() function may fail if:
EBADF The fildes argument is not a file descriptor open for writing.
EINVAL The fildes argument is not associated with a master pseudo-terminal device.

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
open(2), grantpt(3C), ptsname(3C), attributes(5)
STREAMS Programming Guide
NAME  use_env – set values of lines and columns

SYNOPSIS  
#include <curses.h>

void use_env(char bool);

ARGUMENTS  
bool   Is a Boolean expression.

DESCRIPTION  
The use_env() function takes the values for lines and columns from the terminfo database (if bool is FALSE) or from environmental variables LINES and COLUMNS (if bool is TRUE). If no environmental variables have been set, the window size is used. This function must be set before initscr(3XC), newterm(3XC), or setupterm(3XC) is called. The default action is TRUE.

RETURN VALUES  
The use_env() function does not return a value.

ERRORS  
None.

SEE ALSO  
del_curterm(3XC), initscr(3XC)
NAME  usleep – suspend execution for interval in microseconds

SYNOPSIS  #include <unistd.h>
           int usleep(useconds_t useconds);

DESCRIPTION  The usleep() function suspends the current process from execution for the number of microseconds specified by the useconds argument. (A microsecond is .000001 seconds.) Because of other activity, or because of the time spent in processing the call, the actual suspension time may be longer than the amount of time specified.

The useconds argument must be less than 1,000,000. If the value of useconds is 0, then the call has no effect.

The usleep() function uses the process’ real-time interval timer to indicate to the system when the process should be woken up.

There is one real-time interval timer for each process. The usleep() function will not interfere with a previous setting of this timer. If the process has set this timer prior to calling usleep(), and if the time specified by useconds equals or exceeds the interval timer’s prior setting, the process will be woken up shortly before the timer was set to expire.

Interactions between usleep() and either alarm(2) or sleep(3C) are unspecified.

RETURN VALUES  On successful completion, usleep() returns 0. Otherwise, it returns −1 and sets errno to indicate the error.

ERRORS  The usleep() function may fail if:

EINVAL  The time interval specified 1,000,000 or more microseconds.

USAGE  The usleep() function is included for its historical usage. The setitimer(2) function is preferred over this function.

SEE ALSO  alarm(2), poll(2), setitimer(2), sigaction(2), sigprocmask(2), select(3C), sleep(3C), ualarm(3C)
NAME
vidattr, vid_attr, vidputs, vid_puts – display string with video attributes

SYNOPSIS
#include <term.h>

int vidattr(chtype attr);
int vid_attr(attr_t attr, short color_pair, void *opt);
int vidputs(chtype attr, int (*putfunc)(int));
int vid_puts(attr_t attr, short color_pair, void *opt,
             int (*putfunc)(int));

ARGUMENTS
attr Is the rendition of the foreground window.
color_pair Is a color pair.
opt Is reserved for future use. Currently, this must be a null pointer.
putfunc Is a user-supplied output function.
putwfunc Is a user-supplied output function.

DESCRIPTION
These functions change the terminal’s attributes.
The vidattr() function sends a request to the terminal to display subsequent characters
with the rendition specified by attr. It uses the putchar(3S) function to display the char-
acter. The vid_attr() function is similar to the vidattr() function except that it accepts the
rendition as a attr_t object. This lets you use the attribute constants that begin with WA_.
The vidputs() and vid_puts() functions are similar to the vidattr() and vid_attr() func-
tions, respectively, except that the user-supplied putfunc function is used instead of
putchar(). The output of the user-supplied function is ignored by vidputs() and
vid_puts() functions.

RETURN VALUES
On success, these functions return OK. Otherwise, they return ERR.

ERRORS
None.

SEE ALSO
doupdate(3XC), is_linetouched(3XC), putchar(3S), tigetflag(3XC)
NAME

vlfmt – display error message in standard format and pass to logging and monitoring services

DESCRIPTION

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().

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.

RETURN VALUE

Upon success, vlfmt() 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.

EXAMPLE

The following demonstrates how vlfmt() could be used to write an errlog() routine:

```c
#include <pfmt.h>
#include <stdarg.h>
/*
 * errlog should be called like
 * errlog(log_info, format, arg1, ...);
 */

void errlog(long log_info, ...) {  
    va_list ap;
    char *format;
    va_start(ap, );
    format = va_arg(ap, char *);
    (void) vlfmt(stderr, log_info | MM_ERROR, format, ap);
}
```

modified 29 Dec 1996

SunOS 5.6

3C-1751
va_end(ap);
(void) abort();
}

NOTES
Since \texttt{vlfmt()} uses \texttt{gettext(3C)}, it is recommended that \texttt{vlfmt()} not be used.

ATTRIBUTES
See \texttt{attributes(5)} for descriptions of the following attributes:

\begin{table}[h]
\centering
\begin{tabular}{|c|c|}
\hline
ATTRIBUTE TYPE & ATTRIBUTE VALUE \\
\hline
MT-Level & MT-Safe \\
\hline
\end{tabular}
\end{table}

SEE ALSO \texttt{gettext(3C), lfmt(3C), attributes(5), stdarg(5)}
NAME
volmgt_acquire – reserve removable media device

SYNOPSIS
cc [ flag ...] file ... -lvolmgt [ library ...]
#include <sys/types.h> #include <volmgt.h>
int volmgt_acquire(char *dev, char *id, int ovr, char **err, pid_t *pidp);

DESCRIPTION
The volmgt_acquire() routine reserves the removable media device specified as dev.
volmgt_acquire() operates in two different modes, depending on whether or not Volume
Management is running. See vold(1M).
If Volume Management is running, volmgt_acquire() attempts to reserve the removable
media device specified as dev. Specify dev as either a symbolic device name (for example,
floppy0) or a physical device pathname (for example,/vol/dsk/unnamed_floppy).
If Volume Management is not running, volmgt_acquire() requires callers to specify a
physical device pathname for dev. Specifying dev as a symbolic device name is not
acceptable. In this mode, volmgt_acquire() relies entirely on the major and minor
numbers of the device to determine whether or not the device is reserved.
If dev is free, volmgt_acquire() updates the internal device reservation database with the
caller’s process id (pid) and the specified id string.
If dev is reserved by another process, the reservation attempt fails and volmgt_acquire():
  • sets errno to EBUSY
  • fills the caller’s id value in the array pointed to by err
  • fills in the pid to which the pointer pidp points with the pid of the process
    which holds the reservation, if the supplied pidp is non-zero
If the override ovr is non-zero, the call overrides the device reservation.

RETURN VALUES
Upon successful completion, volmgt_acquire() returns a non-zero value.
Upon failure, volmgt_acquire() returns 0. If the return value is 0, and errno is set to
EBUSY, the address pointed to by err contains the string that was specified as id (when the
device was reserved by the process holding the reservation).

ERRORS
The volmgt_acquire() routine fails if one or more of the following are true:
EINVAL One of the specified arguments is invalid or missing.
EBUSY dev is already reserved by another process (and ovr was not set to a
non-zero value)

EXAMPLES
In the following example, Volume Management is running and the first floppy drive is
reserved, accessed and released.
#include <volmgt.h>

char *errp;
if (!volmgt_acquire("floppy0", "FileMgr", 0, NULL,
The following example shows how callers can override a lock on another process using `volmgt_acquire()`. 

```c
char *errp, buf[20];
int override = 0;
pid_t pid;
if (!volmgt_acquire("floppy0", "FileMgr", 0, &errp, &pid)) {
    if (errno == EBUSY) {
        (void) printf("override %s (pid=%ld)?\n", errp, pid);
        (void) fgets(buf, 20, stdin);
        if (buf[0] == 'y') {
            override++;
        } else {
            /* handle other errors */
            ...
        }
    } else {
        /* handle other errors */
        ...
    }
} else {
    /* handle other errors */
    ...
}
if (override) {
    if (!volmgt_acquire("floppy0", "FileMgr", 1, &errp, NULL)) {
        /* really give up this time! */
        ...
    }
}
```

**ATTRIBUTES**

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
SEE ALSO  
void(1M), free(3C), malloc(3C), volmgt_release(3X), attributes(5)

NOTES  
When returning a string through err, `volmgt_acquire()` allocates a memory area using malloc(3C). Use free(3C) to release the memory area when no longer needed. The `ovr` argument is intended to allow callers to override the current device reservation. It is assumed that the calling application has determined that the current reservation can safely be cleared. See EXAMPLES.
NAME       volmgt_check – have Volume Management check for media

SYNOPSIS   cc [ flag ... ] file ... -lvolmgt [ library... ]

#include <volmgt.h>

int volmgt_check(char *pathname);

DESCRIPTION This routine asks Volume Management to check the specified pathname and determine if new media has been inserted in that drive. If a null pointer is passed in, then Volume Management will check each device it is managing that can be checked. If new media is found, volmgt_check() tells Volume Management to initiate any "actions" specified in /etc/vold.conf (see vold.conf(4)).

RETURN VALUES This routine returns 0 if no media was found, and a non-zero value if any media was found.

ERRORS     This routine can fail, returning 0, if a stat(2) or open(2) of the supplied pathname fails, or if any of the following is true:

ENXIO      Volume Management is not running.
EINTR      An interrupt signal was detected while checking for media.

EXAMPLES   To check if any drive managed by Volume Management has any new media inserted in it:

    if (volmgt_check(NULL)) {
        (void) printf("Volume Management found media\n");
    }

This would also request Volume Management to take whatever action was specified in /etc/vold.conf for any media found.

ATTRIBUTES See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO    cc(1B), volcheck(1), vold(1M), open(2), stat(2), volmgt_inuse(3X), volmgt_running(3X), vold.conf(4), attributes(5), volfs(7FS)

NOTES       Volume Management must be running for this routine to work.

Since volmgt_check() returns 0 for two different cases (both when no media is found, and when an error occurs), it is up to the user to to check errno to differentiate the two, and to ensure that Volume Management is running.
NAME volmgt_feature_enabled – check whether specific Volume Management features are enabled

SYNOPSIS cc [flag ...] file ... -l volmgt [library ...]
#include <volmgt.h>
int volmgt_feature_enabled(char *feat_str);

DESCRIPTION The volmgt_feature_enabled() routine checks whether specific Volume Management features are enabled. volmgt_feature_enabled() checks for the Volume Management features passed in to it by the feat_str parameter. Currently, the only supported feature string that volmgt_feature_enabled() checks for is floppy-summit-interfaces. The floppy-summit-interfaces feature string checks for the presence of the libvolmgt routines volmgt_acquire() and volmgt_release(). The list of features that volmgt_feature_enabled() checks for is expected to expand in the future.

RETURN VALUES 0 is returned if the specified feature is not currently available. A non-zero value indicates that the specified feature is currently available.

EXAMPLES In the following example, volmgt_feature_enabled() checks whether the floppy-summit-interfaces feature is enabled.

```c
if (volmgt_feature_enabled("floppy-summit-interfaces")) {
    (void) printf("Media Sharing Routines ARE present\n");
} else {
    (void) printf("Media Sharing Routines are NOT present\n");
}
```

ATTRIBUTES See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO volmgt_acquire(3X), volmgt_release(3X), attributes(5)
NAME
volmgt_inuse – check whether or not Volume Management is managing a pathname

SYNOPSIS
cc [ flag ... ] file ... -lvolmgt [ library... ]
#include <volmgt.h>
int volmgt_inuse(char *pathname);

DESCRIPTION
volmgt_inuse( ) checks whether Volume Management is managing the specified pathname.

RETURN VALUES
A non-zero value is returned if Volume Management is managing the specified pathname, otherwise 0 is returned.

ERRORS
This routine can fail, returning 0, if a stat(2) of the supplied pathname or an open(2) of /dev/volctl fails, or if any of the following is true:
ENXIO Volume Management is not running.
EINTR An interrupt signal was detected while checking for the supplied pathname for use.

EXAMPLES
To see if Volume Management is managing the first floppy disk:

    if (volmgt_inuse("/dev/rdiskette0") != 0) {
        (void) printf("volmgt is managing diskette 0\n");
    } else {
        (void) printf("volmgt is NOT managing diskette 0\n");
    }

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
cc(1B), void(1M), open(2), stat(2), errno(3C), volmgt_check(3X), volmgt_running(3X), attributes(5), volfs(7FS)

NOTES
This routine requires Volume Management to be running.
Since volmgt_inuse( ) returns 0 for two different cases (both when a volume is not in use, and when an error occurs), it is up to the user to check errno to differentiate the two, and to ensure that Volume Management is running.
NAME      volmgt_release – release removable media device reservation

SYNOPSIS  cc [ flag ...] file ... -lvolmgt [ library...]  
#include <volmgt.h>  
int volmgt_release(char *dev);

DESCRIPTION The volmgt_release() routine releases the removable media device reservation specified as dev. See volmgt_acquire(3X) for a description of dev.

If dev is reserved by the caller, volmgt_release() updates the internal device reservation database to indicate that the device is no longer reserved. If the requested device is reserved by another process, the release attempt fails and errno is set to 0.

RETURN VALUES Upon successful completion, volmgt_release returns a non-zero value. Upon failure, 0 is returned.

ERRORS On failure, volmgt_release() returns 0, and sets errno for one of the following conditions:
EINVAL      dev was invalid or missing.
EBUSY       dev was not reserved by the caller.

EXAMPLES In the following example, Volume Management is running, and the first floppy drive is reserved, accessed and released.

#include <volmgt.h>

char *errp;

if (!volmgt_acquire("floppy0", "FileMgr", 0, &errp,
NULL)) {
    /* handle error case */
    ...
}

/* floppy acquired - now access it */
if (!volmgt_release("floppy0")) {
    /* handle error case */
    ...
}
ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
<tr>
<td>Interface Stability</td>
<td>Stable</td>
</tr>
</tbody>
</table>

SEE ALSO

vold(1M), volmgt_acquire(3X), attributes(5)
NAME
volmgt_root – return the Volume Management root directory

SYNOPSIS
cc [ flag ...] file ... -lvolmgt [ library .... ]
#include <volmgt.h>
char *volmgt_root(void);

DESCRIPTION
volmgt_root() returns the current Volume Management root directory, which by default is /vol but can be configured to be in a different location.

RETURN VALUES
A pointer to a static string containing the root directory for Volume Management is returned.

ERRORS
This routine may fail if an open() of /dev/volctl fails. If this occurs a pointer to the default Volume Management root directory is returned.

EXAMPLES
To find out where the Volume Management root directory is:
if ((path = volmgt_root()) != NULL) {
    (void) printf("Volume Management root dir=%s\n", path);
} else {
    (void) printf("can’t find Volume Management root dir\n");
}

FILES
/vol       Default location for the Volume Management root directory

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
cc(1B), vold(1M), open(2), volmgt_check(3X), volmgt_inuse(3X), volmgt_running(3X), attributes(5), volfs(7FS)

NOTES
This routine will return the default root directory location even when Volume Management is not running.

modified 31 Dec 1996
SunOS 5.6
3X-1761
NAME
volmgt_running — return whether or not Volume Management is running

SYNOPSIS
cc [ flag ... ] file ... -lvolmgt [ library.... ]
#include <volmgt.h>
int volmgt_running(void);

DESCRIPTION
volmgt_running() tells whether or not Volume Management is running.

RETURN VALUES
A non-zero value is returned if Volume Management is running, else 0 is returned.

ERRORS
volmgt_running() will fail, returning 0, if a stat(2) or open(2) of /dev/volctl fails, or if any
of the following is true:
ENXIO Volume Management is not running.
EINVAL An interrupt signal was detected while checking to see if Volume
Management was running.

EXAMPLES
To see if Volume Management is running:
if (volmgt_running() != 0) {
    (void) printf("Volume Management is running\n");
} else {
    (void) printf("Volume Management is NOT running\n");
}

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
cc(1B), vold(1M), open(2), stat(2), volmgt_check(3X), volmgt_inuse(3X), attributes(5), volfs(7FS)

NOTES
Volume Management must be running for many of the Volume Management library rou-
tines to work.
NAME
volmgt_symname, volmgt_symdev – convert between Volume Management symbolic names, and the devices that correspond to them

SYNOPSIS
cc [flag ...] file ... -Ivolmgt [library,...]
#include <volmgt.h>
char *volmgt_symname(char *pathname);
char *volmgt_symdev(char *symname);

DESCRIPTION
These two routines compliment each other, translating between Volume Management’s symbolic name for a device, called a symname, and the /dev pathname for that same device.

volmgt_symname() converts a supplied /dev pathname to a symname, Volume Management’s idea of that device’s symbolic name (see volfs(7FS) for a description of Volume Management symbolic names).

volmgt_symdev() does the opposite conversion, converting between a symname, Volume Management’s idea of a device’s symbolic name for a volume, to the /dev pathname for that device.

RETURN VALUES
volmgt_symname() returns the symbolic name for the device pathname supplied, and volmgt_symdev() returns the device pathname for the supplied symbolic name. These strings are allocated upon success, and therefore must be freed by the caller when they are no longer needed (see free(3C)).

ERRORS
volmgt_symname() can fail, returning a null string pointer, if a stat(2) of the supplied pathname fails, or if an open(2) of /dev/volctl fails, or if any of the following is true:
ENXIO Volume Management is not running.
EINVAL An interrupt signal was detected while trying to convert the supplied pathname to a symname.

volmgt_symdev() can fail if an open(2) of /dev/volctl fails, or if any of the following is true:
ENXIO Volume Management is not running.
EINVAL An interrupt signal was detected while trying to convert the supplied symname to a /dev pathname.

EXAMPLES
The following tests how many floppies Volume Management currently sees in floppy drives (up to 10):

```c
for (i=0; i < 10; i++) {
    (void) sprintf(path, "floppy%d", i);
    if (volmgt_symdev(path) != NULL) {
        (void) printf("volume %s is in drive %d\n", path, i);
    }
}
```

modified 31 Dec 1996 SunOS 5.6 3X-1763
This code finds out what symbolic name (if any) Volume Management has for /dev/rdsk/c0t6d0s2:

```c
if ((nm = volmgt_symname("/dev/rdsk/c0t6d0s2")) == NULL) {
    (void) printf("path not managed
");
} else {
    (void) printf("path managed as %s\n", nm);
}
```

### ATTRIBUTES
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### SEE ALSO
- cc(1B), void(1M), open(2), stat(2), free(3C), malloc(3C), volmgt_check(3X), volmgt_inuse(3X), volmgt_running(3X), attributes(5), volfs(7FS)

### NOTES
These routines only work when Volume Management is running.

### BUGS
There should be a straightforward way to query Volume Management for a list of all media types it’s managing, and how many of each type are being managed.
vpfmt – display error message in standard format and pass to logging and monitoring services

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().

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.

Upon success, lfmt() returns the number of bytes transmitted. Upon failure, it returns a negative value:

-1 write error to stream.

The following demonstrates how vpfmt() could be used to write an error() routine:

```c
#include <pfmt.h>
#include <stdarg.h>

/*
 *  error should be called like
 *  error(format, arg1, ...);
 */

void error(...) {
    va_list ap;
    char *format;
    va_start(ap, );
    format = va_arg(ap, char *);
    (void) vpfmt(stderr, MM_ERROR, format, ap);
    va_end(ap);
    (void) abort();
}
```

modified 29 Dec 1996 SunOS 5.6 3C-1765
NOTES

Since `vpfmt()` uses `gettext(3C)`, it is recommended that `vpfmt()` not be used.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-safe</td>
</tr>
</tbody>
</table>

SEE ALSO

`pfmt(3C)`, `attributes(5)`, `stdarg(5)`
NAME

vprintf, vfprintf, vsprintf, vsnprintf – print formatted output of a variable argument list

SYNOPSIS

#include <stdio.h>
#include <stdarg.h>

int vprintf(const char *format, va_list ap);
int vfprintf(FILE *stream, const char *format, va_list ap);
int vsprintf(char *s, const char *format, va_list ap);
int vsnprintf(char *s, size_t n, const char *format, va_list ap);

DESCRIPTION

The vprintf(), vfprintf(), vsprintf() and vsnprintf() functions are the same as printf(), fprintf(), sprintf(), and snprintf(), respectively, except that instead of being called with a variable number of arguments, they are called with an argument list as defined by the <stdarg.h> header.

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 ap to the vprintf 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().

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 ap is of type va_list, and parmN 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 ap can be traversed again if va_start() is called again after va_end().) In the example below, the error() arguments, arg1, arg2, ..., are given to vfprintf() in the argument ap.

RETURN VALUES

The vprintf(), vfprintf(), and vsprintf() functions return the number of characters transmitted (not including the \0 in the case of vsprintf()). The vsnprintf() function returns the number of characters formatted, that is, the number of characters that would have been written to the buffer if it were large enough. Each function returns a negative value if an output error was encountered.

ERRORS

The vprintf() and vfprintf() functions will fail if either the stream is unbuffered or the stream’s buffer needed to be flushed and:

EFBIG The file is a regular file and an attempt was made to write at or beyond the offset maximum.

EXAMPLES

The following demonstrates how vfprintf() could be used to write an error routine:

#include <stdio.h>
#include <stdarg.h>

... 

/* error should be called like

modified 30 Dec 1996 SunOS 5.6 3S-1767
vprintf (3S) Standard I/O Functions

`error(function_name, format, arg1, ...);`

```c
void error(char *function_name, char *format, ...)
{
    va_list ap;
    va_start(ap, );
    /* print out name of function causing error */
    (void) fprintf(stderr, "ERR in %s: ", function_name);
    /* print out remainder of message */
    (void) vfprintf(stderr, format, ap);
    va_end(ap);
    (void) abort;
}
```

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>See NOTES below.</td>
</tr>
</tbody>
</table>

SEE ALSO printf(3S), attributes(5), stdarg(5)

NOTES

The vprintf(), vfprintf(), and vsprintf() functions are MT-Safe in multi-thread applications.

3S-1768 SunOS 5.6 modified 30 Dec 1996
NAME      vsyslog – log message with a varargs argument list

SYNOPSIS  #include <syslog.h>
           #include <varargs.h>
           int vsyslog(int priority, const char *message, va_list ap);

DESCRIPTION vsyslog() is the same as syslog(3) except that instead of being called with a variable number of arguments, it is called with an argument list as defined by varargs(5).

EXAMPLES  The following demonstrates how vsyslog() could be used to write an error routine.

```
#include <syslog.h>
#include <varargs.h>
...
/*
 * error should be called like:
 *    error(pri, function_name, format, arg1, arg2. . .);
 *    Note that pri, function_name, and format cannot be declared
 * separately because of the definition of varargs.
 */

/*VARARGS0*/
void error(va_alist) {
   va_dcl; {
   va_list args;
   int pri;
   char *message;

   va_start(args);
   pri = va_arg(args, int);
   /* log name of function causing error */
   (void) syslog(pri, "ERROR in %s", va_arg(args, char *));
   message = va_arg(args, char *);
   /* log remainder of message */
   (void) vsyslog(pri, msg, args);
   va_end(args);
   (void) abort();
}
```

modified 29 Dec 1996          SunOS 5.6          3-1769
ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO syslog(3), attributes(5), varargs(5)
NAME
wait3, wait4 — wait for process to terminate or stop

SYNOPSIS
#include <sys/wait.h>
#include <sys/time.h>
#include <sys/resource.h>

pid_t wait3(int *statusp, int options,
            struct rusage *rusage);

pid_t wait4(pid_t pid, int *statusp, int options,
            struct rusage *rusage);

DESCRIPTION
wait3() delays its caller until a signal is received or one of its child processes terminates or stops due to tracing. If any child process has died or stopped due to tracing and this has not already been reported, return is immediate, returning the process ID and status of one of those children. If that child process has died, it is discarded. If there are no children, −1 is returned immediately. If there are only running or stopped but reported children, the calling process is blocked.

If statusp is not a NULL pointer, then on return from a successful wait3() call, the status of the child process is stored in the integer pointed to by statusp. *statusp indicates the cause of termination and other information about the terminated process in the following manner:

- If the low-order 8 bits of *statusp are equal to 0177, the child process has stopped; the 8 bits higher up from the low-order 8 bits of *statusp contain the number of the signal that caused the process to stop. See signal(5).
- If the low-order 8 bits of *statusp are non-zero and are not equal to 0177, the child process terminated due to a signal; the low-order 7 bits of *statusp contain the number of the signal that terminated the process. In addition, if the low-order seventh bit of *statusp (that is, bit 0200) is set, a “core image” of the process was produced; see signal(5).
- Otherwise, the child process terminated due to an exit() call; the 8 bits higher up from the low-order 8 bits of *statusp contain the low-order 8 bits of the argument that the child process passed to exit(); see exit(2).

options is constructed from the bitwise inclusive OR of zero or more of the following flags, defined in the header <sys/wait.h>:

WNOHANG
Execution of the calling process is not suspended if status is not immediately available for any child process.

WUNTRACED
The status of any child processes that are stopped, and whose status has not yet been reported since they stopped, are also reported to the requesting process.

modified 3 Mar 1995

SunOS 5.6

3C-1771
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 the `ru_utime` and `ru_stime`, members of the rusage structure respectively.

When the `WNOHANG` option is specified and no processes have status to report, `wait3()` returns 0. The `WNOHANG` and `WUNTRACED` options may be combined by ORing the two values.

`wait4()` is an extended interface. With a `pid` argument of 0, it is equivalent to `wait3()`. If `pid` has a nonzero value, then `wait4()` returns status only for the indicated process ID, but not for any other child processes. The status can be evaluated using the macros defined by `wstat(5)`.

**RETURN VALUES**

If `wait3()` or `wait4()` returns due to a stopped or terminated child process, the process ID of the child is returned to the calling process. Otherwise, a value of −1 is returned and `errno` is set to indicate the error.

If `wait3()` or `wait4()` return due to the delivery of a signal to the calling process, a value of −1 is returned and `errno` is set to `EINTR`. If `WNOHANG` was set in `options`, it has at least one child process specified by `pid` for which status is not available, and status is not available for any process specified by `pid`, a value of zero is returned. Otherwise, a value of −1 is returned, and `errno` is set to indicate the error.

`wait3()` and `wait4()` return 0 if `WNOHANG` is specified and there are no stopped or exited children, and return the process ID of the child process if they return due to a stopped or terminated child process. Otherwise, they return a value of −1 and sets `errno` to indicate the error.

**ERRORS**

`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 `statusp` or `rusage` arguments point to an illegal address.
- `EINTR` The function was interrupted by a signal. The value of the location pointed to by `statusp` is undefined.
- `EINVAL` The value of `options` is not valid.

`wait4()` may set `errno` to:

- `ECHILD` The process specified by `pid` does not exist or is not a child of the calling process.

`wait3()`, and `wait4()` will terminate prematurely, return −1, and set `errno` to `EINTR` upon the arrival of a signal whose `SA_RESTART` bit in its flags field is not set (see `sigaction(2)`).

**SEE ALSO**

`kill(1)`, `exit(2)`, `wait(2)`, `waitid(2)`, `waitpid(2)`, `getrusage(3C)`, `signal(3C)`, `proc(4)`, `signal(5)`, `wstat(5)`
NOTES If a parent process terminates without waiting on its children, the initialization process (process ID = 1) inherits the children. \texttt{wait3()}, and \texttt{wait4()} are automatically restarted when a process receives a signal while awaiting termination of a child process, unless the \texttt{SA\_RESTART} bit is not set in the flags for that signal.
NAME
wait, wait3, wait4, waitpid, WIFSTOPPED, WIFSIGNALED, WIFEXITED – wait for process to terminate or stop

SYNOPSIS
/usr/cc [ flag ... ] file ...
#include <sys/wait.h>
int wait( statusp)
int *statusp;
int wait3( pid, statusp, options)
int pid;
int *statusp;
int options;
#include <sys/time.h>
#include <sys/resource.h>
int wait4( pid, statusp, options, rusage)
int pid;
int *statusp;
int options;
struct rusage *rusage;
int wait3( statusp, options, rusage)
int *statusp;
int options;
struct rusage *rusage;
int wait4( pid, statusp, options, rusage)
int pid;
int *statusp;
int options;
struct rusage *rusage;
WIFSTOPPED( status)
int status;
WIFSIGNALED( status)
int status;
WIFEXITED( status)
int status;

DESCRIPTION
wait() delays its caller until a signal is received or one of its child processes terminates or stops due to tracing. If any child process has died or stopped due to tracing and this has not been reported using wait(), return is immediate, returning the process ID and exit status of one of those children. If that child process has died, it is discarded. If there are no children, return is immediate with the value -1 returned. If there are only running or stopped but reported children, the calling process is blocked.

If status is not a NULL pointer, then on return from a successful wait() call the status of the child process whose process ID is the return value of wait() is stored in the wait() union pointed to by status. The w_status member of that union is an int; it indicates the cause of termination and other information about the terminated process in the following manner:

- If the low-order 8 bits of w_status are equal to 0177, the child process has
stopped; the 8 bits higher up from the low-order 8 bits of \texttt{w_status} contain the number of the signal that caused the process to stop. See \texttt{ptrace(2)} and \texttt{sigvec(3B)}.

- If the low-order 8 bits of \texttt{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 \texttt{w_status} contain the number of the signal that terminated the process. In addition, if the low-order seventh bit of \texttt{w_status} (that is, bit 0200) is set, a “core image” of the process was produced; see \texttt{sigvec(3B)}.

- Otherwise, the child process terminated due to an \texttt{exit()} call; the 8 bits higher up from the low-order 8 bits of \texttt{w_status} contain the low-order 8 bits of the argument that the child process passed to \texttt{exit()}; see \texttt{exit(2)}.

\texttt{waitpid()} behaves identically to \texttt{wait()} if \texttt{pid} has a value of −1 and \texttt{options} has a value of zero. Otherwise, the behavior of \texttt{waitpid()} is modified by the values of \texttt{pid} and \texttt{options} as follows:

\texttt{pid} specifies a set of child processes for which status is requested. \texttt{waitpid()} only returns the status of a child process from this set.

- If \texttt{pid} is equal to −1, status is requested for any child process. In this respect, \texttt{waitpid()} is then equivalent to \texttt{wait()}.

- If \texttt{pid} is greater than zero, it specifies the process ID of a single child process for which status is requested.

- If \texttt{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 \texttt{pid} is less than −1, status is requested for any child process whose process group ID is equal to the absolute value of \texttt{pid}.

\texttt{options} is constructed from the bitwise inclusive OR of zero or more of the following flags, defined in the header \texttt{<sys/wait.h>}:

\begin{itemize}
  \item \texttt{WNOHANG} \texttt{waitpid()} does not suspend execution of the calling process if status is not immediately available for one of the child processes specified by \texttt{pid}.
  \item \texttt{WUNTRACED} The status of any child processes specified by \texttt{pid} that are stopped, and whose status has not yet been reported since they stopped, are also reported to the requesting process.
\end{itemize}

\texttt{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 \texttt{status} parameter is defined as above. The \texttt{options} parameter is used to indicate the call should not block if there are no processes that have status to report (\texttt{WNOHANG}), and/or that children of the current process that are stopped due to a \texttt{SIGTTOU}, \texttt{SIGTTIN}, \texttt{SIGSTP}, or \texttt{SIGSTOP} signal are eligible to have their status reported as well (\texttt{WUNTRACED}). A terminated child is discarded after it reports status, and a stopped process will not report its status more than once. If \texttt{rusage} is not a \texttt{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()` 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), getusage(3C), siginterrupt(3B), signal(3B),
                 sigvec(3B), signal(3C)

NOTES Use of these interfaces should be restricted to only applications written on BSD platforms. Use of these interfaces with any of the system libraries or in multi-thread applications is unsupported.

If a parent process terminates without waiting on its children, the initialization process (process ID = 1) inherits the children.

wait(), and wait3(), and wait4() are automatically restarted when a process receives a signal while awaiting termination of a child process, unless the SV_INTERRUPT bit is set in the flags for that signal.

Calls to wait() with an argument of 0 should be cast to type `int *', as in:

    wait(int *)0

Previous SunOS releases used union wait *statusp and union wait status in place of int *statusp and int status. The union contained a member w_status that could be treated in the same way as status.

Other members of the wait union could be used to extract this information more conveniently:

- If the w_stopval member had the value WSTOPPED, the child process had stopped; the value of the w_stopsig member was the signal that stopped the process.
- If the w_termsig member was non-zero, the child process terminated due to a signal; the value of the w_termsig member was the number of the signal that terminated the process. If the w_coredump member was non-zero, a core dump was produced.
- Otherwise, the child process terminated due to a call to exit(). The value of the w_retcode member was the low-order 8 bits of the argument that the child process passed to exit().

union wait is obsolete in light of the new specifications provided by IEEE Std 1003.1-1988 and endorsed by SVID89 and XPG3. SunOS Release 4.1 supports union wait for backward compatibility, but it will disappear in a future release.

modified 5 Mar 1993

SunOS 5.6

3B-1777
NAME

watchmalloc, malloc, free, realloc, memalign, valloc, calloc, cfree – debugging memory allocator

SYNOPSIS

```c
#include <stdlib.h>

void *malloc(size_t size);
void free(void *ptr);
void *realloc(void *ptr, size_t size);
void *memalign(size_t alignment, size_t size);
void *valloc(size_t size);
void *calloc(size_t nelem, size_t elsize);
void cfree(void *ptr, size_t nelem, size_t elsize);
#include <malloc.h>
int mallopt(int cmd, int value);
struct mallinfo mallinfo(void);
```

DESCRIPTION

The collection of `malloc()` routines in this shared object are an optional replacement for the standard versions of the same routines in the system C library. See `malloc(3C)`. They provide a more strict interface than the standard versions and enable enforcement of the interface via the watchpoint facility of `/proc`. See `proc(4)`.

Any dynamically linked program can be run with these routines in place of the standard routines if the following string is present in the environment (see `ld.so.1(1)`):

```
LD_PRELOAD=watchmalloc.so.1
```

The individual routine interfaces are identical to the standard ones as described in `malloc(3C)`. However, laxities provided in the standard versions are not permitted:

- Memory may not be freed more than once.
- A pointer to freed memory may not be used in a call to `realloc()`.
- A `malloc()` immediately following a `free()` will not return the same space.
- Any reference to memory that has been freed yields undefined results.

To enforce these restrictions partially, without great loss in speed as compared to the watchpoint facility described below, a freed block of memory is overwritten with the pattern `0xdeadbeef` before returning from `free()`. `malloc()` returns with the allocated memory filled with the pattern `0xbaddcafe` as a precaution against programs incorrectly expecting to receive back unmodified memory from the last `free()`. (`calloc()` always returns with the memory zero-filled.)

Entry points for `mallopt()` and `mallinfo()` are provided as empty routines, and are present only because some `malloc()` implementations provide them.
WATCHPOINTS

The watchpoint facility of /proc can be applied by a process to itself. The routines in watchmalloc.so.1 use this feature if the following string is present in the environment:

```
MALLOC_DEBUG=WATCH
```

This causes every block of freed memory to be covered with WA_WRITE watched areas. If the program attempts to write any part of freed memory, it will trigger a watchpoint trap, which will result in a SIGTRAP signal, which normally results in a program core dump.

A header is maintained before each block of allocated memory. Each header is covered with a watched area, thereby providing a red zone before and after each block of allocated memory (the header for the subsequent memory block serves as the trailing red zone for its preceding memory block). Writing just before or just after a memory block returned by malloc() will trigger a watchpoint trap.

Watchpoints incur a large performance penalty. Requesting MALLOC_DEBUG=WATCH can cause the program to run 10 to 100 times slower, depending on the use made of allocated memory.

Further options are enabled by specifying a comma-separated string of options:

```
MALLOC_DEBUG=WATCH,RW,STOP
```

WATCH Enables WA_WRITE watched areas as described above.

RW Enables both WA_READ and WA_WRITE watched areas. An attempt either to read or write freed memory or the red zones will trigger a watchpoint trap. This incurs even more overhead and can cause the program to run up to 1000 times slower.

STOP The process will stop showing a FLTWATCH machine fault if it triggers a watchpoint trap, rather than dumping core with a SIGTRAP signal. This allows a debugger to be attached to the live process at the point where it underwent the watchpoint trap. Also, the various /proc tools described in proc(1) can be used to examine the stopped process.

One of WATCH or RW must be specified, else the watchpoint facility is not engaged. RW overrides WATCH. Unrecognized options are silently ignored.

LIMITATIONS

Interposition of watchmalloc.so.1 fails innocuously if the target program is statically linked with respect to its malloc() routines. The system-supplied libraries −lmalloc and −lbsdmalloc are provided only in archive format and therefore programs linked with these libraries are immune to the interposition of watchmalloc.so.1.

FILES

/usr/lib/watchmalloc.so.1

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

modified 27 Feb 1997

SunOS 5.6

3X-1779
SEE ALSO proc(1), bsdmalloc(3X), calloc(3C), free(3C), malloc(3C), malloc(3X), mapmalloc(3X), memalign(3C), realloc(3C), valloc(3C), libmapmalloc(4), proc(4), attributes(5)
NAME
wcscoll, wscoll – wide character string comparison using collating information

SYNOPSIS
#include <wchar.h>

int wcscoll(const wchar_t *ws1, const wchar_t *ws2);
int wscoll(const wchar_t *ws1, const wchar_t *ws2);

DESCRIPTION
The wcscoll() and wscoll() functions compare the wide character string pointed to by ws1 to the wide character string pointed to by ws2, both interpreted as appropriate to the LC_COLLATE category of the current locale.

RETURN VALUES
Upon successful completion, wcscoll() and wscoll() return an integer greater than, equal to, or less than 0, depending upon whether the wide character string pointed to by ws1 is greater than, equal to, or less than the wide character string pointed to by ws2, when both are interpreted as appropriate to the current locale. On error, wcscoll() and wscoll() may set errno, but no return value is reserved to indicate an error.

ERRORS
wcscoll() and wscoll() may fail if:
EINVAL The ws1 or ws2 arguments contain wide character codes outside the domain of the collating sequence.
ENOSYS The function is not supported.

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe with exceptions</td>
</tr>
<tr>
<td></td>
<td>Enabled</td>
</tr>
</tbody>
</table>

SEE ALSO
setlocale(3C), wcscmp(3C), wcxfrm(3C), attributes(5)

NOTES
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 wcscoll() or wscoll(), then check errno and if it is non-zero, assume an error has occurred.
wcxfrm(3C) and wcscmp(3C) should be used for sorting large lists.
wcscoll() and wscoll() can be used safely in multi-threaded applications as long as setlocale(3C) is not being called to change the locale.
wcsftime – convert date and time to wide character string

#include <wchar.h>

size_t wcsftime(wchar_t *wcs, size_t maxsize, const char *format, const struct tm *timptr);

The wcsftime() function places wide-character codes into the array pointed to by wcs as controlled by the string pointed to by format.

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 maxsize wide-character codes.

If copying takes place between objects that overlap, the behavior is undefined.

If the total number of resulting wide character codes (including the terminating null wide-character code) is no more than maxsize, wcsftime() returns the number of wide-character codes placed into the array pointed to by wcs, not including the terminating null wide-character code. Otherwise, 0 is returned and the contents of the array are indeterminate.

wcsftime() uses malloc(3C) and should malloc() fail, errno will be set by malloc().

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSI</td>
<td>Enabled</td>
</tr>
</tbody>
</table>

SEE ALSO malloc(3C), mbstowcs(3C), setlocale(3C), strftime(3C), attributes(5)
wcstod, wstod, watof — convert wide character string to double-precision number

#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);

DESCRIPTION

The wcstod() and wstod() functions convert the initial portion of the wide character string pointed to by nptr 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(3C)); a subject sequence interpreted as a floating-point constant; and a final wide-character string of one or more unrecognised wide-character codes, including the terminating null wide character code of the input wide character string. They then attempt to convert the subject sequence to a floating-point number, and return the result.

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.

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 endptr, provided that endptr is not a null pointer.

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 (.). In other than the POSIX locale, other 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.

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 `wcstod()` may fail if:

- `EINVAL` No conversion could be performed.

ATTRIBUTES  

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  

`iswspace(3C)`, `localeconv(3C)`, `scanf(3S)`, `setlocale(3C)`, `wcstol(3C)`, `attributes(5)`

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
#include <wchar.h>
long int wcstol(const wchar_t *nptr, wchar_t **endptr, int base);
#include <widec.h>
long int wstol(const wchar_t *nptr, wchar_t **endptr, int base);
long watol(wchar_t *nptr);
long long watoll(wchar_t *nptr);
int watoi(wchar_t *nptr);

DESCRIPTION
The wcstol() and wstol() functions convert the initial portion of the wide character string pointed to by nptr 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(3C)), a subject sequence interpreted as an integer represented in some radix determined by the value of base; 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.

If the value of base 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.

If the value of base 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 base, 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 base are permitted. If the value of base 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.

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.

If the subject sequence has the expected form and the value of base is 0, the sequence of wide-character codes starting with the first digit is interpreted as an integer constant. If the subject sequence has the expected form and the value of base is between 2 and 36, it is used as the base for conversion, ascribing to each letter its value as given above. If the
subject sequence begins with a minus sign (-), the value resulting from the conversion is negated. A pointer to the final wide character string is stored in the object pointed to by endptr, provided that endptr is not a null pointer.

In other than the POSIX locale, additional implementation-dependent subject sequence forms may be accepted.

If the subject sequence is empty or does not have the expected form, no conversion is performed; the value of nptr is stored in the object pointed to by endptr, provided that endptr is not a null pointer.

The watol() function is equivalent to wstol(str, (wchar_t **)NULL, 10).

The watoll() function is the long-long (double long) version of watol(). The watoi() function is equivalent to (int)watol().

RETURN VALUES
Upon successful completion, wcstol() and wstol() return the converted value, if any. If no conversion could be performed, 0 is returned, and errno may be set to indicate the error. If the correct value is outside the range of representable values, {LONG_MAX} or {LONG_MIN} is returned (according to the sign of the value), and errno is set to ERANGE.

ERRORS
The wcstol() and wstol() functions will fail if:

EINVAL  The value of base is not supported.
ERANGE  The value to be returned is not representable.

The wcstol() and wstol() functions may fail if:

EINVAL  No conversion could be performed.

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
iswalpha(3C), iswspace(3C), scanf(3S), wcstod(3C), attributes(5)

NOTES
Because 0, {LONG_MIN}, and {LONG_MAX} are returned on error and are also valid returns on success, an application wishing to check for error situations should set errno to 0, call wcstol() or wstol(), then check errno and if it is non-zero assume an error has occurred.

Truncation from long long to long can take place upon assignment or by an explicit cast.

3C-1786 SunOS 5.6 modified 14 Jan 1996
NAME  
wcstombs – convert a wide-character string to a character string

SYNOPSIS  
#include <stdlib.h>  
size_t wcstombs(char *s, const wchar_t *pwcs, size_t n);

DESCRIPTION  
The wcstombs() function converts the sequence of wide-character codes from the array
pointed to by pwcs into a sequence of characters and stores these characters into the array
pointed to by s, stopping if a character would exceed the limit of n total bytes or if a null
byte is stored. Each wide-character code is converted as if by a call to wctomb(3C).
The behavior of this function is affected by the LC_CTYPE category of the current locale.
No more than n bytes will be modified in the array pointed to by s. If copying takes place
between objects that overlap, the behavior is undefined. If s is a null pointer, wcstombs() returns
the length required to convert the entire array regardless of the value of n, but no
values are stored.

RETURN VALUES  
If a wide-character code is encountered that does not correspond to a valid character (of
one or more bytes each), wcstombs() returns (size_t)−1. Otherwise, wcstombs() returns
the number of bytes stored in the character array, not including any terminating NULL
byte. The array will not be null-terminated if the value returned is n.

ERRORS  
The wcstombs() function may fail if the following error is detected:
EILSEC A wide-character code does not correspond to a valid character.

ATTRIBUTES  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
<tr>
<td>CSI</td>
<td>Enabled</td>
</tr>
</tbody>
</table>

SEE ALSO  
mblen(3C), mbstowcs(3C), mbtowc(3C), setlocale(3C), wctomb(3C), attributes(5)

modified 20 Dec 1996  SunOS 5.6  3C-1787
NAME
wcstoul – convert wide character string to unsigned long

SYNOPSIS
#include <wchar.h>

unsigned long int wcstoul(const wchar_t *nptr, wchar_t **endptr, int base);

DESCRIPTION
The wcstoul() function converts the initial portion of the wide character string pointed to by nptr 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(3C)); a subject sequence interpreted as an integer represented in some radix determined by the value of base; and a final wide-character string of one or more unrecognized wide character codes, including the terminating null wide-character code of the input wide character string. It then attempts to convert the subject sequence to an unsigned integer, and returns the result.

If the value of base 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.

If the value of base 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 base, 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 base are permitted. If the value of base is 16, the wide-character codes `0x' or `0X' may optionally precede the sequence of the decimal digits and letters `a' (or `A') to `f' (or `F'), with values 10 to 15, respectively.

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.

If the subject sequence has the expected form and the value of base is 0, the sequence of wide-character codes starting with the first digit is interpreted as an integer constant. If the subject sequence has the expected form and the value of base is between 2 and 36, it is used as the base for conversion, ascribing to each letter its value as given above. If the subject sequence begins with a minus sign, the value resulting from the conversion is negated. A pointer to the final wide character string is stored in the object pointed to by endptr, provided that endptr is not a null pointer.

In other than the POSIX locale, additional implementation-dependent subject sequence forms may be accepted.
If the subject sequence is empty or does not have the expected form, no conversion is performed; the value of `nptr` is stored in the object pointed to by `endptr`, provided that `endptr` is not a null pointer.

**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.
- **ERANGE** The value to be returned is not representable.

`wcstoul()` function may fail if:

- **EINVAL** No conversion could be performed.

**ATTRIBUTES**

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**SEE ALSO**

`isspace(3C), iswalpha(3C), scanf(3S), wcstod(3C), wcstol(3C), attributes(5)`

**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(3C)` and `wcstol(3C)`, `wcstoul()` must always return a non-negative number; so, using the return value of `wcstoul()` for out-of-range numbers with `wcstoul()`.
NAME
wcstring, wcscat, wscat, wcsncat, wnsncat, wcsncmp, wnsncmp, wcscpy, wcsncpy, wcscpy, wcsncpy, wcslen, wslen, wcschr, wscr, wcsrchr, wcschr, windex, wrindex, wcspbrk, wspbrk, wcswcs, wcsspn, wcspspn, wcspn, wcstok, wstok – wide character string operations

SYNOPSIS
#include <wchar.h>

wchar_t *wcscat(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 *wcsncat(wchar_t *ws1, const wchar_t *ws2, size_t n);
int wcscmp(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 *wcscpy(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);
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, wchar_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 *wcspbrk(const wchar_t *ws1, const wchar_t *ws2);
wchar_t *wspbrk(const wchar_t *ws1, const wchar_t *ws2);
wchar_t *wcswcs(const wchar_t *ws1, const wchar_t *ws2);
size_t wcspn(const wchar_t *ws1, const wchar_t *ws2);
size_t wsspn(const wchar_t *ws1, const wchar_t *ws2);
size_t wcscspn(const wchar_t *ws1, const wchar_t *ws2);
size_t wcspn(const wchar_t *ws1, const wchar_t *ws2);
wchar_t *wcstok(wchar_t *ws1, const wchar_t *ws2);
wchar_t *wstok(wchar_t *ws1, const wchar_t *ws2);

DESCRIPTION
These functions operate on wide character strings terminated by wchar_t NULL characters. During appending or copying, these routines do not check for an overflow condition of the receiving string. In the following, ws, ws1, and ws2 point to wide character strings terminated by a wchar_t NULL.

3C-1790 SunOS 5.6 modified 20 Dec 1996
The `wcscat()` and `wscat()` functions append a copy of the wide character string pointed to by `ws2` (including the terminating null wide-character code) to the end of the wide character string pointed to by `ws1`. The initial wide-character code of `ws2` overwrites the null wide-character code at the end of `ws1`. If copying takes place between objects that overlap, the behavior is undefined. Both functions return `ws1`; no return value is reserved to indicate an error.

The `wcsncat()` and `wsncat()` functions append not more than `n` wide-character codes (a null wide-character code and wide character codes that follow it are not appended) from the array pointed to by `ws2` to the end of the wide character string pointed to by `ws1`. The initial wide-character code of `ws2` overwrites the null wide-character code at the end of `ws1`. A terminating null wide-character code is always appended to the result. Both functions return `ws1`; no return value is reserved to indicate an error.

The `wcscmp()` and `wscmp()` functions compare the wide character string pointed to by `ws1` to the wide character string pointed to by `ws2`. The sign of a non-zero return value is determined by the sign of the difference between the values of the first pair of wide-character codes that differ in the objects being compared. Upon completion, both functions return an integer greater than, equal to, or less than zero, if the wide character string pointed to by `ws1` is greater than, equal to, or less than the wide character string pointed to by `ws2`.

The `wcsncmp()` and `wsncmp()` functions compare not more than `n` wide-character codes (wide-character codes that follow a null wide character code are not compared) from the array pointed to by `ws1` to the array pointed to by `ws2`. The sign of a non-zero return value is determined by the sign of the difference between the values of the first pair of wide-character codes that differ in the objects being compared. Upon successful completion, both functions return an integer greater than, equal to, or less than zero, if the possibly null-terminated array pointed to by `ws1` is greater than, equal to, or less than the possibly null-terminated array pointed to by `ws2`.

The `wcscpy()` and `wscpy()` functions copy the wide character string pointed to by `ws2` (including the terminating null wide-character code) into the array pointed to by `ws1`. If copying takes place between objects that overlap, the behavior is undefined. Both functions return `ws1`; no return value is reserved to indicate an error.

The `wcsncpy()` and `wsncpy()` functions copy not more than `n` wide-character codes (wide-character codes that follow a null wide character code are not copied) from the array pointed to by `ws2` to the array pointed to by `ws1`. If copying takes place between objects that overlap, the behavior 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.

modified 20 Dec 1996
The `wcslen()` and `wslen()` functions compute the number of wide-character codes in the wide character string to which `ws` points, not including the terminating null wide-character code. Both functions return `ws`; no return value is reserved to indicate an error.

The `wcschr()` and `wschr()` functions locate the first occurrence of `wc` in the wide character string pointed to by `ws`. The value of `wc` 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 `wc` is not found.

The `wcsrchr()` and `wsrchr()` functions locate the last occurrence of `wc` in the wide character string pointed to by `ws`. The value of `wc` 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 `wc` does not occur in the wide character string.

The `windex()` and `wrindex()` functions behave the same as `wschr()` and `wsrchr()`, respectively.

The `wcsbrk()` and `wspbrk()` functions locate the first occurrence in the wide character string pointed to by `ws1` of any wide-character code from the wide character string pointed to by `ws2`. Upon successful completion, the function returns a pointer to the wide-character code, or a null pointer if no wide-character code from `ws2` occurs in `ws1`.

The `wcswcs()` function locates the first occurrence in the wide character string pointed to by `ws1` of the sequence of wide-character codes (excluding the terminating null wide-character code) in the wide character string pointed to by `ws2`. 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 `ws2` points to a wide character string with zero length, the function returns `ws1`.

The `wcspn()` and `wspn()` functions compute the length of the maximum initial segment of the wide character string pointed to by `ws1` which consists entirely of wide-character codes from the wide string pointed to by `ws2`. Both functions return `ws1`; no return value is reserved to indicate an error.

The `wcspn()` and `wspn()` functions compute the length of the maximum initial segment of the wide character string pointed to by `ws1` which consists entirely of wide-character codes not from the wide character string pointed to by `ws2`. Both functions return `ws1`; no return value is reserved to indicate an error.

A sequence of calls to the `wcstok()` and `wstok()` functions break the wide character string pointed to by `ws1` into a sequence of tokens, each of which is delimited by a wide-character code from the wide character string pointed to by `ws2`. The first call in the sequence sets `ws1` to the location following the last token found in the wide character string pointed to by `ws1`.
sequence has \textit{ws1} as its first argument, and is followed by calls with a null pointer as their first argument. The separator string pointed to by \textit{ws2} may be different from call to call. The first call in the sequence searches the wide character string pointed to by \textit{ws1} for the first wide-character code that is \textit{not} contained in the current separator string pointed to by \textit{ws2}. If no such wide-character code is found, then there are no tokens in the wide character string pointed to by \textit{ws1}, and \texttt{wcstok()} and \texttt{wstok()} return a null pointer. If such a wide-character code is found, it is the start of the first token. \texttt{wcstok()} and \texttt{wstok()} then search from that point for a wide-character code that \textit{is} contained in the current separator string. If no such wide-character code is found, the current token extends to the end of the wide character string pointed to by \textit{ws1}, and subsequent searches for a token will return a null pointer. If such a wide-character code is found, it is overwritten by a null wide character, which terminates the current token. \texttt{wcstok()} and \texttt{wstok()} save a pointer to the following wide-character code, from which the next search for a token will start. Each subsequent call, with a null pointer as the value of the first argument, starts searching from the saved pointer and behaves as described above. Upon successful completion, both functions return a pointer to the first wide-character code of a token. Otherwise, if there is no token, a null pointer is returned.

\begin{table}[h]
\centering
\begin{tabular}{|l|l|}
\hline
\textbf{ATTRIBUTE TYPE} & \textbf{ATTRIBUTE VALUE} \\
\hline
MT-Level & MT-Safe \\
CSI & Enabled \\
\hline
\end{tabular}
\caption{Attributes for wcstok(3C) and wstok(3C)}
\end{table}

\textbf{SEE ALSO} \texttt{malloc(3C)}, \texttt{string(3C)}, \texttt{wcswidth(3C)}, \texttt{wcwidth(3C)}, \texttt{attributes(5)}

modified 20 Dec 1996

SunOS 5.6

3C-1793
NAME  wcswidth – number of column positions of a wide-character string

SYNOPSIS  
#include <wchar.h>

int wcswidth (const wchar_t *pwcs, size_t n);

DESCRIPTION  The wcswidth() function determines the number of column positions required for \( n \) wide-character codes (or fewer than \( n \) wide-character codes if a null wide-character code is encountered before \( n \) wide-character codes are exhausted) in the string pointed to by \( pwcs \).

RETURN VALUES  The wcswidth() function either returns 0 (if \( pwcs \) points to a null wide-character code), or returns the number of column positions to be occupied by the wide-character string pointed to by \( pwcs \), or returns \(-1\) (if any of the first \( n \) wide-character codes in the wide-character string pointed to by \( pwcs \) is not a printing wide-character code).

ERRORS  No errors are defined.

ATTRIBUTES  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe with exceptions</td>
</tr>
<tr>
<td>CSI</td>
<td>Enabled</td>
</tr>
</tbody>
</table>

SEE ALSO  setlocale(3C), wcwidth(3C), attributes(5)
NAME  wcsxfrm, wsxfrm – wide character string transformation

SYNOPSIS  

```
#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);
```

DESCRIPTION  The wcsxfrm() and wsxfrm() functions transform the wide character string pointed to by ws2 and place the resulting wide character string into the array pointed to by ws1. The transformation is such that if either the wcscmp(3C) or wscmp(3C) functions are applied to two transformed wide strings, they return a value greater than, equal to, or less than 0, corresponding to the result of the wcscoll(3C) or wscoll(3C) function applied to the same two original wide character strings. No more than n wide-character codes are placed into the resulting array pointed to by ws1, including the terminating null wide-character code. If n is 0, ws1 is permitted to be a null pointer. If copying takes place between objects that overlap, the behavior is undefined.

RETURN VALUES  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 n or more, the contents of the array pointed to by ws1 are indeterminate.

On error, wcsxfrm() and wsxfrm() return (size_t)−1, and set errno to indicate the error.

ERRORS  wcsxfrm() and wsxfrm() may fail if:

- **EINVAL**  The wide character string pointed to by ws2 contains wide-character codes outside the domain of the collating sequence.
- **ENOSYS**  The function is not supported.

ATTRIBUTES  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe with exceptions</td>
</tr>
<tr>
<td>CSI</td>
<td>Enabled</td>
</tr>
</tbody>
</table>

SEE ALSO  setlocale(3C), wcscmp(3C), wcscoll(3C), wscmp(3C), wscoll(3C), attributes(5)

NOTES  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).

The fact that when n is 0, ws1 is permitted to be a null pointer, is useful to determine the size of the ws1 array prior to making the transformation.

Because no return value is reserved to indicate an error, an application wishing to check for error situations should set errno to 0, call wcsxfrm() or wsxfrm(), then check errno and if it is non-zero, assume an error has occurred.

modified 20 Dec 1996  SunOS 5.6  3C-1795
wcsxfrm() and wszxfrm() can be used safely in multi-threaded applications as long as `setlocale(3C)` is not being called to change the locale.
NAME  
wctomb – convert a wide-character code to a character

SYNOPSIS  
#include <stdlib.h>

int wctomb(char *s, wchar_t wchar);

DESCRIPTION  
The wctomb() function determines the number of bytes needed to represent the character corresponding to the wide-character code whose value is wchar. It stores the character representation (possibly multiple bytes) in the array object pointed to by s (if s is not a null pointer). At most MB_CUR_MAX bytes are stored.

A call with s as a null pointer causes this function to return 0. The behavior of this function is affected by the LC_CTYPE category of the current locale.

RETURN VALUES  
If s is a null pointer, wctomb() returns 0 value. If s is not a null pointer, wctomb() returns −1 if the value of wchar does not correspond to a valid character, or returns the number of bytes that constitute the character corresponding to the value of wchar.

In no case will the value returned be greater than the value of the MB_CUR_MAX macro.

ERRORS  
No errors are defined.

ATTRIBUTES  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe with exceptions</td>
</tr>
<tr>
<td>CSI</td>
<td>Enabled</td>
</tr>
</tbody>
</table>

SEE ALSO  
mblen(3C), mbstowcs(3C), mbtowc(3C), setlocale(3C), wcstombs(3C), attributes(5)

NOTES  
The wctomb() function can be used safely in a multi-thread application, as long as setlocale(3C) is not being called to change the locale.

modified 20 Dec 1996
NAME
wctrans – define wide-character mapping

SYNOPSIS
#include <wctype.h>

wctrans_t wctrans(const char *property);

DESCRIPTION
The wctrans() function constructs a value with type wctrans_t that describes a mapping between wide characters identified by the string argument property.

tolower and toupper shall be valid in all locales as property arguments to the wctrans() function.

RETURN VALUES
If property identifies a valid mapping of wide characters according the LC_CTYPE category of the current locale, the wctrans() function returns a nonzero value that is valid as the second argument to the towctrans(3C) function; otherwise, it returns 0.

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe with exceptions</td>
</tr>
<tr>
<td>CSI</td>
<td>Enabled</td>
</tr>
</tbody>
</table>

SEE ALSO
setlocale(3C), towctrans(3C), attributes(5)
NAME wctype – define character class

SYNOPSIS
#include <wchar.h>

wctype_t wctype(const char *charclass);

DESCRIPTION The wctype() function is defined for valid character class names as defined in the current locale. The charclass 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
- xdigit

Additional character class names defined in the locale definition file (category LC_CTYPE) can also be specified.

The function returns a value of type wctype_t, which can be used as the second argument to subsequent calls of iswctype(3C). 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.

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().

ATTRIBUTES See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe with exceptions</td>
</tr>
<tr>
<td>CSI</td>
<td>Enabled</td>
</tr>
</tbody>
</table>

SEE ALSO iswctype(3C), setlocale(3C), attributes(5)
NAME
wcwidth – number of column positions of a wide-character code

SYNOPSIS
#include <wchar.h>

int wcwidth (wchar_t wc);

DESCRIPTION
The wcwidth() function determines the number of column positions required for the
wide character wc. The value of wc must be a character representable as a wchar_t, and
must be a wide-character code corresponding to a valid character in the current locale.

RETURN VALUES
The wcwidth() function either returns 0 (if wc is a null wide-character code), or returns
the number of column positions to be occupied by the wide-character code wc, or returns
−1 (if wc does not correspond to a printing wide-character code).

ERRORS
No errors are defined.

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe with exceptions</td>
</tr>
<tr>
<td>CSI</td>
<td>Enabled</td>
</tr>
</tbody>
</table>

SEE ALSO
setlocale(3C), wcswidth(3C), attributes(5)
NAME wordexp, wordfree – perform word expansions

SYNOPSIS
#include <wordexp.h>

int wordexp(const char ∗ words, wordexp_t ∗ pwordexp, int flags);
void wordfree(wordexp_t ∗ pwordexp);

DESCRIPTION
The wordexp() function performs word expansions, subject to quoting, and places
the list of expanded words into the structure pointed to by pwordexp.

The wordfree() function frees any memory allocated by wordexp() associated with pwordexp.

words Argument
The words argument is a pointer to a string containing one or more words to be
expanded. The expansions will be the same as would be performed by the shell if words
were the part of a command line representing the arguments to a utility. Therefore, words
must not contain an unquoted NEWLINE or any of the unquoted shell special characters:
| & ; < >
except in the context of command substitution. It also must not contain unquoted
parentheses or braces, except in the context of command or variable substitution. If the
argument words contains an unquoted comment character (number sign) that is the
beginning of a token, wordexp() may treat the comment character as a regular character,
or may interpret it as a comment indicator and ignore the remainder of words.

pwordexp Argument
The structure type wordexp_t is defined in the header <wordexp.h> and includes at least
the following members:

size_t we_wordc    Count of words matched by words.
char **we_wordv    Pointer to list of expanded words.
size_t we_offs     Slots to reserve at the beginning of pwordexp->we_wordv.

The wordexp() function stores the number of generated words into
pwordexp->we_wordc and a pointer to a list of pointers to words in
pwordexp->we_wordv. Each individual field created during field splitting is a separate
word in the pwordexp->we_wordv list. The words are in order. The first pointer after
the last word pointer will be a NULL pointer.

It is the caller’s responsibility to allocate the storage pointed to by pwordexp. The wordexp() function allocates other space as needed, including memory pointed to by
pwordexp->we_wordv. The wordfree() function frees any memory associated with pwordexp
from a previous call to wordexp().

flags Argument
The flags argument is used to control the behavior of wordexp(). The value of flags is the
bitwise inclusive OR of zero or more of the following constants, which are defined in
<wordexp.h>:

WRDE_APPEND Append words generated to the ones from a previous call to wordexp().
WRDE_DOOFFS Make use of pwordexp->we_offs. If this flag is set,
\texttt{pwordexp->we_offs} is used to specify how many NULL pointers to add to the beginning of \texttt{pwordexp->we_wordv}. In other words, 
\texttt{pwordexp->we_wordv} will point to \texttt{pwordexp->we_offs} NULL pointers, followed by \texttt{pwordexp->we_wordc} word pointers, followed by a NULL pointer.

\textbf{WRDE_NOCMD} Fail if command substitution is requested.

\textbf{WRDE_REUSE} The \texttt{pwordexp} argument was passed to a previous successful call to \texttt{wordexp()}, and has not been passed to \texttt{wordfree()}. The result will be the same as if the application had called \texttt{wordfree()} and then called \texttt{wordexp()} without \texttt{WRDE_REUSE}.

\textbf{WRDE_SHOWERR} Do not redirect \texttt{stderr} to /dev/null.

\textbf{WRDE_UNDEF} Report error on an attempt to expand an undefined shell variable.

The \textbf{WRDE_APPEND} flag can be used to append a new set of words to those generated by a previous call to \texttt{wordexp()}. The following rules apply when two or more calls to \texttt{wordexp()} are made with the same value of \texttt{pwordexp} and without intervening calls to \texttt{wordfree()}:

1. The first such call must not set \texttt{WRDE_APPEND}. All subsequent calls must set it.
2. All of the calls must set \texttt{WRDE_DOOFFS}, or all must not set it.
3. After the second and each subsequent call, \texttt{pwordexp->we_wordv} will point to a list containing the following:
   a. zero or more NULL pointers, as specified by \texttt{WRDE_DOOFFS} and \texttt{pwordexp->we_offs}.
   b. pointers to the words that were in the \texttt{pwordexp->we_wordv} list before the call, in the same order as before.
   c. pointers to the new words generated by the latest call, in the specified order.
4. The count returned in \texttt{pwordexp->we_wordc} will be the total number of words from all of the calls.
5. The application can change any of the fields after a call to \texttt{wordexp()}, but if it does it must reset them to the original value before a subsequent call, using the same \texttt{pwordexp} value, to \texttt{wordfree()} or \texttt{wordexp()} with the \texttt{WRDE_APPEND} or \texttt{WRDE_REUSE} flag.

If \texttt{words} contains an unquoted:

\begin{verbatim}
NEWLINE | & ; < > ( ) { }
\end{verbatim}

in an inappropriate context, \texttt{wordexp()} will fail, and the number of expanded words will be zero.

Unless \textbf{WRDE_SHOWERR} is set in \texttt{flags}, \texttt{wordexp()} will redirect \texttt{stderr} to /dev/null for any utilities executed as a result of command substitution while expanding \texttt{words}. If \textbf{WRDE_SHOWERR} is set, \texttt{wordexp()} may write messages to \texttt{stderr} if syntax errors are detected while expanding \texttt{words}. 
If WRDE_DOOFFS is set, then \texttt{wordexp->we_offs} must have the same value for each \texttt{wordexp()} call and \texttt{wordfree()} call using a given \texttt{pwordexp}.

The following constants are defined as error return values:

- \textbf{WRDE_BADCHAR} One of the unquoted characters: \texttt{NEWLINE | & ; < > ( ) { }}, appears in \texttt{words} in an inappropriate context.
- \textbf{WRDE_BADVAL} Reference to undefined shell variable when \texttt{WRDE_UNDEF} is set in flags.
- \textbf{WRDE_CMDSUB} Command substitution requested when \texttt{WRDE_NOCMD} was set in flags.
- \textbf{WRDE_NOSPACE} Attempt to allocate memory failed.
- \textbf{WRDE_SYNTAX} Shell syntax error, such as unbalanced parentheses or unterminated string.

\textbf{RETURN VALUES} The following values are returned by \texttt{wordexp()}:

- 0 successful completion.
- non-zero an error has occurred.
- \texttt{WRDE_NOSPACE} \texttt{pwordexp->we_wordc} and \texttt{pwordexp->we_wordv} will be updated to reflect any words that were successfully expanded. In other cases, they will not be modified.

The \texttt{wordfree()} function returns no value.

\textbf{USAGE} This function is intended to be used by an application that wants to do all of the shell’s expansions on a word or words obtained from a user. For example, if the application prompts for a filename (or list of filenames) and then uses \texttt{wordexp()} to process the input, the user could respond with anything that would be valid as input to the shell.

The \texttt{WRDE_NOCMD} flag is provided for applications that, for security or other reasons, want to prevent a user from executing shell commands. Disallowing unquoted shell special characters also prevents unwanted side effects such as executing a command or writing a file.

\textbf{ATTRIBUTES} See \texttt{attributes(5)} for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

\textbf{SEE ALSO} \texttt{fnmatch(3C), glob(3C), attributes(5)}
NAME
wsprintf – formatted output conversion

SYNOPSIS
#include <stdio.h>
#include <widec.h>
int wsprintf(wchar_t *s, const char *format, /* arg */ ... );

DESCRIPTION
wsprintf() outputs a Process Code string ending with a Process Code (wchar_t) NULL character. It is the user’s responsibility to allocate enough space for this wchar_t string. This returns the number of Process Code characters (excluding the NULL terminator) that have been written. The conversion specifications and behavior of 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).

RETURN VALUES
Upon success, wsprintf() returns the number of characters printed. When an error condition is encountered, a negative value is returned.

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
wsscanf(3C), printf(3S), scanf(3S), sprintf(3S), attributes(5)
NAME  

wsscanf – formatted input conversion

SYNOPSIS  

#include <stdio.h>
#include <widec.h>

int wsscanf(wchar_t *s, const char *format, /* pointer */...);

DESCRIPTION  

wsscanf() reads Process Code characters from the Process Code string s, interprets them
according to the format, and stores the results in its arguments. wsscanf() expects, as
arguments, a control string format, and a set of pointer arguments indicating where the
converted input should be stored. The results are undefined if there are insufficient args
for the format. If the format is exhausted while args remain, the excess args are simply
ignored.

The conversion specifications and behavior of 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 scanf(3S).

RETURN VALUES  

wsscanf() returns the number of characters matched. On error wsscanf() returns a nega-
tive value.

ATTRIBUTES  

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  

wsprintf(3C), printf(3S), scanf(3S), attributes(5)
NAME
wstring, wcasecmp, wsncasecmp, wsdup, wscol – Process Code string operations

SYNOPSIS
#include <widec.h>
int wscasecmp(const wchar_t *s1, const wchar_t *s2);
int wsncasecmp(const wchar_t *s1, const wchar_t *s2, int n);
wchar_t *wsdup(const wchar_t *s);
int wscol(const wchar_t *s);

DESCRIPTION
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, s, s1, and s2 point to Process Code strings terminated by a wchar_t NULL.

wcasecmp(), wsncasecmp() The wscasecmp() function compares its arguments, ignoring case, and returns an integer greater than, equal to, or less than 0, depending upon whether s1 is lexicographically greater than, equal to, or less than s2. wsncasecmp() makes the same comparison but compares at most n 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.

wsdup() The wsdup() function returns a pointer to a new Process Code string, which is a duplicate of the string pointed to by s. The space for the new string is obtained using malloc(3C). If the new string cannot be created, a null pointer is returned.

wscol() The wscol() function returns the screen display width (in columns) of the Process Code string s.

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
malloc(3C), string(3C), wcstring(3C), attributes(5)
NAME  wunctrl – convert a wide character to printable form

SYNOPSIS  
```c
#include <curses.h>

wchar_t *wunctrl(cchar_t wc);
```

ARGUMENTS  
`wc`   Is a wide character.

DESCRIPTION  
The `wunctrl()` function converts the wide character code `wc` into a printable form (if unprintable). Control characters are displayed using the ”x notation where ” identifies the control key and x represents an alphanumeric character that is pressed while the control key is held down.

Characters which have their eighth bit set are represented using the meta notation `M-X` where `X` is the byte with eighth bit stripped. This stripped byte will represent either a printable character or a control character. If it is a control character, `X` is actually represented using ”X notation. For example, "0xCD" in ASCII is M-ÃK.

RETURN VALUES  
On success, the `wunctrl()` function returns the generated string. Otherwise, it returns a null pointer.

ERRORS  
None.

SEE ALSO  
`keyname(3XC)`, `unctrl(3XC)`

modified 1 Jun 1996
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>
<thead>
<tr>
<th>XDR Routine</th>
<th>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_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>
<tr>
<td>xdr_wrapstring</td>
<td>xdr_complex(3N)</td>
</tr>
<tr>
<td>xdrmem_create</td>
<td>xdr_create(3N)</td>
</tr>
</tbody>
</table>
### xdrrec_create
- xdrrec_create
- xdrrec_endofrecord
- xdrrec_eof
- xdrrec_readbytes
- xdrrec_skiprecord
- xdrstdio_create

### ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

### SEE ALSO
- rpc(3N), xdr_admin(3N), xdr_complex(3N), xdr_create(3N), xdr_simple(3N), attributes(5)

modified 30 Dec 1996

SunOS 5.6

3N-1809
NAME  
xdr_admin, xdr_control, xdr_getpos, xdr_inline, xdrrec_endofrecord, xdrrec_eof, xdrrec_readbytes, xdrrec_skiprecord, xdr_setpos, xdr_strerror – library routines for external data representation

DESCRIPTION  
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.

These routines deal specifically with the management of the XDR stream.

Routines  
See rpc(3N) for the definition of the XDR data structure. Note that any buffers passed to the XDR routines must be properly aligned. It is suggested that malloc(3C) be used to allocate these buffers or that the programmer insure that the buffer address is divisible evenly by four.

#include <rpc/xdr.h>

bool_t xdr_control( XDR *xdrs, int req, void *info);

A function macro to change or retrieve various information about an XDR stream. req indicates the type of operation and info is a pointer to the information. The supported values of req, their argument types and what they do are:

<table>
<thead>
<tr>
<th>req</th>
<th>Argument Types</th>
<th>What They Do</th>
</tr>
</thead>
<tbody>
<tr>
<td>XDR_GETBYTES_AVAIL</td>
<td>xdr_bytesrec</td>
<td>return number of bytes left unconsumed in the stream and a flag indicating whether or not this is the last fragment.</td>
</tr>
</tbody>
</table>

u_int xdr_getpos(const XDR *xdrs);

A macro that invokes the get-position routine associated with the XDR stream, xdrs. 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.

long *xdr_inline(XDR *xdrs, const int len);

A macro that invokes the in-line routine associated with the XDR stream, xdrs. The routine returns a pointer to a contiguous piece of the stream’s buffer; len is the byte length of the desired buffer. Note: pointer is cast to long *.

Warning: xdr_inline() may return NULL (0) if it cannot allocate a contiguous piece of a buffer. Therefore the behavior may vary among stream instances; it exists for the sake of efficiency, and applications written for portability should not depend on this feature.
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.
ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO malloc(3C), rpc(3N), xdr_complex(3N), xdr_create(3N), xdr_simple(3N), attributes(5)
**NAME**

xdr_complex, xdr_array, xdr_bytes, xdr_opaque, xdr_pointer, xdr_reference, xdr_string, xdr_union, xdr_vector, xdr_wrapstring – library routines for external data representation

**DESCRIPTION**

XDR library routines allow C programmers to describe complex data structures in a machine-independent fashion. Protocols such as remote procedure calls (RPC) use these routines to describe the format of the data. These routines are the XDR library routines for complex data structures. They require the creation of XDR stream (see xdr_create(3N)).

**Routines**

See rpc(3N) for the definition of the XDR data structure. Note that any buffers passed to the XDR routines must be properly aligned. It is suggested that malloc() be used to allocate these buffers or that the programmer insure that the buffer address is divisible evenly by four.

```c
#include <rpc/xdr.h>

bool_t xdr_array(XDR *xdrs, caddr_t *arrp, u_int *sizep, const u_int maxsize, const u_int elsize, const xdrproc_t elproc);

xdr_array() translates between variable-length arrays and their corresponding external representations. The parameter arrp is the address of the pointer to the array, while sizep is the address of the element count of the array; this element count cannot exceed maxsize. The parameter elsize is the size of each of the array’s elements, and elproc is an XDR routine that translates between the array elements’ C form and their external representation. If *aarrp is null when decoding, xdr_array() allocates memory and *aarrp points to it. This routine returns TRUE if it succeeds, FALSE otherwise.

bool_t xdr_bytes(XDR *xdrs, char **sp, u_int *sizep, const u_int maxsize);

xdr_bytes() translates between counted byte strings and their external representations. The parameter sp is the address of the string pointer. The length of the string is located at address sizep; strings cannot be longer than maxsize. If *sp is null when decoding, xdr_bytes() allocates memory and *sp points to it. This routine returns TRUE if it succeeds, FALSE otherwise.

bool_t xdr_opaque(XDR *xdrs, caddr_t cp, const u_int cnt);

xdr_opaque() translates between fixed size opaque data and its external representation. The parameter cp is the address of the opaque object, and cnt is its size in bytes. This routine returns TRUE if it succeeds, FALSE otherwise.

bool_t xdr_pointer(XDR *xdrs, char **objpp, u_int objsize, const xdrproc_t xdраДobj);

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 TRUE if it succeeds, FALSE otherwise. Note: xdr_string() can be used to send an empty string (""), but not a NULL string.

bool_t xdr_union(XDR *xdrs, enum_t *dscmp, char *unp, const struct xdr_discrim *choices, const xdrproc_t (*defaultarm);

xdr_union() translates between a discriminated C union and its corresponding external representation. It first translates the discriminant of the union located at dscmp. This discriminant is always an enum_t. Next the union located at unp is translated. The parameter choices is a pointer to an array of xdr_discrim structures. Each structure contains an ordered pair of [value, proc]. If the union’s discriminant is equal to the associated value, then the proc is called to translate the union. The end of the xdr_discrim structure array is denoted by a routine of value NULL. If the discriminant is not found in the choices array, then the defaultarm procedure is called (if it is not NULL). Returns TRUE if it succeeds, FALSE otherwise.

bool_t xdr_vector(XDR *xdrs, char *arrp, const u_int size, const u_int elsze, 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 elsze is the sizeof each of the array’s elements, and elproc is an XDR routine that translates between the array elements’ C form and their external representation. This routine returns TRUE if it succeeds, FALSE otherwise.
bool_t xdr_wrapstring(XDR *xdrs, char **sp);

A routine that calls xdr_string(xdrs, sp, maxuint); where maxuint is the maximum value of an unsigned integer.

Many routines, such as xdr_array(), xdr_pointer(), and xdr_vector() take a function pointer of type xdrproc_t(), which takes two arguments. xdr_string(), one of the most frequently used routines, requires three arguments, while xdr_wrapstring() only requires two. For these routines, xdr_wrapstring() is desirable. This routine returns TRUE if it succeeds, FALSE otherwise.

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO      rpc(3N), xdr_admin(3N), xdr_create(3N), xdr_simple(3N), attributes(5)
NAME
xdr_create, xdr_destroy, xdrmem_create, xdrrec_create, xdrstdio_create – library routines for external data representation stream creation

DESCRIPTION
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.
These routines deal with the creation of XDR streams. XDR streams have to be created before any data can be translated into XDR format.

Routines
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.

```c
#include <rpc/xdr.h>

void xdr_destroy(XDR *xdrs);
    A macro that invokes the destroy routine associated with the XDR stream, xdrs. Destruction usually involves freeing private data structures associated with the stream. Using xdrs after invoking xdr_destroy() is undefined.

void xdrmem_create(XDR *xdrs, const caddr_t addr, const u_int size, const enum xdr_op op);
    This routine initializes the XDR stream object pointed to by xdrs. The stream’s data is written to, or read from, a chunk of memory at location addr whose length is no less than size bytes long. The op determines the direction of the XDR stream (either XDR_ENCODE, XDR_DECODE, or XDR_FREE).

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));
    This routine initializes the read-oriented XDR stream object pointed to by xdrs. The stream’s data is written to a buffer of size sendsz; a value of 0 indicates the system should use a suitable default. The stream’s data is read from a buffer of size recvsz; it too can be set to a suitable default by passing a 0 value. When a stream’s output buffer is full, writeit is called. Similarly, when a stream’s input buffer is empty, readit 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 (read_handle or write_handle) is passed to the former routines as the first parameter instead of a file descriptor. Note: the XDR stream’s op field must be set by the caller.
Warning: this XDR stream implements an intermediate record stream. Therefore there are additional bytes in the stream to provide record boundary information.

```c
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.

**ATTRIBUTES**

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**SEE ALSO**

`read(2), write(2), malloc(3C), rpc(3N), xdr_admin(3N), xdr_complex(3N), xdr_simple(3N), fclose(3S), attributes(5)`
xdr_simple (3N)  Network Functions

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

DESCRIPTION

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.

These routines require the creation of XDR streams (see xdr_create(3N)).

Routines

See rpc(3N) for the definition of the XDR data structure. Note that any buffers passed to the XDR routines must be properly aligned. It is suggested that malloc(3C) be used to allocate these buffers or that the programmer insure that the buffer address is divisible evenly by four.

#include <rpc/xdr.h>

bool_t xdr_bool(XDR *xdrs, bool_t *bp);

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.

bool_t xdr_char(XDR *xdrs, char *cp);

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(), xdrOpaque(), or xdr_string() (see xdr_complex(3N)).

bool_t xdr_double(XDR *xdrs, double *dp);

xdr_double() translates between C double precision numbers and their external representations. This routine returns TRUE if it succeeds, FALSE otherwise.

bool_t xdr_enum(XDR *xdrs, enum_t *ep);

xdr_enum() translates between C enums (actually integers) and their external representations. This routine returns TRUE if it succeeds, FALSE otherwise.

bool_t xdr_float(XDR *xdrs, float *fp);

xdr_float() translates between C floats and their external representations. This routine returns TRUE if it succeeds, FALSE otherwise.
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.

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO malloc(3C), rpc(3N), xdr_admin(3N), xdr_complex(3N), xdr_create(3N), attributes(5)
NAME  
xfn – overview of the XFN interface

DESCRIPTION  
The primary service provided by a federated naming system is to map a composite name to a reference. A composite name is composed of name components from one or more naming systems. A reference consists of one or more communication end points. An additional service provided by a federated naming system is to provide access to attributes associated with named objects. This extension is to satisfy most applications’ additional naming service needs without cluttering the basic naming service model. XFN is a programming interface for a federated naming service.

To use the XFN interface, include the xfn/xfn.h header file and link the application with -lxfn.

The xfn/xfn.h header file contains the interface declarations for:

• the XFN base context interface,
• the XFN base attribute interface,
• status object and status codes used by operations in these two interfaces,
• abstract data types passed as parameters to and returned as values from operations in these two interfaces, and
• the interface for the XFN standard syntax model for parsing compound names.

FILES  
/usr/include/xfn/xfn.h

SEE ALSO  
FN_ctx_t(3N), FN_status_t(3N), xfn_attributes(3N), xfn_composite_names(3N), xfn_compound_names(3N), xfn_status_codes(3N), fns(5), fns_policies(5)

NOTES  
The implementation of XFN in this Solaris release is based on the X/Open preliminary specification. It is likely that there will be minor changes to these interfaces to reflect changes in the final version of this specification. The next minor release of Solaris will offer binary compatibility for applications developed using the current interfaces. As the interfaces evolve toward standardization, it is possible that future releases of Solaris will require minor source code changes to applications that have been developed against the preliminary specification.

modified 4 Nov 1994  SunOS 5.6  3N-1821
### NAME

xfn_attributes – an overview of XFN attribute operations

### DESCRIPTION

XFN assumes the following model for attributes. A set of zero or more attributes is associated with a named object. Each attribute in the set has a unique attribute identifier, an attribute syntax, and a (possibly empty) set of distinct data values. Each attribute value has an opaque data type. The attribute identifier serves as a name for the attribute. The attribute syntax indicates how the value is encoded in the buffer.

The operations of the base attribute interface may be used to examine and modify the settings of attributes associated with existing named objects. These objects may be contexts or other types of objects. The attribute operations do not create names or remove names from contexts.

The range of support for attribute operations may vary widely. Some naming systems may not support any attribute operations. Other naming systems may only support read operations, or operations on attributes whose identifiers are in some fixed set. A naming system may limit attributes to have a single value, or may require at least one value. Some naming systems may only associate attributes with context objects, while others may allow associating attributes with non-context objects.

These are the interfaces:

```c
#include <xfn/xfn.h>

FN_attribute_t *fn_attr_get(FN_ctx_t *ctx, const FN_composite_name_t *name,
    const FN_identifier_t *attribute_id, FN_status_t *status);

int fn_attr_modify(FN_ctx_t *ctx, const FN_composite_name_t *name,
    unsigned int mod_op, const FN_attribute_t *attr, FN_status_t *status);

FN_attrset_t *fn_attr_get_ids(FN_ctx_t *ctx, const FN_composite_name_t *name,
    FN_status_t *status);

FN_valuelist_t *fn_attr_get_values(FN_ctx_t *ctx, const FN_composite_name_t *name,
    const FN_identifier_t *attribute_id, FN_status_t *status);

FN_attribute_t *fn_valuelist_next(FN_valuelist_t *vl, FN_identi®er_t **attr_syntax,
    FN_status_t *status);

void fn_valuelist_destroy(FN_valuelist_t *vl, FN_status_t *status);

FN_multigetlist_t *fn_attr_multi_get(FN_ctx_t *ctx,
    const FN_composite_name_t *name, const FN_attrset_t *attr_ids,
    FN_status_t *status);

FN_attribute_t *fn_multigetlist_next(FN_multigetlist_t *ml, FN_status_t *status);

void fn_multigetlist_destroy(FN_multigetlist_t *ml, FN_status_t *status);

int fn_attr_multi_modify(FN_ctx_t *ctx, const FN_composite_name_t *name,
    const FN_attrmodlist_t *mods, FN_status_t *status,
    FN_attrmodlist_t **unexecuted_mods);
```
FN_attrset_t *fn_ctx_get_syntaxAttrs(FN_ctx_t *ctx,
    const FN_composite_name_t *name, FN_status_t *status);

The following describes briefly the operations in the base attribute interface. Detailed
descriptions are given in the respective reference manual pages for these operations.

fn_attr_get() returns the attribute identified. fn_attr_modify() modifies the attribute
identified as described by mod_op.

fn_attr_get_ids() returns the identifiers of the attributes of the named object.

fn_attr_get_values() and its set of related operations are used for returning the individ-
ual values of an attribute.

fn_attr_multi_get() and its set of related operations are used for returning the requested
attributes associated with the named object. fn_attr_multi_modify() modifies multiple
attributes associated with the named object in a single invocation.

fn_ctx_get_syntax_attrs() returns the syntax attributes associated with the named con-
text.

ERRORS status is set as described in FN_status_t(3N) and xfn_status_codes(3N). The following
status codes are of special relevance to attribute operations:

FN_E_ATTR_VALUE_REQUIRED
    The operation attempted to create an attribute without a value, and the specific
    naming system does not allow this.

FN_E_ATTR_NO_PERMISSION
    The caller did not have permission to perform the attempted attribute operation.

FN_E_INSUFFICIENT_RESOURCES
    There are insufficient resources to retrieve the requested attribute(s).

FN_E_INVALID_ATTR_IDENTIFIER
    The attribute identifier was not in a format acceptable to the naming system, or
    its contents was not valid for the format specified for the identifier.

FN_E_INVALID_ATTR_VALUE
    One of the values supplied was not in the appropriate form for the given attri-
bute.

FN_E_NO_SUCH_ATTRIBUTE
    The object did not have an attribute with the given identifier.

FN_E_TOO_MANY_ATTR_VALUES
    The operation attempted to associate more values with an attribute than the nam-
ing system supported.

USAGE Except for fn_ctx_get_syntax_attrs(), an attribute operation using a composite name is
not necessarily equivalent to an independent fn_ctx_lookup() operation followed by an
attribute operation in which the caller supplies the resulting reference and an empty
name. This is because there is a range of attribute models in which an attribute is associ-
ated with a name in a context, or an attribute is associated with the object named, or both.
XFN accommodates all of these alternatives. Invoking an attribute operation using the
target context and the terminal atomic name accesses either the attributes that are associated with the target name or target named object; this is dependent on the underlying attribute model. This document uses the term attributes associated with a named object to refer to all of these cases.

XFN specifies no guarantees about the relationship between the attributes and the reference associated with a given name. Some naming systems may store the reference bound to a name in one or more attributes associated with a name. Attribute operations might affect the information used to construct a reference.

To avoid undefined results, programmers must use the operations in the context interface and not attribute operations when the intention is to manipulate a reference. Programmers should avoid the use of specific knowledge about how an XFN context implementation over a particular naming system constructs references.

SEE ALSO

FN_attribute_t(3N), FN_attrset_t(3N), FN_attrvalue_t(3N), FN_composite_name_t(3N), FN_ctx_t(3N), FN_identifier_t(3N), FN_status_t(3N), fn_attr_get(3N), fn_attr_get_ids(3N), fn_attr_get_values(3N), fn_attr_modify(3N), fn_attr_multi_get(3N), fn_attr_multi_modify(3N), fn_ctx_get_syntax_attrs(3N), fn_ctx_lookup(3N), xfn_status_codes(3N)

NOTES

The implementation of XFN in this Solaris release is based on the X/Open preliminary specification. It is likely that there will be minor changes to these interfaces to reflect changes in the final version of this specification. The next minor release of Solaris will offer binary compatibility for applications developed using the current interfaces. As the interfaces evolve toward standardization, it is possible that future releases of Solaris will require minor source code changes to applications that have been developed against the preliminary specification.
NAME  xfn_composite_names – XFN composite syntax: an overview of the syntax for XFN composite name

DESCRIPTION  An XFN composite name consists of an ordered list of zero or more components. Each component is a string name from the namespace of a single naming system. It may be an atomic or a compound name in that namespace.

XFN defines an abstract data type, FN_composite_name_t, for representing the structural form of a composite name. XFN also defines a standard string form for composite names. This form is the concatenation of the components of a composite name from left to right with the XFN component separator ('/') character to separate each component.

These are the interfaces:

```
#include <xfn/xfn.h>
FN_composite_name_t *fn_composite_name_from_string( const FN_string_t *str);
FN_string_t *fn_string_from_composite_name( const FN_composite_name_t *name);
```

The function fn_composite_name_from_string parses the string representation of a composite name into its corresponding composite name object FN_composite_name_t. The function fn_string_from_composite_name composes the string representation of a composite name given its composite name object form FN_composite_name_t.

APPLICATION USAGE  Special characters used in the XFN composite name syntax, such as the separator or escape characters, have the same encoding as they would in ISO 646.

All XFN implementations are required to support the portable representation, ISO 646. All other representations are optional.

All characters of the string form of a XFN composite name use a single encoding. This does not preclude component names of a composite name in its structural form from having different encodings. Code set mismatches that occur during the process of converting a composite name structure to its string form are resolved in an implementation-dependent way. When an implementation discovers that a composite name has components with incompatible code sets, it returns the error code FN_E_INCOMPATIBLE_CODE_SETS.

SEE ALSO  FN_string_t(3N), FN_compound_name_t(3N), xfn(3N)
NAME | xfn_compound_names – XFN compound syntax: an overview of XFN model for compound name parsing

DESCRIPTION | Each naming system in an XFN federation has a naming convention. XFN defines a standard model of expressing compound name syntax that covers a large number of specific name syntaxes and is expressed in terms of syntax properties of the naming convention.

The model uses the attributes in the following table to describe properties of the syntax. Unless otherwise qualified, these syntax attributes have attribute identifiers that use the FN_ID_STRING format. A context that supports the XFN standard syntax model has an attribute set containing the fn_syntax_type attribute with the value “standard” (ASCII attribute syntax).

These are the interfaces:

```c
#include <xfn/xfn.h>
FN_attrset_t *fn_ctx_get_syntax_attrs(FN_ctx_t *ctx, const FN_composite_name_t *name, FN_status_t *status);
FN_compound_name_t *fn_compound_name_from_syntax_attrs(const FN_attrset_t *aset, const FN_string_t *name, FN_status_t *status);
```

fn_syntax_type
Its value is the ASCII string "standard" if the context supports the XFN standard syntax model. Its value is an implementation-specific value if another syntax model is supported.

fn_std_syntax_direction
Its value is an ASCII string, one of "left_to_right", "right_to_left", or "flat". This determines whether the order of components in a compound name string goes from left to right, right to left, or whether the namespace is flat (in other words, not hierarchical; em all names are atomic).

fn_std_syntax_separator
Its value is the separator string for this name syntax. This attribute is required unless the fn_std_syntax_direction is "flat".

fn_std_syntax_escape
If present, its value is the escape string for this name syntax.

fn_std_syntax_case_insensitive
If this attribute is present, it indicates that names that differ only in case are considered identical. If this attribute is absent, it indicates that case is significant. If a value is present, it is ignored.

fn_std_syntax_begin_quote
If present, its value is the begin-quote string for this syntax. There can be multiple values for this attribute.

fn_std_syntax_end_quote
If present, its value is the end-quote string for this syntax. There can be multiple values for this attribute.

3N-1826 | SunOS 5.6 | modified 4 Nov 1994
fn_std_syntax_av separator
If present, its value is the attribute value assertion separator string for this syntax.

fn_std_syntax_typeval_separator
If present, its value is the attribute type-value separator string for this syntax.

fn_std_syntax_code_sets
If present, its value identifies the code sets of the string representation for this syntax. Its value consists of a structure containing an array of code sets supported by the context; the first member of the array is the preferred code set of the context. The values for the code sets are defined in the X/Open code set registry. If this attribute is not present, or if the value is empty, the default code set is ISO 646 (same encoding as ASCII).

fn_std_syntax_locale_info
If present, identifies locale information, such as character set information, of the string representation for this syntax. The interpretation of its value is implementation-dependent.

The XFN standard syntax attributes are interpreted according to the following rules:
1. In a string without quotes or escapes, any instance of the separator string delimits two atomic names.
2. A separator, quotation or escape string is escaped if preceded immediately (on the left) by the escape string.
3. A non-escaped begin-quote which precedes a component must be matched by a non-escaped end-quote at the end of the component. Quotes embedded in non-quoted names are treated as simple characters and do not need to be matched. An unmatched quotation fails with the status code FN_E_ILLEGAL_NAME.
4. If there are multiple values for begin-quote and end-quote, a specific begin-quote value must be matched with its corresponding end-quote value.
5. When the separator appears between a (non-escaped) begin quote and the end quote, it is ignored.
6. When the separator is escaped, it is ignored. An escaped begin-quote or end-quote string is not treated as a quotation mark. An escaped escape string is not treated as an escape string.
7. A non-escaped escape string appearing within quotes is interpreted as an escape string. This can be used to embed an end-quote within a quoted string.

After constructing a compound name from a string, the resulting component atoms have one level of escape strings and quotations interpreted and consumed.

fn_ctx_get_syntax_attrs() is used to obtain the syntax attributes associated with a context.

fn_compound_name_from_syntax() is used to construct a compound name object using the string form of the name and the syntax attributes of the name.
### ERRORS

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FN_E_ILLEGAL_NAME</td>
<td>The name supplied to the operation was not a well-formed component according to the name syntax of the context.</td>
</tr>
<tr>
<td>FN_E_INCOMPATIBLE_CODE_SETS</td>
<td>Code set mismatches that occur during the construction of the compound name’s string form are resolved in an implementation-dependent way. When an implementation discovers that a compound name has components with incompatible code sets, it returns this error code.</td>
</tr>
<tr>
<td>FN_E_INVALID_SYNTAX_ATTRS</td>
<td>The syntax attributes supplied are invalid or insufficient to fully specify the syntax.</td>
</tr>
<tr>
<td>FN_E_SYNTAX_NOT_SUPPORTED</td>
<td>The syntax specified is not supported.</td>
</tr>
</tbody>
</table>

### USAGE
Most applications treat names as opaque data. Hence, the majority of clients of the XFN interface will not need to parse compound names from specific naming systems. Some applications, however, such as browsers, need such capabilities. These applications would use `fn_ctx_get_syntax_attrs()` to obtain the syntax-related attributes of a context and, if the context uses the XFN standard syntax model, it would examine these attributes to determine the name syntax of the context.

### SEE ALSO
- `FN_attribute_t(3N)`, `FN_attrset_t(3N)`, `FN_compound_name_t(3N)`, `FN_identifier_t(3N)`, `FN_string_t(3N)`, `fn_ctx_get_syntax_attrs(3N)`, `xfn(3N)`

### NOTES
The implementation of XFN in this Solaris release is based on the X/Open preliminary specification. It is likely that there will be minor changes to these interfaces to reflect changes in the final version of this specification. The next minor release of Solaris will offer binary compatibility for applications developed using the current interfaces. As the interfaces evolve toward standardization, it is possible that future releases of Solaris will require minor source code changes to applications that have been developed against the preliminary specification.

---

SunOS 5.6
modified 4 Nov 1994
NAME  xfn_links – XFN links: an overview of XFN links

DESCRIPTION  An XFN link is a special form of reference that contains a composite name, the link name, and that may be bound to an atomic name in an XFN context. Because the link name is a composite name, it may span multiple namespaces.

Normal resolution of names in context operations always follows XFN links. If the first composite name component of the link name is the atomic name ",", the link name is resolved relative to the same context in which the link is bound, otherwise, the link name is resolved relative to the XFN Initial Context of the client. The link name may itself cause resolution to pass through other XFN links. This gives rise to the possibility of a cycle of links whose resolution could not terminate normally. As a simple means to avoid such non-terminating resolutions, implementations may define limits on the number of XFN links that may be resolved in any single operation invoked by the caller.

These are the interfaces:

```c
#include <xfn/xfn.h>
FN_ref_t *fn_ref_create_link( const FN_composite_name_t *link_name);
int fn_ref_is_link(const FN_ref_t *ref);
FN_composite_name_t *fn_ref_link_name( const FN_ref_t *link_ref);
FN_ref_t *fn_ctx_lookup_link(FN_ctx_t *ctx, const FN_composite_name_t *name,
                            FN_status_t *status);
unsigned int fn_status_link_code(const FN_status_t *stat);
const FN_composite_name_t *fn_status_link_remaining_name(
                            const FN_status_t *stat);
const FN_composite_name_t *fn_status_link_resolved_name( const FN_status_t *stat);
const FN_ref_t *fn_status_link_resolved_ref( const FN_status_t *stat);
int fn_status_set_link_code(FN_status_t *stat, unsigned int code);
int fn_status_set_link_remaining_name(FN_status_t *stat,
                            const FN_composite_name_t *name);
int fn_status_set_link_resolved_name(FN_status_t *stat,
                            const FN_composite_name_t *name);
int fn_status_set_link_resolved_ref(FN_status_t *stat, const FN_ref_t *ref);
```

Links are bound to names using the normal `fn_ctx_bind()` and unbound using the normal `fn_ctx_unbind()` operation. The operation `fn_ref_create_link()` is provided for constructing a link reference from a composite name. Since normal resolution always follows links, a separate operation, `fn_ctx_lookup_link()` is provided to lookup the link itself.

In the case that an error occurred while resolving an XFN link, the status object set by the operation contains additional information about that error and sets the corresponding link status fields using `fn_status_set_link_code()`, `fn_status_set_link_remaining_name()`, `fn_status_set_link_resolved_name()` and `fn_status_set_link_resolved_ref()`.
The link status fields can be retrieved using
\texttt{fn_status_link_code()}, \texttt{fn_status_link_remaining_name()},
\texttt{fn_status_link_resolved_name()} and \texttt{fn_status_link_resolved_ref()}.

**ERRORS**

The following status codes are of special relevance when performing operations involving XFN links:

\textbf{FN\_E\_LINK\_ERROR}

There was an error encountered resolving an XFN link encountered during resolution of the supplied name. Check the link part of the status object to determine cause of the link error.

\textbf{FN\_E\_LINK\_LOOP\_LIMIT}

A non-terminating loop (cycle) in the resolution can arise due to XFN links encountered during the resolution of a composite name. This code indicates either the definite detection of such a cycle, or that resolution exceeded an implementation-defined limit on the number of XFN links allowed for a single operation invoked by the caller.

\textbf{FN\_E\_MALFORMED\_LINK}

A malformed link reference was encountered. For the \texttt{fn_ctx_lookup_link()} operation, the name supplied resolved to a reference that was not a link.

**APPLICATION USAGE**

For the \texttt{fn_ctx_bind()}, \texttt{fn_ctx_unbind()}, \texttt{fn_ctx_rename()}, \texttt{fn_ctx_lookup_link()},
\texttt{fn_ctx_create_subcontext()} and \texttt{fn_ctx_destroy_subcontext()} operations, resolution of the given name continues to the target context — that named by all but the terminal atomic part of the given name; the terminal atomic name is not resolved. Consequently, for operations that involve unbinding the terminal atomic part such as \texttt{fn_ctx_unbind()}, if the terminal atomic name is bound to a link, the link is not followed and the link itself is unbound from the terminal atomic name.

Many naming systems support a native notion of link that may be used within the naming system itself. XFN does not determine whether there is any relationship between such native links and XFN links.

**SEE ALSO**

\texttt{FN\_composite\_name\_t(3N), FN\_ref\_t(3N), FN\_status\_t(3N), fn\_ctx\_bind(3N),}
\texttt{fn\_ctx\_destroy\_subcontext(3N), fn\_ctx\_lookup(3N), fn\_ctx\_lookup\_link(3N),}
\texttt{fn\_ctx\_rename(3N), fn\_ctx\_unbind(3N), xfn\_status\_codes(3N), xfn(3N)}
Network Functions

NAME  
xfn_status_codes – descriptions of XFN status codes

SYNOPSIS  
#include <xfn/xfn.h>

DESCRIPTION  
The result status of operations in the context interface and the attribute interface is encapsulated in an FN_status_t object. This object contains information about how the operation completed: whether an error occurred in performing the operation; if so, what kind of error; and information localizing where the error occurred. In the case that the error occurred while resolving an XFN link, the status object contains additional information about that error.

The context status object consists of several items of information. One of them is the primary status code, describing the disposition of the operation. In the case that an error occurred while resolving an XFN link, the primary status code has the value FN_E_LINK_ERROR, and the link status code describes the error that occurred while resolving the XFN link.

XFN Status Codes  
Both the primary status code and the link status code are values of type unsigned int that are drawn from the same set of meaningful values. XFN reserves the values 0 through 127 for standard meanings. Currently, values and interpretations for the following codes are determined by XFN.

<table>
<thead>
<tr>
<th>XFN Status Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FN_SUCCESS</td>
<td>The operation succeeded.</td>
</tr>
<tr>
<td>FN_E_ATTR_NO_PERMISSION</td>
<td>The caller did not have permission to perform the attempted attribute operation.</td>
</tr>
<tr>
<td>FN_E_ATTR_VALUE_REQUIRED</td>
<td>The operation attempted to create an attribute without a value, and the specific naming system does not allow this.</td>
</tr>
<tr>
<td>FN_E_AUTHENTICATION_FAILURE</td>
<td>The identity of the client principal could not be verified.</td>
</tr>
<tr>
<td>FN_E_COMMUNICATION_FAILURE</td>
<td>An error occurred in communicating with one of the contexts involved in the operation.</td>
</tr>
<tr>
<td>FN_E_CONFIGURATION_ERROR</td>
<td>A problem was detected that indicated an error in the installation of the XFN implementation.</td>
</tr>
<tr>
<td>FN_E_CONTINUE</td>
<td>The operation should be continued using the remaining name and the resolved reference returned in the status.</td>
</tr>
<tr>
<td>FN_E_CTX_NO_PERMISSION</td>
<td>The client did not have permission to perform the operation.</td>
</tr>
<tr>
<td>FN_E_CTX_NOT_EMPTY</td>
<td>(Applies only to fn_ctx_destroy_subcontext().) The naming system required that the context be empty before its destruction, and it was not empty.</td>
</tr>
<tr>
<td>FN_E_CTX_UNAVAILABLE</td>
<td>Service could not be obtained from one of the contexts</td>
</tr>
</tbody>
</table>

modified 7 Dec 1995

SunOS 5.6

3N-1831
<table>
<thead>
<tr>
<th><strong>Code</strong></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FN_E_ILLEGAL_NAME</td>
<td>The name supplied to the operation was not a well-formed XFN composite name, or one of the component names was not well-formed according to the syntax of the naming system(s) involved in its resolution.</td>
</tr>
<tr>
<td>FN_E_INCOMPATIBLE_CODE_SETS</td>
<td>The operation involved character strings of incompatible code sets, or the supplied code set is not supported by the implementation.</td>
</tr>
<tr>
<td>FN_E_INSUFFICIENT_RESOURCES</td>
<td>Either the client or one of the involved contexts could not obtain sufficient resources (for example, memory, file descriptors, communication ports, stable media space, and so on) to complete the operation successfully.</td>
</tr>
<tr>
<td>FN_E_INVALID_ATTR_IDENTIFIER</td>
<td>The attribute identifier was not in a format acceptable to the naming system, or its content was not valid for the format specified for the identifier.</td>
</tr>
<tr>
<td>FN_E_INVALID_ATTR_VALUE</td>
<td>One of the values supplied was not in the appropriate form for the given attribute.</td>
</tr>
<tr>
<td>FN_E_INVALID_ENUM_HANDLE</td>
<td>The enumeration handle supplied was invalid, either because it was from another enumeration, or because an update operation occurred during the enumeration, or because of some other reason.</td>
</tr>
<tr>
<td>FN_E_INVALID_SYNTAX_ATTRS</td>
<td>The syntax attributes supplied are invalid or insufficient to fully specify the syntax.</td>
</tr>
<tr>
<td>FN_E_LINK_ERROR</td>
<td>There was an error in resolving an XFN link encountered during resolution of the supplied name.</td>
</tr>
<tr>
<td>FN_E_LINK_LOOP_LIMIT</td>
<td>A non-terminating loop (cycle) in the resolution can arise due to XFN links encountered during the resolution of a composite name. This code indicates either the definite detection of such a cycle, or that resolution exceeded an implementation-defined limit on the number of XFN links allowed for a single operation invoked by the caller.</td>
</tr>
<tr>
<td>FN_E_MALFORMED_LINK</td>
<td>A malformed link reference was encountered. For \texttt{fn_ctx_lookup_link()}, the name supplied resolved to a</td>
</tr>
</tbody>
</table>
Network Functions

FN_E_MALFORMED_REFERENCE
A context object could not be constructed from the supplied reference, because the reference was not properly formed.

FN_E_NAME_IN_USE
(Only for operations that bind names.) The supplied name was already in use.

FN_E_NAME_NOT_FOUND
Resolution of the supplied composite name proceeded to a context in which the next atomic component of the name was not bound.

FN_E_NO_SUCH_ATTRIBUTE
The object did not have an attribute with the given identifier.

FN_E_NO_SUPPORTED_ADDRESS
A context object could not be constructed from a particular reference. The reference contained no address type over which the context interface was supported.

FN_E_NOT_A_CONTEXT
Either one of the intermediate atomic names did not name a context, and resolution could not proceed beyond this point, or the operation required that the caller supply the name of a context, and the name did not resolve to a reference for a context.

FN_E_OPERATION_NOT_SUPPORTED
The operation attempted is not supported.

FN_E_PARTIAL_RESULT
The operation attempted is returning a partial result.

FN_E_SYNTAX_NOT_SUPPORTED
The syntax type specified is not supported.

FN_E_TOO_MANY_ATTR_VALUES
The operation attempted to associate more values with an attribute than the naming system supported.

FN_E_UNSPECIFIED_ERROR
An error occurred that could not be classified by any of the other error codes.

#include <xfn/xfn.h> XFN status codes header file

SEE ALSO FN_status_t(3N), xfn(3N)

NOTES
The implementation of XFN in this Solaris release is based on the X/Open preliminary specification. It is likely that there will be minor changes to these interfaces to reflect changes in the final version of this specification. The next minor release of Solaris will offer binary compatibility for applications developed using the current interfaces. As the interfaces evolve toward standardization, it is possible that future releases of Solaris will require minor source code changes to applications that have been developed against the
preliminary specification.
NAME  
y0, y1, yn – Bessel functions of the second kind

SYNOPSIS  
cc [ flag . . . ] file . . . -Im [ library . . . ]
  double y0(double x);
  double y1(double x);
  double yn(int n, double x);

DESCRIPTION  
The y0(), y1() and yn() functions compute Bessel functions of x of the second kind of orders 0, 1 and n respectively. The value of x must be positive.

RETURN VALUES  
Upon successful completion, y0(), y1() and yn() will return the relevant Bessel value of x of the second kind.

If x is NaN, NaN is returned.
If the x argument to y0(), y1() or yn() is negative, -HUGE_VAL or NaN is returned, and errno may be set to EDOM.
If x is 0.0, -HUGE_VAL is returned and errno may be set to ERANGE or EDOM.
If the correct result would cause overflow, -HUGE_VAL is returned and errno may be set to ERANGE.
For exceptional cases, matherr(3M) tabulates the values to be returned as dictated by Standards other than XPG4.

ERRORS  
The y0(), y1() and yn() functions may fail if:

EDOM The value of x is negative.
ERANGE The value of x is too large in magnitude, or x is 0.0, or the correct result would cause overflow.

USAGE  
An application wishing to check for error situations should set errno to 0 before calling y0(), y1() or yn(). If errno is non-zero on return, or the return value is NaN, an error has occurred.

ATTRIBUTES  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO  
isnan(3M), j0(3M), matherr(3M), attributes(5), standards(5)
NAME
ypclnt, yp_get_default_domain, yp_bind, yp_unbind, yp_match, yp_first, yp_next,
yp_all, yp_order, yp_master, yperr_string, ypprot_err - NIS Version 2 client interface

SYNOPSIS
cc [flag ...] file ... -lnsl [ library ... ]
#include <rpcsvc/ypclnt.h>
#include <rpcsvc/yp_prot.h>

DESCRIPTION
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 nis+(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.

All input parameter names begin with in. Output parameters begin with out. 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 outkey and outval, two extra bytes of memory are allocated at the end that contain NEWLINE and null, respectively, but these two bytes are not reflected in outkeylen or outvallen. indomain and inmap 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.

All functions in this package of type int return 0 if they succeed, and a failure code (YPERR_xxxx) otherwise. Failure codes are described under ERRORS below.

Routines

yp_bind (char *indomain);

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.
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 mul-
tiple 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 run-
ning, or if a bound ypser 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.bynam 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 nth call to yp_next() (to get the nth + 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.

<table>
<thead>
<tr>
<th>Code</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>YPERR_ACCESS 15</td>
<td>access violation</td>
</tr>
<tr>
<td>YPERR_BADARGS 1</td>
<td>args to function are bad</td>
</tr>
</tbody>
</table>

modified 30 Dec 1996
YPERR_BADDB 13 /* yp database is bad */
YPERR_BUSY 16 /* database busy */
YPERR.DOMAIN 3 /* can’t bind to server on this domain */
YPERR_KEY 5 /* no such key in map */
YPERR_MAP 4 /* no such map in server’s domain */
YPERR_NODOM 12 /* local domain name not set */
YPERR_NOMORE 8 /* no more records in map database */
YPERR_PMAP 9 /* can’t communicate with rpcbinder */
YPERR_RESRC 7 /* resource allocation failure */
YPERR_RPC 2 /* RPC failure – domain has been unbound */
YPERR_YPBIND 10 /* can’t communicate with ypbind */
YPERR_YPERR 6 /* internal yp server or client error */
YPERR_YPSERV 11 /* can’t communicate with ypserv */
YPERR_VERS 14 /* yp version mismatch */

FILES /usr/lib/libnsl.so.1

ATTRIBUTES See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO nis+(1), ypcat(1), ypmatch(1), ypwhich(1), rpc.nisd(1M), rpcbind(1M), ypbind(1M), sysinfo(2), malloc(3C), ypfiles(4), attributes(5)

NOTES This interface is unsafe in multithreaded applications. Unsafe interfaces should be called only from the main thread.
NAME

yp_update – change NIS information

SYNOPSIS

#include <rpcsvc/ypclnt.h>

int yp_update(char *domain, char *map, unsigned ypop, char *key, int keylen,
              char *data, int datalen);

DESCRIPTION

yp_update() is used to make changes to the NIS database. The syntax is the same as that
of yp_match() except for the extra parameter ypop 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 ypop has the value YPOP_INSERT then the key-value pair will be inserted into the data-
base. 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 ypop
as YPOP_STORE and no error will be returned if the key already or does not exist. To
delete an entry, the value of ypop should be YPOP_DELETE.

This routine depends upon secure RPC, and will not work unless the network is running
secure RPC.

RETURN VALUES

If the value of ypop is POP_CHANGE, yp_update() returns the error YPERR_KEY if the key
is not found in the database.

If the value of ypop is POP_INSERT, yp_update() returns the error YPERR_KEY if the key
already exists in the database.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO

secure_rpc(3N), ypclnt(3N), attributes(5)

NOTES

This interface is unsafe in multithreaded applications. Unsafe interfaces should be called
only from the main thread.

modified 30 Dec 1996
Index

Special Characters

,longjmp — non-local goto, 3B-1411, 3C-937
,NOTE — annotate source code with info for tools, 3X-1101
,setjmp — non-local goto, 3B-1411, 3C-937
,tolower — transliterate upper-case characters to lower-case, 3C-1675
,toupper — transliterate lower-case characters to upper-case, 3C-1688

A

abort — terminate the process abnormally, 3C-103
abs — return absolute value of integer, 3C-104
absolute value function — fabs, 3M-507
accept — accept a connection on a socket, 3N-105, 3XN-107
accept a new connection on a socket — accept, 3XN-107
access utmpx file entry  
  — endutxent, 3C-793
  — getutmp, 3C-793
  — getutmpx, 3C-793
  — getutxent, 3C-793
  — getutxid, 3C-793
  — getutxline, 3C-793
  — pututxline, 3C-793
  — setutxent, 3C-793
access utmpx file entry, continued  
  — updwtmp, 3C-793
  — updwtmpx, 3C-793
  — utmpxname, 3C-793
accounting  
  time accounting for current process — times, 3B-1628
acos — arc cosine function, 3M-116
acosh — inverse hyperbolic functions, 3M-117
acquire and release stream lock — flockfile, 3S-528
  funlockfile, 3S-528
activate audio-visual alarm  
  — beep, 3XC-166
  — flash, 3XC-166
add a character (with rendition) to a window  
  — addch, 3XC-118
  — mvaddch, 3XC-118
  — mvwaddch, 3XC-118
  — waddch, 3XC-118
add a complex character (with rendition) to a window  
  — add_wch, 3XC-128
  — mvadd_wch, 3XC-128
  — mvwadd_wch, 3XC-128
  — wadd_wch, 3XC-128
add a complex character and refresh window  
  — echo_wchar, 3XC-440

Index-1
add a complex character and refresh window, continued
   — wecho_wchar, 3XC-440
add a multi-byte character string (without rendition) to a window
   — addnstr, 3XC-121
   — addstr, 3XC-121
   — mvaddnstr, 3XC-121
   — mvaddstr, 3XC-121
   — mvwaddnstr, 3XC-121
   — mvwaddstr, 3XC-121
   — waddnstr, 3XC-121
   — waddstr, 3XC-121
add a single-byte border to a window
   — border, 3XC-175
   — box, 3XC-175
   — wborder, 3XC-175
add a single-byte character and refresh window
   — echochar, 3XC-439
   — wechochar, 3XC-439
add a string of wchar_t characters to a curses window and advance cursor — curs_addwstr, 3X-277
   addnwstr, 3X-277
   addwstr, 3X-277
   mvaddnwstr, 3X-277
   mvaddwstr, 3X-277
   mvwaddnwstr, 3X-277
   mvwaddwstr, 3X-277
   waddnwstr, 3X-277
   waddwstr, 3X-277
add wchar_t character (with attributes) to a curses window and advance cursor — curs_addwch, 3X-272
   addwch, 3X-272
   echowchar, 3X-272
   mvaddwch, 3X-272
   mvwaddwch, 3X-272
   waddwch, 3X-272
   wechowchar, 3X-272
add a wide-character string to a window
   — addnstr, 3XC-123
   — addstr, 3XC-123
   — mvaddnstr, 3XC-123
add a wide-character string to a window, continued
   — mvaddwstr, 3XC-123
   — mvwaddnstr, 3XC-123
   — mvwaddwstr, 3XC-123
   — waddnstr, 3XC-123
   — waddwstr, 3XC-123
add character and refresh window
   — pecho_wchar, 3XC-1160
   — pechochar, 3XC-1160
add string of wchar_t characters (and attributes) to a curses window — curs_addwchstr, 3X-275
   addwchnstr, 3X-275
   addwchstr, 3X-275
   mvaddwchstr, 3X-275
   mvwaddwchstr, 3X-275
   waddwchstr, 3X-275
add wchar_t character (with rendition) to a window, 3XC-128
add_wch — copy a string of complex characters (with renditions) to a window, 3XC-130
add_wchstr — copy a string of complex characters (with renditions) to a window, 3XC-130
addch — add a character (with rendition) to a window, 3XC-118
addchstr — copy a character string (with renditions) to a window, 3XC-119
addchstr — copy a character string (with renditions) to a window, 3XC-119
additional severities
define — addsev, 3C-125
addnstr — add a multi-byte character string (without rendition) to a window, 3XC-121
addnwstr — add a string of wchar_t characters to a curses window and advance cursor, 3X-277, 3XC-123
address in an XFN reference
   — fn_ref_addr_assign, 3N-607
   — fn_ref_addr_copy, 3N-607
   — fn_ref_addr_create, 3N-607
   — fn_ref_addr_data, 3N-607

Index—2
address in an XFN reference, continued
— fn_ref_addr_description, 3N-607
— fn_ref_addr_destroy, 3N-607
— fn_ref_addr_length, 3N-607
— FN_ref_addr_t, 3N-607
— fn_ref_addr_type, 3N-607

address of symbol
get address in shared object — dlsym, 3X-395
addsev — define additional severities, 3C-125
addseverity — build a list of severity levels for
an application for use with fmtmsg, 3C-126
addstr — add a multi-byte character string
(without rendition) to a window, 3XC-121
addwch — add a wchar_t character (with attributes)
to a curses window and advance cursor, 3X-272
addwchnstr — add string of wchar_t characters
(and attributes) to a curses window, 3X-275
addwchstr — add string of wchar_t characters
(and attributes) to a curses window, 3X-275
addwstr — add a string of wchar_t characters to a
curses window and advance cursor, 3X-277,
3XC-123
adjcurspos — moving the cursor by character,
3X-278
advance — regular expression compile and match
routines, 3G-1290
aio_cancel — cancel asynchronous I/O request,
3R-133
aio_error — retrieve error status of asynchronous
I/O operation, 3R-143
aio_fsync — asynchronous file synchronization,
3R-135
aio_read — asynchronous read and write opera-
tions, 3R-140
aio_return — retrieve return status of asynchro-
 nous I/O operation, 3R-143
aio_suspend — wait for asynchronous I/O
request, 3R-146
aio_write — asynchronous read and write opera-
tions, 3R-140
aiocancel — cancel an asynchronous operation,
3-132
aioread — initiate asynchronous read, 3-137
aioread64 — initiate asynchronous read, 3-137
aiowait — wait for completion of asynchronous
I/O operation, 3-148
aioread — initiate asynchronous write, 3-137
aioread64 — initiate asynchronous write, 3-137
alarm
schedule signal after interval in microseconds
— ualarm, 3C-1742
ALE curses library, See curses library
allocate and deallocate process handles for
libthread_db
— td_tad_delete, 3T-1575
— td_tad_get_ph, 3T-1575
— td_tad_new, 3T-1575
allocate or deallocate a buffer for trace data
— tnfctl_buffer_alloc, 3X-1637
— tnfctl_buffer_dealloc, 3X-1637
alphasort — scan a directory, 3B-1359
annotate source code with info for tools
— _NOTE, 3X-1101
— NOTE, 3X-1101
applications
build a list of severity levels for use with
fmtmsg — addseverity, 3C-126
display a message on stderr or system console
— fmtmsg, 3C-533
get entries from symbol table — nlist,
3B-1092
apply padding information and output string
— putp, 3XC-1244
— tputs, 3XC-1244
arc — graphics interface, 3-1166
arc cosine function — acos, 3M-116
arc sine function — asin, 3M-149
arc tangent function — atan2, 3M-151, 3M-152
arithmetic
compute the quotient and remainder — div,
3C-377
arithmetic, 48-bit integer
generate uniformly distributed pseudo-random
numbers — drand48, 3C-434
ascftime — convert date and time to string,
asin — arc sine function, 3M-149
asinh — inverse hyperbolic functions, 3M-117
assert — verify program assertion, 3C-150
associate a stream with a file descriptor — fdopen, 3S-514
asynchronous file synchronization
— aio_sync, 3R-135
asynchronous I/O
— aio_cancel, 3R-133
— aiocancel, 3-132
— aioread, 3-137
— aioread64, 3-137
— aiowait, 3-148
— aiowrite, 3-137
— aiowrite64, 3-137
retrieve error status — aio_error, 3R-143
retrieve return status — aio_return, 3R-143
wait for request — aio_suspend, 3R-146
asynchronous read and write operations
— aio_read, aio_write, 3R-140
asysmem — return physical memory information, 3-1529
atan — arc tangent function, 3M-152
atan2 — arc tangent function, 3M-151
atanh — inverse hyperbolic functions, 3M-117
atexit — add program termination routine, 3C-153
atof — convert string to double-precision number, 3C-1506
atoi — string conversion routines, 3C-1508
atol — string conversion routines, 3C-1508
atoll — string conversion routines, 3C-1508
attr_get — control window attributes, 3XC-154
attr_off — control window attributes, 3XC-154
attr_on — control window attributes, 3XC-154
attr_set — control window attributes, 3XC-154
attribute modifications, list of, continued
— fn_attrmodlist_destroy, 3N-555
— fn_attrmodlist_first, 3N-555
— fn_attrmodlist_next, 3N-555
— FN_attrmodlist_t, 3N-555
attribute search options
— fn_search_control_assign, 3N-612
— fn_search_control_copy, 3N-612
— fn_search_control_create, 3N-612
— fn_search_control_destroy, 3N-612
— fn_search_control_follow_links, 3N-612
— fn_search_control_max_names, 3N-612
— fn_search_control_return_attr_ids, 3N-612
— fn_search_control_return_ref, 3N-612
— fn_search_control_scope, 3N-612
— FN_search_control_t, 3N-612
attroff — change foreground window attributes, 3XC-156, 3X-280
attron — change foreground window attributes, 3XC-156, 3X-280
attrset — change foreground window attributes, 3XC-156, 3X-280
au_close — construct audit records, 3-157
au_open — construct audit records, 3-157
au_preselect — preselect an audit record, 3-158
au_to_arg — creating audit record tokens, 3-160
au_to_attr — creating audit record tokens, 3-160
au_to_data — creating audit record tokens, 3-160
au_to_groups — creating audit record tokens, 3-160
au_to_in_addr — creating audit record tokens, 3-160
au_to_ipc — creating audit record tokens, 3-160
au_to_ipc_perm — creating audit record tokens, 3-160
au_to_iport — creating audit record tokens, 3-160
au_to_me — creating audit record tokens, 3-160
au_to_opaque — creating audit record tokens,
au_to_path — creating audit record tokens, 3-160
au_to_process — creating audit record tokens, 3-160
au_to_return — creating audit record tokens, 3-160
au_to_text — creating audit record tokens, 3-160
au_user_mask — get user’s binary preselection mask, 3-163
au_write — write audit records, 3-157

audit control file
— endac, 3-684
— getacdir, 3-684
— getacfg, 3-684
— getacinfo, 3-684
— getacm, 3-684
— getacna, 3-684
— setac, 3-684

audit record tokens, creating
— au_to_attr, 3-160
— au_to_data, 3-160
— au_to_groups, 3-160
— au_to_in_addr, 3-160
— au_to_in_ipc, 3-160
— au_to_in_ipc_perm, 3-160
— au_to_iport, 3-160
— au_to_me, 3-160
— au_to_opaque, 3-160
— au_to_path, 3-160
— au_to_process, 3-160
— au_to_return, 3-160
— au_to_socket, 3-160
— au_to_subject, 3-160
— au_to_text, 3-160

audit record tokens, manipulating
— au_close, 3-157
— au_open, 3-157
— au_preselect, 3-158
— au_to_attr, 3-160
— au_to_data, 3-160
— au_to_groups, 3-160
— au_to_in_addr, 3-160
— au_to_in_ipc, 3-160
— au_to_in_ipc_perm, 3-160
— au_to_iport, 3-160
— au_to_me, 3-160
— au_to_opaque, 3-160
— au_to_path, 3-160
— au_to_process, 3-160
— au_to_return, 3-160
— au_to_socket, 3-160
— au_to_subject, 3-160
— au_to_text, 3-160
— au_user_mask, 3-163
— au_write, 3-157

auth_destroy — library routines for client side remote procedure call authentication, 3N-1313
authnone_create — library routines for client side remote procedure call authentication, 3N-1313
authsys_create — library routines for client side remote procedure call authentication, 3N-1313
authsys_create_default — library routines for client side remote procedure call authentication, 3N-1313

B
base 10 logarithm function — log10, 3M-933
base-64 ASCII characters
convert from long integer — l64a, 3C-102
convert to long integer — a64l, 3C-102
basename — return the last element of path name, 3C-164

Basic Security Module functions
— au_close, 3-157
— au_open, 3-157
— au_preselect, 3-158
— au_to_attr, 3-160
— au_to_data, 3-160
— au_to_groups, 3-160
— au_to_in_addr, 3-160
— au_to_in_ipc, 3-160
— au_to_in_ipc_perm, 3-160
— au_to_iport, 3-160
— au_to_me, 3-160
— au_to_opaque, 3-160
— au_to_path, 3-160
— au_to_process, 3-160
— au_to_return, 3-160
— au_to_socket, 3-160
— au_to_subject, 3-160
— au_to_text, 3-160
— au_user_mask, 3-163
— au_write, 3-157

baudrate — return terminal baud rate, 3XC-165
bcmp — operates on variable length strings of bytes, 3C-184
bcopy — operates on variable length strings of
beps — activate audio-visual alarm, 3XC-166
Bessel functions of the first kind
— j0, 3M-867
— j1, 3M-867
— jn, 3M-867
Bessel functions of the second kind
— y0, 3M-1835
— y1, 3M-1835
— yn, 3M-1835
bgets — read stream up to next delimiter, 3G-167
binary search of sorted table
— bsearch, 3C-182
binary search trees, manage
— tdelete, 3C-1717
— tfind, 3C-1717
— tsearch, 3C-1717
— twalk, 3C-1717
bind — bind a name to a socket, 3N-168, 3XN-170
bind a name to a socket — bind, 3XN-170
bind a reference to a name — fn_ctx_bind, 3N-579
bind a reference to a name and associate attributes
with named object — fn_attr_bind, 3N-538
bind an address to a transport endpoint —
t_bind, 3N-1540
bind or unbind the current thread with the door
server pool
— door_bind, 3X-419
— door_unbind, 3X-419
bindtextdomain — select location of domain, 3C-783
bit and byte operations
find first set bit — ffs, 3C-519
bkgd — set the background character (and rendition) of window, 3XC-172
bkgdset — set the background character (and rendition) of window, 3XC-172
bkgrnd — set or get the background character (and rendition) of window using a complex character, 3XC-173
border — add a single-byte border to a window, 3XC-175
border_set — use complex characters (and renditions) to draw borders, 3XC-177
box — add a single-byte border to a window, 3XC-175, 3-1166
box_set — use complex characters (and renditions) to draw borders, 3XC-173
bsd_signal — simplified signal facilities, 3C-181
bsdmalloc — memory allocator, 3X-179
bsearch — binary search a sorted table, 3C-182
BSM, See Basic Security Module
bstring — bit and byte string operations, 3C-184
buffer
split into fields — bsplit, 3G-185
buffering, assign to stream
— setbuffer, 3C-1408
— setlinebuf, 3C-1408
byte order, convert values between host and network
— byteorder, 3N-186
— htonl, 3N-186
— htons, 3N-186
— ntohl, 3N-186
— ntohs, 3N-186
byte swap — swab, 3C-1516
bzero — operates on variable length strings of bytes, 3C-184

C
C Compilation
close a shared object — dlclose, 3X-380
create new file from dynamic object component
— dldump, 3X-381
get address of symbol in shared object —
dlsym, 3X-395
get diagnostic information — dlerror, 3X-388
open a shared object — dlopen, 3X-391
translate address to symbolic information —
dladdr, 3X-378
call refresh on changes to window — immedok, 3XC-828
can_change_color — manipulate color information, 3XC-193
cancel execution of a thread — pthread_cancel, 3T-1198
canceling execution of a thread —
pthread_cancel, 3T-1198
cancellation — overview of concepts related to
   POSIX thread cancellation, 3T-187
   Cancel-Safe, 3T-190
   Cancellation, 3T-187
   Cancellation Points, 3T-188
   Cancellation State, 3T-189
   Cancellation Type, 3T-189
   Cleanup Handlers, 3T-189
   Planning Steps, 3T-187
   POSIX Threads Only, 3T-190
catclose — close a message catalog, 3C-197
catgets — read a program message, 3C-196
catopen — open a message catalog, 3C-197
cbreak — set input mode controls, 3XC-200
cbrt — cube root function, 3M-201
ceil — ceiling value function, 3M-202
ceiling value function — ceil, 3M-202
cfgetispeed — get input baud rate, 3-203
cfgetospeed — get output baud rate, 3-203
cfsetispeed — set input baud rate, 3-204
cfsetospeed — set output baud rate, 3-204
cftime — convert date and time to string, 3C-1493
change foreground window attributes
   — attroff, 3XC-156
   — attron, 3XC-156
   — attrset, 3XC-156
   — wattroff, 3XC-156
   — wattroff, 3XC-156
   — wattstat, 3XC-156
change or add a value to the PAM environment —
pam_putenv, 3-1125
change the rendition of characters in a window
   — chgat, 3XC-205
   — mvchgt, 3XC-205
   — mvwchgt, 3XC-205
   — wcwchgt, 3XC-205
character based forms package, continued
   — forms, 3X-634
character based menus package
   — menus, 3X-969
character based panels package
   — panels, 3X-1148
character handling
   — ctype, 3C-243
   — isalnum, 3C-243
   — isalpha, 3C-243
   — isascii, 3C-243
   — iscntrl, 3C-243
   — isdigit, 3C-243
   — isgraph, 3C-243
   — islower, 3C-243
   — isprint, 3C-243
   — ispunct, 3C-243
   — isspace, 3C-243
   — isupper, 3C-243
   — isxdigit, 3C-243
character string
   — fn_string_assign, 3N-626
   — fn_string_bytecount, 3N-626
   — fn_string_charcount, 3N-626
   — fn_string_code_set, 3N-626
   — fn_string_compare, 3N-626
   — fn_string_compare_substring, 3N-626
   — fn_string_contents, 3N-626
   — fn_string_copy, 3N-626
   — fn_string_create, 3N-626
   — fn_string_destroy, 3N-626
   — fn_string_from_contents, 3N-626
   — fn_string_from_str, 3N-626
   — fn_string_from_str_n, 3N-626
   — fn_string_from_strings, 3N-626
   — fn_string_from_substring, 3N-626
   — fn_string_is_empty, 3N-626
   — fn_string_next_substring, 3N-626
   — fn_string_prev_substring, 3N-626
   — fn_string_str, 3N-626
   — FN_string_t, 3N-626
check for type-ahead characters — typeahead, 3XC-1741
check whether or not Volume Management is
   managing a pathname — volmgt_inuse,
check whether specific Volume Management features are enabled — volmgt_feature_enabled, 3X-1757
cchgat — change the rendition of characters in a window, 3XC-205
circle — graphics interface, 3-1166
clear — clear a window, 3XC-206
clear a window
— clear, 3XC-206
— erase, 3XC-206
— wc1ear, 3XC-206
— werae, 3XC-206
clear to the end of a line
— clrtoeol, 3XC-214
— wc1rtoeol, 3XC-214
clear to the end of a window
— clrtobot, 3XC-213
— wclrtobot, 3XC-213
clearok — set terminal output controls, 3XC-207
client side remote procedure call authentication, library routines for
— auth_destroy, 3N-1313
— authnone_create, 3N-1313
— authsys_create, 3N-1313
— authsys_create_default, 3N-1313
— rpc_clnt_auth, 3N-1313
c1nt_call — library routines for client side calls, 3N-1315
c1nt_control — library routines for dealing with creation and manipulation of CLIENT handles, 3N-1318
c1nt_create — library routines for dealing with creation and manipulation of CLIENT handles, 3N-1318
c1nt_create timed — library routines for dealing with creation and manipulation of CLIENT handles, 3N-1318
c1nt_create_vers — library routines for dealing with creation and manipulation of CLIENT handles, 3N-1318
c1nt_create_vers timed — library routines for dealing with creation and manipulation of
CLIENT handles, 3N-1318
c1nt_destroy — library routines for dealing with creation and manipulation of CLIENT handles, 3N-1318
c1nt_dg create — library routines for dealing with creation and manipulation of CLIENT handles, 3N-1318
c1nt_freeres — library routines for client side calls, 3N-1315
c1nt_geterr — library routines for client side calls, 3N-1315
c1nt_pcreateerror — library routines for dealing with creation and manipulation of CLIENT handles, 3N-1318
c1nt_perror — library routines for client side calls, 3N-1315
c1nt_spcreateerror — library routines for dealing with creation and manipulation of CLIENT handles, 3N-1318
c1nt_sperrno — library routines for client side calls, 3N-1315
c1nt_sperror — library routines for client side calls, 3N-1315
c1nt_sperror — library routines for client side calls, 3N-1315
c1nt_tli_create — library routines for dealing with creation and manipulation of CLIENT handles, 3N-1318
c1nt_tp create — library routines for dealing with creation and manipulation of CLIENT handles, 3N-1318
c1nt_tp_create timed — library routines for dealing with creation and manipulation of CLIENT handles, 3N-1318
c1nt_vc_create — library routines for dealing with creation and manipulation of CLIENT handles, 3N-1318
clock — report CPU time used, 3C-209
clock_getres — high-resolution clock operations,
clock_gettime — high-resolution clock operations, 3R-210

clock_settime — high-resolution clock operations, 3R-210

close a directory stream — closedir, 3C-212
close a shared object — dlclose, 3X-380
close a stream — fclose, 3S-510
close a tfnctl handle — tfnctl_close, 3X-1639
closedir — close a directory stream, 3C-212
closelog — control system log, 3-1525
closepl — graphics interface, 3-1166
closevt — graphics interface, 3-1166
clrtobot — clear to the end of a window, 3XC-213
clrtoeol — clear to the end of a line, 3XC-214
code conversion allocation function —
iconv_open, 3-825
code conversion deallocation function —
iconv_close, 3-824
code conversion for Process Code and File Code
— strtows, 3C-1513
— wstostr, 3C-1513
code conversion function — iconv, 3-822
collect target process statistics for libthread_db
— td_ta_enable_stats, 3T-1566
— td_ta_get_stats, 3T-1566
— td_ta_reset_stats, 3T-1566
color_content — manipulate color information, 3XC-193
COLOR_PAIR — manipulate color information, 3XC-193
color_set — control window attributes, 3XC-154
column positions of a wide-character code —
wcwidth, 3C-1800
column positions of a wide-character string —
wcswidth, 3C-1794
command options
get option letter from argument vector —
getopt, 3C-740
command suboptions
parse suboptions from a string —
getsubopt, 3C-780
command suboptions, continued
commands
open, close to and from a command —
p2open, p2close, 3G-1111
return stream to remote — rcmd, 3N-1259
communications
accept a connection on a socket — accept,
3N-105
accept a connection request — t_accept,
3N-1531
acknowledge receipt of an orderly release indication —
t_rcvrel, 3N-1707
allocate memory for argument structures —
t_alloc, 3N-1535
bind a name to a socket — bind, 3N-168
bind an address to a transport endpoint —
t_bind, 3N-1540
close a transport endpoint — t_close,
3N-1550
create a pair of connected sockets — socket-pair,
3N-1465
create an endpoint for communication —
socket, 3N-1460
disable a transport endpoint — t_unbind,
3N-1739
establish a connection with another transport user — t_connect, 3N-1552
establish a transport endpoint — t_open,
3N-1677
free allocated memory — t_free, 3N-1599
get name of peer connected to socket —
getpeername, 3N-745
get protocol-specific service information —
t_getinfo, 3N-1602
get socket name — getsockname, 3N-769
get the current state — t_getstate, 3N-1608
initiate a connection on a socket — connect,
3N-224
initiate an orderly release of connection —
t_sndrel, 3N-1727
listen for a connect indication — t_listen,
3N-1629
listen for connections on a socket — listen,
3N-924

Index—9
communications, continued

look at the current event on a transport endpoint — t_look, 3N-1632
manage options for a transport endpoint —
t_optmgmt, 3N-1681
produce error message — t_error, 3N-1597
receive a data unit — t_rcvudata, 3N-1709
receive a unit data error indication —
t_rcvuderr, 3N-1712
receive data or expedited data sent over a connection — t_rcv, 3N-1698
receive the confirmation from a connection request — t_rcvconnect, 3N-1701
retrieve information from disconnect —
t_rcvdis, 3N-1704
scatter data in order to test the network —
spray, 3N-1468
send a data unit — t_sndudata, 3N-1729
send a message from a socket — send,
sendto, sendmsg, 3N-1396
send data or expedited data over a connection — t_snd, 3N-1720
send user-initiated disconnect request —
t_snddis, 3N-1724
shut down part of a full-duplex connection —
shutdown, 3N-1429
synchronize transport library — t_sync, 3N-1734

compile — regular expression compile and match routines, 3G-1290
compile and execute regular expressions
— re_comp, 3C-1270
— re_exec, 3C-1270

component names spanning multiple naming systems
— fn_composite_name_append_comp, 3N-571
— fn_composite_name_append_name, 3N-571
— fn_composite_name_assign, 3N-571
— fn_composite_name_copy, 3N-571
— fn_composite_name_create, 3N-571
— fn_composite_name_delete_comp, 3N-571
— fn_composite_name_delete_name, 3N-571
— fn_composite_name_first, 3N-571
— fn_composite_name_from_comp, 3N-571
— fn_composite_name_from_name, 3N-571
— fn_composite_name_from_str, 3N-571
— fn_composite_name_from_string, 3N-571
— fn_composite_name_get_comp, 3N-571
— fn_composite_name_get_name, 3N-571
— fn_composite_name_is_empty, 3N-571
— fn_composite_name_is_equal, 3N-571
— fn_composite_name_is_prefix, 3N-571
— fn_composite_name_is_suffix, 3N-571
— fn_composite_name_last, 3N-571
— fn_composite_name_next, 3N-571
— fn_composite_name_prev, 3N-571
— FN_composite_name_t, 3N-571
— fn_composite_name_prepend_comp, 3N-571
— fn_composite_name_prepend_name, 3N-571
— fn_composite_name_prefix, 3N-571
— fn_composite_name_prepend, 3N-571
— fn_composite_name_suffix, 3N-571
— fn_composite_name_remove_comp, 3N-571
— fn_composite_name_remove_name, 3N-571
— fn_composite_name_reverse, 3N-571
— fn_composite_name_search_comp, 3N-571
— fn_composite_name_search_name, 3N-571
— fn_composite_name_str, 3N-571
— fn_composite_name_string, 3N-571
— fn_composite_name_sub, 3N-571
— fn_composite_name_substring, 3N-571
— fn_composite_name_to_comp, 3N-571
— fn_composite_name_to_name, 3N-571
— fn_composite_name_to_str, 3N-571
— fn_composite_name_to_string, 3N-571
— fn_composite_name_value, 3N-571
compute natural logarithm — log1p, 3M-934
compiles exponential functions — expm1, 3M-506
cond_attr — condition variable initialization attributes, 3T-1201
cond_broadcast() — signal a condition variable, 3T-215
cond_destroy() — destroy a condition variable, 3T-215
cond_init() — initialize a condition variable, 3T-215
cond_signal() — signal a condition variable, 3T-215
cond_wait() — wait for a condition variable, 3T-215
cond_wait() — wait for a condition variable, 3T-215
condition variable initialization attributes —
cond_attr, 3T-1201
configuration script
   execute — doconfig, 3N-417
confstr — get configurable variables, 3C-223
connect — connect a socket, 3XN-227, 3N-224
connect a socket — connect, 3XN-227
construct a handle to a context object using the
given reference —
   fn_ctx_handle_from_ref, 3N-590
construct equivalent name in same context —
   fn_ctx_equivalent_name, 3N-583
cont — graphics interface, 3-1166
control flush of input and output on interrupt
   — noqiflush, 3XC-1100
   — qiflush, 3XC-1100
control kernel tracing and process filtering
   — tnfctl_filter_list_add, 3X-1663
   — tnfctl_filter_list_delete, 3X-1663
   — tnfctl_filter_list_get, 3X-1663
   — tnfctl_filter_state_set, 3X-1663
   — tnfctl_trace_state_set, 3X-1663
control probes of another process where caller pro-
vides /proc functionality
   — tnfctl_check_libs, 3X-1641
   — tnfctl_indirect_open, 3X-1641
control system log
   — closelog, 3-1525
   — openlog, 3-1525
   — setlogmask, 3-1525
   — syslog, 3-1525
control window attributes
   — attr_get, 3XC-154
   — attr_off, 3XC-154
   — attr_on, 3XC-154
   — attr_set, 3XC-154
   — color_set, 3XC-154
   — wattr_get, 3XC-154
   — wattr_off, 3XC-154
   — wattr_on, 3XC-154
   — wattr_set, 3XC-154
   — wcolor_set, 3XC-154
control window refresh
   — is_linetouched, 3XC-858
   — is_wintouched, 3XC-858
control window refresh, continued
   — touchline, 3XC-858
   — touchwin, 3XC-858
   — untouchwin, 3XC-858
   — wtouchln, 3XC-858
convert a character string to a wide-character string
   — mbstowcs, 3C-960
convert a character to a wide-character code —
   mbtowc, 3C-961
convert a supplied name into an absolute pathname
   that can be used to access removable media —
   media_findname, 3X-964
convert a thread id or thread address to a thread
   handle
   — td_ta_map_addr2thr, 3T-1574
   — td_ta_map_id2thr, 3T-1574
convert a wide character to printable form —
   wunctrl, 3XC-1807
convert a wide-character code to a character —
   wctomb, 3C-1797
convert a wide-character string to a character string
   — wcstombs, 3C-1787
convert between Volume Management symbolic
   names, and the devices that correspond to them
   — volmgt_symdev, 3X-1763
   — volmgt_symname, 3X-1763
convert character to printable form — unctrl, 3XC-1743
convert date and time to string — strftime,
   3C-1493
   — asctime, 3C-1493
   — cftime, 3C-1493
convert date and time to wide character string —
   wcsftime, 3C-1782
convert floating-point number to string
   — ecvt, 3C-443
   — fcvt, 3C-443
   — gcvt, 3C-443
convert formatted input — scanf, 3S-1360
   — fscanf, 3S-1360
   — sscanf, 3S-1360
convert monetary value to string — strfmon,
   3C-1489
Index—11
convert numbers to strings
   — ecvt, 3-441
   — fconvert, 3-441
   — fcvt, 3-441
   — fprintf, 3B-1171
   — gconvert, 3-441
   — gcvt, 3-441
   — printf, 3B-1171
   — qconvert, 3-441
   — qfconvert, 3-441
   — qgconvert, 3-441
   — qseconvert, 3-441
   — qfconvert, 3-441
   — qgconvert, 3-441
   — qseconvert, 3-441
   — fprintf, 3B-1171
   — gconvert, 3-441
   — gcvt, 3-441
   — printf, 3B-1171
   — qconvert, 3-441
   — qfconvert, 3-441
   — qgconvert, 3-441
   — qseconvert, 3-441
   — qfconvert, 3-441
   — qgconvert, 3-441
   — qseconvert, 3-441
convert string to double-precision number
   — atof, 3C-1506
   — strtod, 3C-1506
convert string to unsigned long — strtoul, 3C-1511
convert to wchar_t strings
   — wprintf, 3C-1804
convert values between host and network byte order
   — htonl, 3XN-820
   — htons, 3XN-820
   — ntohl, 3XN-820
   — ntohs, 3XN-820
convert wide character string to double-precision number — wcstod, 3C-1783
   — watof, 3C-1783
   — wstod, 3C-1783
convert wide character string to unsigned long — wcstoul, 3C-1788
copy a character string (with renditions) to a window, continued
   — mvaddchnstr, 3XC-119
   — mvaddchstr, 3XC-119
   — waddchnstr, 3XC-119
   — waddchstr, 3XC-119
copy a string of complex characters (with renditions) to a window
   — add_wchstr, 3XC-130
   — add_wchstr, 3XC-130
   — mvadd_wchstr, 3XC-130
   — mvadd_wchstr, 3XC-130
   — mvadd_wchstr, 3XC-130
   — wadd_wchstr, 3XC-130
   — wadd_wchstr, 3XC-130
copysign — return magnitude of first argument and sign of second argument, 3M-231
copywin — overlay or overwrite any portion of window, 3XC-232
cos — cosine function, 3M-234
cosh — hyperbolic cosine function, 3M-235
cosine function — cos, 3M-234
cplus_demangle — decode a C++ encoded symbol name, 3-363
CPU time
   — report for calling process — clock, 3C-209
CPU-use
   — prepare execution profile — monitor, 3C-1007
create a door descriptor — door_create, 3X-425
create a new window or subwindow
   — derwin, 3XC-364
   — newwin, 3XC-364
   — subwin, 3XC-364
create a pair of connected sockets — socketpair, 3XN-1466
create a temporary file — tmpfile, 3S-1634
create an endpoint for communication — socket, 3XN-1463
create cancellation point in the calling thread. — pthread_testcancel, 3T-1239
create handle for internal process probe control — tnfctl_internal_open, 3X-1644
create handle for kernel probe control —
  tnfctl_kernel_open, 3X-1646
create new file from dynamic object component —
  dldump, 3X-381
create or refresh a pad or subpad —
  newpad, 3XC-1049
  pnoutrefresh, 3XC-1049
  refresh, 3XC-1049
  subpad, 3XC-1049
create subcontext and associate attributes —
  fn_attr_create_subcontext, 3N-539
CRT handling and optimization package —
  curses, 3X-246
crypt — string encoding function, 3C-236
cset — get information on EUC codesets, 3C-237
csetcol — get information on EUC codesets,
  3C-237
csetlen — get information on EUC codesets,
  3C-237
csetno — get information on EUC codesets,
  3C-237
cetermin — generate path name for controlling termi-
  nal, 3S-238
cetermin_r — generate path name for controlling
  terminal, 3S-238
ctype — character handling, 3C-243
cube root function — cb rodents, 3M-201
current location of a named directory stream —
  telldir, 3C-1593
current working directory
get path name — getcwd, 3C-701
curs_addwch — add a wchar_t character (with
  attributes) to a curses window and advance
cursor, 3X-272
curs_addwchstr — add string of wchar_t charac-
  ters (and attributes) to a curses window, 3X-275
curs_addwchstr — add string of wchar_t characters
  to a curses window and advance cursor,
  3X-277
curs_aecompat — moving the cursor by charac-
  ter, 3X-278
curs_attr — curses character and window attri-
  bute control routines, 3X-280
  Attributes, 3X-280
curs_getch — get (or push back) wchar_t charac-
  ters from curses terminal keyboard, 3X-297
  Function Keys, 3X-297
curs_getwch — get wchar_t character strings
  from curses terminal keyboard, 3X-301
curs_inswch — insert a wchar_t character before
  the character under the cursor in a curses win-
  dow, 3X-313
curs_inswstr — insert wchar_t string before
  character under the cursor in a curses window,
  3X-314
curs_inwch — get a wchar_t character and its
  attributes from a curses window, 3X-316
curs_inwchstr — get a string of wchar_t characters
  (and attributes) from a curses window,
  3X-317
curs_inwstr — get a string of wchar_t characters
  from a curses window, 3X-318
curs_pad — create and display curses pads,
  3X-325
curs_set — set visibility of cursor, 3XC-333
curses — CRT handling and optimization pack-
  age, 3X-246, 3XC-257
  Attributes, Color Pairs, and Renditions,
  3XC-259
  Complex Characters, 3XC-261
  Data Types, 3XC-258
  Display Operations, 3XC-262
  Input Processing, 3XC-265
  Non-Spacing Characters, 3XC-261
  Overlapping Windows, 3XC-262
  Screens, Windows, and Terminals, 3XC-258
  Special Characters, 3XC-264
  curses bell and screen flash routines
  — beep, 3X-282
  — curses_beep, 3X-282
  — flash, 3X-282
  curses borders, horizontal and vertical lines, create
  — border, 3X-284
  — box, 3X-284
  — curses_border, 3X-284
curses borders, horizontal and vertical lines, create, continued
   — wborder, 3X-284
   — whline, 3X-284
   — wvline, 3X-284

curses character and window attribute control routines
   — attroff, 3X-280
   — attron, 3X-280
   — attrset, 3X-280
   — curs_atr, 3X-280
   — standend, 3X-280
   — standout, 3X-280
   — wattroff, 3X-280
   — wattron, 3X-280
   — wattset, 3X-280
   — wstandend, 3X-280
   — wstandout, 3X-280

curses color manipulation routines
   — can_change_colors, 3X-287
   — color_content, 3X-287
   — curs_color, 3X-287
   — has_colors, 3X-287
   — init_color, 3X-287
   — init_pair, 3X-287
   — pair_content, 3X-287
   — start_color, 3X-287

curses cursor and window coordinates
   — curs_getyx, 3X-302
   — getbegyx, 3X-302
   — getmaxyx, 3X-302
   — getparyx, 3X-302
   — getyx, 3X-302

curses environment query routines
   — baudrate, 3X-336
   — curs_termattrs, 3X-336
   — erasechar, 3X-336
   — has_ic, 3X-336
   — has_il, 3X-336
   — killchar, 3X-336
   — longname, 3X-336
   — termattrs, 3X-336
   — termname, 3X-336

curses interfaces to termcap library
   — curs_termcap, 3X-338

curses interfaces to termcap library, continued
   — tgetent, 3X-338
   — tgetflag, 3X-338
   — tgetnum, 3X-338
   — tgetstr, 3X-338
   — tgoto, 3X-338
   — tputs, 3X-338

curses interfaces to terminfo database
   — curs_terminfo, 3X-339
   — del_curterm, 3X-339
   — mvcur, 3X-339
   — putp, 3X-339
   — restartterm, 3X-339
   — set_curterm, 3X-339
   — setterm, 3X-339
   — setupterm, 3X-339
   — tigetflag, 3X-339
   — tigetnum, 3X-339
   — tigetstr, 3X-339
   — tparm, 3X-339
   — tputs, 3X-339
   — vidattr, 3X-339
   — vid_puts, 3X-339

curses library, See also form library, menu library, or panel library
   — adjcurspos, 3X-278
   — curs_alecompat, 3X-278
   — move_next_char, 3X-278
   — move_prev_char, 3X-278
   — wadjcurspos, 3X-278
   — wmovenext, 3X-278
   — wmoverest, 3X-278
   — wmoveprev, 3X-278

Index—14
curses pads, create and display — curs_pad, continued
  pechochar, 3X-325
  pechowchar, 3X-325
  pnoutrefresh, 3X-325
  prefresh, 3X-325
  subpad, 3X-325

  curses refresh control routines
    — curs_touch, 3X-342
    — is_linetouched, 3X-342
    — is_wintouched, 3X-342
    — touchline, 3X-342
    — touchwin, 3X-342
    — untouchwin, 3X-342
    — wtouchln, 3X-342

  curses screen initialization and manipulation routines
    — curs_initscr, 3X-305
    — delscreen, 3X-305
    — endwin, 3X-305
    — initscr, 3X-305
    — isendwin, 3X-305
    — newterm, 3X-305
    — set_term, 3X-305

  curses screen, read/write from/to file
    — curs_scr_dump, 3X-331
    — scr_dump, 3X-331
    — scr_init, 3X-331
    — scr_restore, 3X-331
    — scr_set, 3X-331

  curses soft label routines
    — curs_slk, 3X-334
    — slk_attroff, 3X-334
    — slk_attron, 3X-334
    — slk_attrset, 3X-334
    — slk_clear, 3X-334
    — slk_init, 3X-334
    — slk_label, 3X-334
    — slk_noutrefresh, 3X-334
    — slk_refresh, 3X-334
    — slk_restore, 3X-334
    — slk_set, 3X-334
    — slk_touch, 3X-334

  curses terminal input option control routines, continued
    — curs_inopts, 3X-307
    — echo, 3X-307
    — halfdelay, 3X-307
    — intrflush, 3X-307
    — keypad, 3X-307
    — meta, 3X-307
    — nobreak, 3X-307
    — nodelay, 3X-307
    — noecho, 3X-307
    — noqiflush, 3X-307
    — noraw, 3X-307
    — notimeout, 3X-307
    — qiflush, 3X-307
    — raw, 3X-307
    — timeout, 3X-307
    — typeahead, 3X-307
    — wtimeout, 3X-307

  curses terminal keyboard
    — curs_getstr, 3X-296
    — getstr, 3X-296
    — mvgetstr, 3X-296
    — mvwgetstr, 3X-296
    — wgetnstr, 3X-296
    — wgetstr, 3X-296

  curses terminal keyboard, get characters
    — curs_getch, 3X-292
    — getch, 3X-292
    — mvgetch, 3X-292
    — mvwgetch, 3X-292
    — ungetch, 3X-292
    — wgetch, 3X-292

  curses terminal output option control routines
    — clearok, 3X-322
    — curs_outopts, 3X-322
    — idcok, 3X-322
    — idlok, 3X-322
    — immedok, 3X-322
    — leaveok, 3X-322
    — nl, 3X-322
    — nonl, 3X-322
    — scrollok, 3X-322
    — setscren, 3X-322
    — wsetscren, 3X-322

Index—15
curses window background manipulation routines
  — bkgd, 3X-283
  — bkgdset, 3X-283
  — curs_bkgd, 3X-283
  — wbkgd, 3X-283
  — wbkgdset, 3X-283

  curses window, clear all or part, continued
  — wclrtobot, 3X-286
  — wclrtoeol, 3X-286
  — werase, 3X-286

curses window, convert formatted input
  — curs_scanw, 3X-330
  — mvscanw, 3X-330
  — mvwscanw, 3X-330
  — scanw, 3X-330
  — vscanw, 3X-330
  — wscanw, 3X-330

  curses window cursor
  — curs_move, 3X-321
  — move, 3X-321
  — wmove, 3X-321

curses window, add character and advance cursor
  — addch, 3X-268
  — curs_addch, 3X-268
  — echochar, 3X-268
  — mvaddch, 3X-268
  — mvwaddch, 3X-268
  — waddch, 3X-268
  — wechochar, 3X-268

  curses window, add string of characters
  — addchnstr, 3X-270
  — addchstr, 3X-270
  — curs_addchstr, 3X-270
  — mvaddchstr, 3X-270
  — mvwaddchstr, 3X-270
  — waddchstr, 3X-270
  — waddchstr, 3X-270

  curses window, add string of characters and advance cursor
  — addnstr, 3X-271
  — addstr, 3X-271
  — curs_addstr, 3X-271
  — mvaddstr, 3X-271
  — mvwaddstr, 3X-271
  — waddnstr, 3X-271
  — waddstr, 3X-271

  curses window, clear all or part
  — clear, 3X-286
  — clrtoeol, 3X-286
  — curs_clear, 3X-286
  — erase, 3X-286
  — wclear, 3X-286

  curses window, delete and insert lines
  — curs_deleteln, 3X-291
  — deleteln, 3X-291
  — insdelln, 3X-291
  — insertln, 3X-291
  — wdeleteln, 3X-291
  — winsertln, 3X-291

  curses window, delete character under cursor
  — curs_delch, 3X-290
  — delch, 3X-290
  — mvdelch, 3X-290
  — mvwdelch, 3X-290
  — wdelch, 3X-290

  curses window, get character and its attributes
  — curs_inch, 3X-303
  — inch, 3X-303
  — mvinch, 3X-303
  — mvwinch, 3X-303
  — winch, 3X-303

  curses window, get string of characters
  — curs_inchnstr, 3X-304
  — curs_instr, 3X-312
  — inchstr, 3X-304
  — innstr, 3X-312
  — instr, 3X-312
  — mvinchstr, 3X-304
  — mvinchnstr, 3X-304
  — mvinstr, 3X-312
  — mvinstr, 3X-312
  — mvwinchstr, 3X-304
  — mvwinchstr, 3X-304
  — mvwinnstr, 3X-312
  — mvwinnstr, 3X-312
curses window, get string of characters, continued
   — mvwinst, 3X-312
   — wchinst, 3X-304
   — wchstr, 3X-304
   — winnstr, 3X-312
   — winstr, 3X-312

curses window, insert character before character under cursor
   — curs_insch, 3X-310
   — insch, 3X-310
   — mvinsch, 3X-310
   — mvwinsch, 3X-310
   — winsch, 3X-310

curses window, insert string before character under cursor
   — curs_instr, 3X-311
   — insnstr, 3X-311
   — instr, 3X-311
   — mvinsstr, 3X-311
   — mvwinsstr, 3X-311
   — mvwinsnstr, 3X-311
   — winnsstr, 3X-311
   — winsstr, 3X-311

curses window, scroll
   — curs_scroll, 3X-332
   — scrl, 3X-332
   — scroll, 3X-332
   — wscrl, 3X-332

curses windows and lines, refresh
   — curs_refresh, 3X-328
   — doupdate, 3X-328
   — redrawwin, 3X-328
   — refresh, 3X-328
   — wdrawwin, 3X-328
   — wrefresh, 3X-328

curses windows, create
   — curs_window, 3X-345
   — delwin, 3X-345
   — derwin, 3X-345
   — dupwin, 3X-345
   — mvderwin, 3X-345
   — mvwin, 3X-345
   — newwin, 3X-345

curses windows, create, continued
   — subwin, 3X-345
   — syncok, 3X-345
   — wcredsync, 3X-345
   — wsyncdown, 3X-345
   — wsyncup, 3X-345

curses windows, overlap and manipulate
   — copywin, 3X-324
   — curs_overlay, 3X-324
   — overlay, 3X-324
   — overwrite, 3X-324

curses windows, print formatted output
   — curs_printw, 3X-327
   — mvprintw, 3X-327
   — mvwprintw, 3X-327
   — printw, 3X-327
   — vwprintw, 3X-327
   — wprintw, 3X-327

curses, low-level routines
   — curs_kernel, 3X-319
   — curs_set, 3X-319
   — def_prog_mode, 3X-319
   — def_shell_mode, 3X-319
   — getsyx, 3X-319
   — napms, 3X-319
   — reset_prog_mode, 3X-319
   — reset_shell_mode, 3X-319
   — resettty, 3X-319
   — ripoffline, 3X-319
   — savetty, 3X-319
   — settty, 3X-319

cuserid — get character-string representation of login name of user, 3S-347

D

data base subroutines — dbm, 3B-348
   dbmclose, 3B-348
   dbminit, 3B-348
   delete, 3B-348
   fetch, 3B-348
   firstkey, 3B-348
   nextkey, 3B-348
   store, 3B-348

database functions
   — dbm_clearerr, 3-350
database functions, continued
   — dbm_close, 3-350
   — dbm_delete, 3-350
   — dbm_error, 3-350
   — dbm_fetch, 3-350
   — dbm_firstkey, 3-350
   — dbm_nextkey, 3-350
   — dbm_open, 3-350
   — dbm_store, 3-350

date and time
   convert to string — asctime, 3C-239
   convert user format date and time — getdate, 3C-703
   — gettimeofday, 3C-787

date and time conversion — strptime, 3C-1502
dbm — data base subroutines, 3B-348
dbm_clearerr — database functions, 3-350
dbm_close — database functions, 3-350
dbm_delete — database functions, 3-350
dbm_error — database functions, 3-350
dbm_fetch — database functions, 3-350
dbm_firstkey — database functions, 3-350
dbm_nextkey — database functions, 3-350
dbm_open — database functions, 3-350
   — dbm_store — database functions, 3-350
dbmclose — data base subroutines, 3B-348
dbminit — data base subroutines, 3B-348
debugging memory allocator
   — calloc, 3X-1778
   — cfree, 3X-1778
   — free, 3X-1778
   — mallinfo, 3X-1778
   — malloc, 3X-1778
   — mallocpt, 3X-1778
   — memalign, 3X-1778
   — realloc, 3X-1778
   — valloc, 3X-1778
decimal record from double-precision floating —
   double_to_decimal, 3-524
decimal record from extended-precision floating —
   extended_to_decimal, 3-524
decimal record from quadruple-precision floating —
   quadruple_to_decimal, 3-524
decimal record from single-precision floating —
   single_to_decimal, 3-524
decimal record to double-precision floating —
   decimal_to_double, 3-354
decimal record to extended-precision floating —
   decimal_to_extended, 3-354
decimal record to quadruple-precision floating —
   decimal_to_quadruple, 3-354
decimal record to single-precision floating —
   decimal_to_single, 3-354
decimal_to_double — decimal record to double-precision floating, 3-354
decimal_to_extended — decimal record to extended-precision floating, 3-354
decimal_to_quadruple — decimal record to quadruple-precision floating, 3-354
decimal_to_single — decimal record to single-precision floating, 3-354
decode a C++ encoded symbol name
   — cplusplus_demangle, 3-363
   — demangle, 3-363
decompose floating-point number
   — modf, 3C-1006
   — modff, 3C-1006
defprog_mode — save/restore terminal modes, 3XC-355
defshell_mode — save/restore terminal modes, 3XC-355
define character class — wctype, 3C-1799
define default catalog — setcat, 3C-1409
define the label for pfmt() and lfmt(). — setlabel, 3C-1418
define wide-character mapping — wctrans, 3C-1798
delCurterm — free space pointed to by terminal
   , 3XC-358
delay_output — delays output, 3XC-356
delch — remove a character, 3XC-357
deletem — delete a character, 3B-348
delete — data base subroutines, 3B-348
delete — delete a window — delwin, 3XC-362
deletein — remove a line, 3XC-360
delwin — delete a window, 3XC-362

Index—18
demangle — decode a C++ encoded symbol name, 3-363
derwin — create a new window or subwindow, 3XC-364
descriptions of XFN status codes —
xfn_status_codes, 3N-1831
detach a name from a STREAMS-based file descriptor — fdetach, 3C-513
determine insert/delete character/line capability —
  has_ic, 3XC-814
  has_il, 3XC-814
device id interfaces for a user environment —
devid_compare, 3-366
  devid_deviceid_to_nmplist, 3-366
  devid_free, 3-366
  devid_free_nmplist, 3-366
  devid_get, 3-366
  devid_get_minor_name, 3-366
  devid_sizeof, 3-366
device number
  manage — makedev, major, minor, 3C-946
devid_compare — device id interfaces for a user environment, 3-366
devid_deviceid_to_nmplist — device id interfaces for a user environment, 3-366
devid_free — device id interfaces for a user environment, 3-366
devid_free_nmplist — device id interfaces for a user environment, 3-366
devid_get — device id interfaces for a user environment, 3-366
devid_get_minor_name — device id interfaces for a user environment, 3-366
devid_sizeof — device id interfaces for a user environment, 3-366
dladdr — translate address to symbolic information, 3X-378
dlclose — close a shared object, 3X-380
dldump — create new file from dynamic object component of calling process, 3X-381
dlerror — get diagnostic information, 3X-388
dlinfo — dynamic load information, 3X-389
dlopen — open a shared object, 3X-391
dlsym — get address of symbol in shared object, 3X-395
DmiAddComponent — Management Interface database administration functions, 3X-397
DmiAddGroup — Management Interface database
administration functions, 3X-397
DmiAddLanguage — Management Interface database administration functions, 3X-397
DmiAddRow — Management Interface operation functions, 3X-401
DmiDeleteComponent — Management Interface database administration functions, 3X-397
DmiDeleteGroup — Management Interface database administration functions, 3X-397
DmiDeleteLanguage — Management Interface database administration functions, 3X-397
DmiDeleteRow — Management Interface operation functions, 3X-401
DmiGetAttribute — Management Interface operation functions, 3X-401
DmiGetConfig — Management Interface initialization functions, 3X-406
DmiGetMultiple — Management Interface operation functions, 3X-401
DmiGetVersion — Management Interface initialization functions, 3X-406
DmiListAttributes — Management Interface listing functions, 3X-409
DmiListClassNames — Management Interface listing functions, 3X-409
DmiListComponents — Management Interface listing functions, 3X-409
DmiListComponentsByClass — Management Interface listing functions, 3X-409
DmiListGroups — Management Interface listing functions, 3X-409
DmiListLanguages — Management Interface listing functions, 3X-409
DmiOriginateEvent — Service Provider functions for components, 3X-415
DmiRegister — Management Interface initialization functions, 3X-406
DmiRegisterCi — Service Provider functions for components, 3X-415
DmiSetAttribute — Management Interface operation functions, 3X-401
DmiSetConfig — Management Interface initialization functions, 3X-406
DmiSetMultiple — Management Interface operation functions, 3X-401
DmiUnregister — Management Interface initialization functions, 3X-406
DmiUnRegisterCi — Service Provider functions for components, 3X-415
dn_comp — resolver routines, 3N-1296
dn_expand — resolver routines, 3N-1296
doconfig — execute a configuration script, 3N-417
door_bind — bind or unbind the current thread with the door server pool, 3X-419
door_call — invoke the function associated with a door descriptor, 3X-422
door_create — create a door descriptor, 3X-425
door_cred — return credential information associated with the client, 3X-427
door_info — return information associated with a door descriptor, 3X-428
door_return — return from a door invocation, 3X-429
door_revoke — revoke access to a door descriptor, 3X-430
door_server_create — specify an alternative door server thread creation function, 3X-431
door_unbind — bind or unbind the current thread with the door server pool, 3X-419
double_to_decimal — decimal record from double-precision floating, 3-524
doupdate — refresh windows and lines, 3XC-433
dup2 — duplicate an open file descriptor, 3C-436
duplicate a window — dupwin, 3XC-437
duplicate an open file descriptor — dup2, 3C-436
dupwin — duplicate a window, 3XC-437
dynamic linking
close a shared object — dlclose, 3X-380
create new file from dynamic object component — dldump, 3X-381
get address of symbol in shared object — dlinsym, 3X-395
get diagnostic information — dlerror, 3X-388
open a shared object — dlopen, 3X-391

Index−20
dynamic load information — dlinfo, 3X-389

echo — enable/disable terminal echo, 3XC-438
echo wchar — add a complex character and refresh window, 3X-440
echochar — add a single-byte character and refresh window, 3XC-439
echowchar — add a wchar_t character (with attributes) to a curses window and advance cursor, 3X-272

econvert — convert number to ASCII, 3-441
ecvt — convert number to ASCII, 3-441
data — last location in program, 3C-489
ef — object file access library, 3E-445

g get entries from name list — nlist, 3E-1093
ef_begin — process ELF object files, 3E-457
ef_cntl — control an elf file descriptor, 3E-463
ef_end — process ELF object files, 3E-457
ef_errmsg — error handling, 3E-464
ef_errno — error handling, 3E-464
ef_fill — set fill byte, 3E-465
ef_flagdata — manipulate flags, 3E-466
ef_flagehdr — manipulate flags, 3E-466
ef_faglelf — manipulate flags, 3E-466
ef_flagphdr — manipulate flags, 3E-466
ef_flagshdr — manipulate flags, 3E-466
ef_getarhdr — retrieve archive member header, 3E-468
ef_getarsym — retrieve archive symbol table, 3E-470
ef_getbase — get the base offset for an object file, 3E-471
ef_getdata — get section data, 3E-472
ef_getident — retrieve file identification data, 3E-476
ef_getscn — get section information, 3E-478

ef_hash — compute hash value, 3E-480
ef_kind — determine file type, 3E-481
ef_memory — process ELF object files, 3E-457
efndxscn — get section information, 3E-478

ef_newdata — get section data, 3E-472
ef_newscn — get section information, 3E-478

ef_next — process ELF object files, 3E-457
ef_nextscn — get section information, 3E-478
ef_rand — process ELF object files, 3E-457
ef_rdata — get section data, 3E-472
ef_rawdata — retrieve uninterpreted file contents, 3E-482

ef_strptr — make a string pointer, 3E-483
ef_update — update an ELF descriptor, 3E-484
ef_version — coordinate ELF library and application versions, 3E-487
ef_fsize — return the size of an object file type, 3E-451

ef32_getehdr — retrieve class-dependent object file header, 3E-452
ef_getphdr — retrieve class-dependent program header table, 3E-453
ef32_getshdr — retrieve class-dependent section header, 3E-454
ef32_newehdr — retrieve class-dependent object file header, 3E-452

ef_newphdr — retrieve class-dependent program header table, 3E-453
ef32_xlatetof — class-dependent data translation, 3E-455
ef32_xlatetom — class-dependent data translation, 3E-455

eulate the termcap database
— tgetent, 3XC-1601
— tgetflag, 3XC-1601
— tgetnum, 3XC-1601
— tgetstr, 3XC-1601
— tgoto, 3XC-1601

enable or disable cancellation —
— pthread_setcancelstate, 3T-1227

enable/disable half-delay mode — halfdelay, 3XC-813

enable/disable hardware insert-character and delete-character features — idcok, 3XC-826

enable/disable keypad handling — keypad, 3XC-876
enable/disable meta keys — meta, 3XC-995
enable/disable newline control
— nl, 3XC-1091
— nonl, 3XC-1091
enable/disable terminal echo
— echo, 3XC-438
— noecho, 3XC-438
enabling or disabling cancellation —
pthread_setcancelstate, 3T-1227
encryption
determine whether a buffer of characters is encrypted — isencrypt, 3G-857
end — last location in program, 3C-489
endac — get audit control file information, 3-684
endauclass — close audit_class database file
— 3-686
endauevent — close audit_event database file
— 3-689
endauser — get audit_user database entry, 3-691
dgrent — get group entry from database
— 3C-712
endhostent — network host database functions
— 3XN-490
endnetent — network database functions
— 3XN-492
endprotoent — network protocol database functions
— 3XN-494
endpwent — get password entry from user database
— 3C-754
endservent — get service entry, 3N-765, 3XN-496
endspent — get shadow password database entry
— 3C-776
endusershell() — function, 3C-790
endutent — access utmp file entry, 3C-791
endutxent — access utmpx file entry, 3C-793
endwin — restore initial terminal environment
— 3XC-498
environment name
return value — getenv, 3C-709
environment variables
change or add value — putenv, 3C-1243
erase — clear a window, 3XC-206, 3-1166
erasechar — return current ERASE or KILL characters, 3XC-499
erasewchar — return current ERASE or KILL characters, 3XC-499
erf — error and complementary error functions
— 3M-500
erfc — error and complementary error functions
— 3M-500
error and complementary error functions
— erf, 3M-500
— erfc, 3M-500
error messages
gem string — strerror, 3C-1487, 3N-1732
error messages, system
print — perror, 3C-1161
establish a transport endpoint — t_open, 3N-1677
etext — last location in program, 3C-489
Ethernet address mapping operations
— ethers, 3N-501
ethers — Ethernet address mapping operations
— 3N-501
EUC character bytes
— euclen, 3C-503
EUC characters
convert a string of EUC characters from the stream to Process Code — getws
— 3S-805
convert a string of EUC characters from the stream to Process Code — getwfs
— 3S-805
convert a string of Process Code characters to EUC characters and put it on a stream
— putws, 3S-1250
EUC codeset, get information
— getwidth, 3C-803
EUC codesets, get information
— cset, 3C-237
— csetcol, 3C-237
— csetw, 3C-237
— csetno, 3C-237
— wcsetno, 3C-237
EUC display width
— eucwcol, 3C-503
EUC display width, continued
— eucscol, 3C-503
euccol — get EUC character display width, 3C-503
euclen — get EUC byte length, 3C-503
Euclidean distance function — hypot, 3M-821
eucscol — get EUC string display width, 3C-503
Executable and Linking Format, See elf
exit — terminate process, 3C-504
exit program add routine — atexit, 3C-153
exp — exponential function, 3M-505
expm1 — computes exponential functions, 3M-506
exponential function — exp, 3M-505
Extended Unix Code, See EUC
extended_to_decimal — decimal record from extended-precision floating, 3-524
external data representation See XDR, 3N-1808
extract mantissa and exponent from double precision number — frexp, 3C-672

F
fabs — absolute value function, 3M-507
fattach — attach a STREAMS-based file descriptor to an object in the file system name space, 3C-508
fclose — close a stream, 3S-510
fchroot — convert number to ASCII, 3-441
fdopen — associate a stream with a file descriptor, 3S-514
fdatasync — synchronize a file’s data, 3R-512
fdetach — detach a name from a STREAMS-based file descriptor, 3C-513
flock — synchronize a file’s data, 3R-512
fetch — data base subroutines, 3B-348
fflush — flush a stream, 3S-517
ffs — find first set bit, 3C-519
fgetgrent — get group entry from file, 3C-712
fgetgrent_r — get group entry from file, 3C-712
fgetpos — get current file position information, 3S-520
fgetpwent — get password entry from a file, 3C-754
fgetpwent_r — get password entry from a file, 3C-754
fgetspent — get shadow password database entry, 3C-776
fgetspent_r — get shadow password database entry(reentrant), 3C-776
fgetwc — get a wide-character code from a stream, 3S-521
fgetws — em convert a string of EUC characters from the stream to Process Code, 3S-805
FIFO
create a new one — mkfifo, 3C-997
file descriptor
duplicate an open one — dup2, 3C-436
STREAMS-based, attach to an object in file system name space — fattach, 3C-508
test for a STREAMS file — isastream, 3C-855
file descriptors
apply or remove advisory lock on open file — flock, 3B-526
file name
make a unique one — mktemp, 3C-999
make a unique file name — mkstemp, 3C-998
file pointer in a stream
reposition — fsetpos, fgetpos, 3S-676
file tree
recursively descend — ftw, 3C-682
file_to_decimal — decimal record from character stream, 3-1499
files
allows sections of file to be locked — lockf, 3C-930
optimizing usage of files — directio, 3C-373
— remove, 3C-1294
report parent directory of file path name — dirname, 3C-375

Index—23
files, continued
search for named file in named directories — pathfind, 3G-1158
set a file to a specified length — truncate, 3C-1715
synchronize a file's in-memory state with that on the physical medium — fsync, 3C-677
filter — disable use of certain terminal capabilities, 3XC-523
filter expression for attribute search
— fn_search_filter_arguments, 3N-615
— fn_search_filter_assign, 3N-615
— fn_search_filter_copy, 3N-615
— fn_search_filter_create, 3N-615
— fn_search_filter_destroy, 3N-615
— fn_search_filter_expression, 3N-615
— FN_search_filter_t, 3N-615
find pathname of a terminal
— ttyname, 3C-1736
— ttyname_r, 3C-1736
firstkey — data base subroutines, 3B-348
flash — activate audio-visual alarm, 3XC-166
floating-point number
convert to string — ecvt, 3C-443
floating-point number, determine type
— finite, 3C-860
— fpclass, 3C-860
— isnan, 3C-860
— isnand, 3C-860
— isnanf, 3C-860
— unordered, 3C-860
floating-point remainder value function — fmod, 3M-532
flock — apply or remove an advisory lock on an open file, 3B-526
flockfile — acquire and release stream lock, 3S-528
floor — floor function, 3M-530
floor function — floor, 3M-530
flush a stream — fflush, 3S-517
flush non-transmitted output data, non-read input data or both — tcflush, 3S-1546
flush output in tty on interrupt — intrflush, 3XC-852
flushinp — discard type-ahead characters, 3X-531
fmod — floating-point remainder value function, 3M-532
fmsg — display a message on stderr or system console, 3C-533
fn_attr_bind — bind a reference to a name and associate attributes with named object, 3N-538
fn_attr_create_subcontext — create subcontext and associate attributes, 3N-539
fn_attr_ext_search — search for names whose attributes satisfy filter, 3N-540
fn_attr_get — return specified attribute associated with name, 3N-547
fn_attr_get_ids — get list of attribute identifiers, 3N-548
fn_attr_get_values — return values of an attribute, 3N-549
fn_attr_modify — modify specified attribute associated with name, 3N-553
fn_attr_multi_get — return multiple attributes associated with named object, 3N-557
fn_attr_multi_modify — modify multiple attributes associated with named object, 3N-561
fn_attr_search — search for atomic name with specified attributes in single context, 3N-562
fn_attribute_add — an XFN attribute, 3N-551
fn_attribute_assign — an XFN attribute, 3N-551
fn_attribute_copy — an XFN attribute, 3N-551
fn_attribute_create — an XFN attribute, 3N-551
fn_attribute_destroy — an XFN attribute, 3N-551
fn_attribute_first — an XFN attribute, 3N-551
fn_attribute_identifier — an XFN attribute, 3N-551
fn_attribute_next — an XFN attribute, 3N-551
fn_attribute_remove — an XFN attribute, 3N-551

Index—24
fn_attribute_syntax — an XFN attribute, 3N-551
FN_attribute_t — an XFN attribute, 3N-551
fn_attribute_valuecount — an XFN attribute, 3N-551
fn_attrmodlist_add — a list of attribute modifications, 3N-555
fn_attrmodlist_assign — a list of attribute modifications, 3N-555
fn_attrmodlist_copy — a list of attribute modifications, 3N-555
fn_attrmodlist_count — a list of attribute modifications, 3N-555
fn_attrmodlist_create — a list of attribute modifications, 3N-555
fn_attrmodlist_destroy — a list of attribute modifications, 3N-555
fn_attrmodlist_first — a list of attribute modifications, 3N-555
fn_attrmodlist_next — a list of attribute modifications, 3N-555
FN_attrmodlist_t — a list of attribute modifications, 3N-555
fn_attrset_add — a set of XFN attributes, 3N-567
fn_attrset_assign — a set of XFN attributes, 3N-567
fn_attrset_copy — a set of XFN attributes, 3N-567
fn_attrset_count — a set of XFN attributes, 3N-567
fn_attrset_create — a set of XFN attributes, 3N-567
fn_attrset_destroy — a set of XFN attributes, 3N-567
fn_attrset_first — a set of XFN attributes, 3N-567
fn_attrset_get — a set of XFN attributes, 3N-567
fn_attrset_next — a set of XFN attributes, 3N-567
fn_attrset_remove — a set of XFN attributes, 3N-567
FN_attrset_t — a set of XFN attributes, 3N-567
fn_bindinglist_destroy — list the atomic names and references bound in a context, 3N-592
fn_bindinglist_next — list the atomic names and references bound in a context, 3N-592
FN_bindinglist_t — list the atomic names and references bound in a context, 3N-592
fn_composite_name_append_comp — component names spanning multiple naming systems, 3N-571
fn_composite_name_append_name — component names spanning multiple naming systems, 3N-571
fn_composite_name_assign — component names spanning multiple naming systems, 3N-571
fn_composite_name_copy — component names spanning multiple naming systems, 3N-571
fn_composite_name_count — component names spanning multiple naming systems, 3N-571
fn_composite_name_create — component names spanning multiple naming systems, 3N-571
fn_composite_name_delete_comp — component names spanning multiple naming systems, 3N-571
fn_composite_name_destroy — component names spanning multiple naming systems, 3N-571
fn_composite_name_first — component names spanning multiple naming systems, 3N-571
fn_composite_name_from_str — component names spanning multiple naming systems, 3N-571
fn_composite_name_from_string — component names spanning multiple naming systems, 3N-571
fn_composite_name_insert_comp — component names spanning multiple naming systems, 3N-571
ponent names spanning multiple naming systems, 3N-571
fn_composite_name_insert_name — component names spanning multiple naming systems, 3N-571
fn_composite_name_is_empty — component names spanning multiple naming systems, 3N-571
fn_composite_name_is_equal — component names spanning multiple naming systems, 3N-571
fn_composite_name_is_prefix — component names spanning multiple naming systems, 3N-571
fn_composite_name_is_suffix — component names spanning multiple naming systems, 3N-571
fn_composite_name_last — component names spanning multiple naming systems, 3N-571
fn_composite_name_next — component names spanning multiple naming systems, 3N-571
fn_composite_name_prefix — component names spanning multiple naming systems, 3N-571
fn_composite_name_prepend_comp — component names spanning multiple naming systems, 3N-571
fn_composite_name_prepend_name — component names spanning multiple naming systems, 3N-571
fn_composite_name_prev — component names spanning multiple naming systems, 3N-571
fn_composite_name_suffix — component names spanning multiple naming systems, 3N-571
FN_composite_name_t — component names spanning multiple naming systems, 3N-571
fn_compound_name_append_comp — an XFN compound name, 3N-575
fn_compound_name_assign — an XFN compound name, 3N-575
fn_compound_name_count — an XFN compound name, 3N-575
fn_compound_name_delete_all — an XFN compound name, 3N-575
fn_compound_name_delete_comp — an XFN compound name, 3N-575
fn_compound_name_destroy — an XFN compound name, 3N-575
fn_compound_name_first — an XFN compound name, 3N-575
fn_compound_name_from_syntax_attrs — an XFN compound name, 3N-575
fn_compound_name_get_syntax_attrs — an XFN compound name, 3N-575
fn_compound_name_insert_comp — an XFN compound name, 3N-575
fn_compound_name_is_empty — an XFN compound name, 3N-575
fn_compound_name_is_equal — an XFN compound name, 3N-575
fn_compound_name_is_prefix — an XFN compound name, 3N-575
fn_compound_name_is_suffix — an XFN compound name, 3N-575
fn_compound_name_last — an XFN compound name, 3N-575
fn_compound_name_next — an XFN compound name, 3N-575
fn_compound_name_prefix — an XFN compound name, 3N-575
fn_compound_name_prepend_comp — an XFN compound name, 3N-575
fn_compound_name_prepend_name — an XFN compound name, 3N-575
fn_compound_name_prev — an XFN compound name, 3N-575
fn_compound_name_suffix — an XFN compound name, 3N-575
FN_compound_name_t — an XFN compound name, 3N-575
fn_ctx_bind — bind a reference to a name, 3N-579
fn_ctx_equivalent_name — construct

Index—26
equivalent name in same context, 3N-583
fn_ctx_handle_from_initial — return a handle to the Initial Context, 3N-588
fn_ctx_handle_from_ref — construct a handle to a context object using the given reference, 3N-590
fn_ctx_list_bindings — list the atomic names and references bound in a context, 3N-592
fn_ctx_list_names — list the atomic names bound in a context, 3N-593
fn_ctx_lookup_link — look up the link reference bound to a name, 3N-597
fn_ctx_rename — rename the name of a binding, 3N-598
FN_ctx_t — an XFN context, 3N-601
fn_ext_searchlist_destroy — search for names whose attributes satisfy filter, 3N-540
fn_ext_searchlist_next — search for names whose attributes satisfy filter, 3N-540
FN_ext_searchlist_t — search for names whose attributes satisfy filter, 3N-540
FN_identifier_t — an XFN identifier, 3N-604
fn_multigetlist_destroy — return multiple attributes associated with named object, 3N-557
fn_multigetlist_next — return multiple attributes associated with named object, 3N-557
FN_multigetlist_t — return multiple attributes associated with named object, 3N-557
fn_namelist_destroy — list the atomic names bound in a context, 3N-593
fn_namelist_next — list the atomic names bound in a context, 3N-593
FN_namelist_t — list the atomic names bound in a context, 3N-593
fn_ref_addr_assign — an address in an XFN reference, 3N-607
fn_ref_addr_destroy — an address in an XFN reference, 3N-607
fn_ref_addr_length — an address in an XFN reference, 3N-607
FN_ref_addr_t — an address in an XFN reference, 3N-607
fn_ref_addr_type — an address in an XFN reference, 3N-607
fn_ref_addrcount — an XFN reference, 3N-609
fn_ref_append_addr — an XFN reference, 3N-609
fn_ref_assign — an XFN reference, 3N-609
fn_ref_copy — an XFN reference, 3N-609
fn_ref_create — an XFN reference, 3N-609
fn_ref_create_link — an XFN reference, 3N-609
fn_ref_delete_addr — an XFN reference, 3N-609
fn_ref_delete_all — an XFN reference, 3N-609
fn_ref_description — an XFN reference, 3N-609
fn_ref_destroy — an XFN reference, 3N-609
fn_ref_first — an XFN reference, 3N-609
fn_ref_insert_addr — an XFN reference, 3N-609
fn_ref_is_link — an XFN reference, 3N-609
fn_ref_link_name — an XFN reference, 3N-609
fn_ref_next — an XFN reference, 3N-609
fn_ref_prepend_addr — an XFN reference, 3N-609
FN_ref_t — an XFN reference, 3N-609
fn_ref_type — an XFN reference, 3N-609
fn_search_control_assign — options for attribute search, 3N-612
fn_search_control_copy — options for attribute search, 3N-612
fn_search_control_create — options for attribute search, 3N-612
fn_search_control_destroy — options for attribute search, 3N-612
fn_search_control_follow_links — options for attribute search, 3N-612
fn_search_control_max_names — options for attribute search, 3N-612
fn_search_control_return_attr_ids — options for attribute search, 3N-612
fn_search_control_return_ref — options for attribute search, 3N-612
fn_search_control_scope — options for attribute search, 3N-612
FN_search_control_t — options for attribute search, 3N-612
fn_search_filter_arguments — filter expression for attribute search, 3N-615
fn_search_filter_assign — filter expression for attribute search, 3N-615
fn_search_filter_copy — filter expression for attribute search, 3N-615
fn_search_filter_create — filter expression for attribute search, 3N-615
fn_search_filter_destroy — filter expression for attribute search, 3N-615
fn_search_filter_expression — filter expression for attribute search, 3N-615
FN_search_filter_t — filter expression for attribute search, 3N-615
BNF of Filter Expression, 3N-615
Extended Operations, 3N-618
Precedence, 3N-616
Relational Operators, 3N-617
Specification of Filter Expression, 3N-616
Wildcarded Strings, 3N-617
fn_searchlist_destroy — terminate search for atomic name with specified attributes in single context, 3N-562
fn_searchlist_next — search for next atomic name with specified attributes in single context, 3N-562
FN_searchlist_t — search for atomic name with specified attributes in single context, 3N-562
fn_status_advance_by_name — an XFN status object, 3N-622
fn_status_append_remaining_name — an XFN status object, 3N-622
fn_status_append_resolved_name — an XFN status object, 3N-622
fn_status_assign — an XFN status object, 3N-622
fn_status_code — an XFN status object, 3N-622
fn_status_copy — an XFN status object, 3N-622
fn_status_create — an XFN status object, 3N-622
fn_status_description — an XFN status object, 3N-622
fn_status_destroy — an XFN status object, 3N-622
fn_status_diagnostic_message — an XFN status object, 3N-622
fn_status_is_success — an XFN status object, 3N-622
fn_status_link_code — an XFN status object, 3N-622
fn_status_link_diagnostic_message — an XFN status object, 3N-622
fn_status_link_remaining_name — an XFN status object, 3N-622
fn_status_link_resolved_name — an XFN status object, 3N-622
fn_status_link_resolved_ref — an XFN status object, 3N-622
fn_status_remaining_name — an XFN status object, 3N-622
fn_status_resolved_name — an XFN status object, 3N-622
fn_status_resolved_ref — an XFN status object, 3N-622
fn_status_set — an XFN status object, 3N-622
fn_status_set_code — an XFN status object, 3N-622
fn_status_set_diagnostic_message — an XFN status object, 3N-622
fn_status_set_link_code — an XFN status object, 3N-622
fn_status_set_link_diagnostic_message —
an XFN status object, 3N-622
fn_status_set_link_remaining_name — an XFN status object, 3N-622
fn_status_set_link_resolved_name — an XFN status object, 3N-622
fn_status_set_link_resolved_ref — an XFN status object, 3N-622
fn_status_set_remaining_name — an XFN status object, 3N-622
fn_status_set_resolved_name — an XFN status object, 3N-622
fn_status_set_resolved_ref — an XFN status object, 3N-622
fn_status_set_success — an XFN status object, 3N-622
FN_status_t — an XFN status object, 3N-622
fn_string_assign — a character string, 3N-626
fn_string_bytecount — a character string, 3N-626
fn_string_charcount — a character string, 3N-626
fn_string_code_set — a character string, 3N-626
fn_string_compare — a character string, 3N-626
fn_string_compare_substring — a character string, 3N-626
fn_string_contents — a character string, 3N-626
fn_string_copy — a character string, 3N-626
fn_string_create — a character string, 3N-626
fn_string_destroy — a character string, 3N-626
fn_string_from_composite_name — component names spanning multiple naming systems, 3N-571
fn_string_from_compound_name — an XFN compound name, 3N-575
fn_string_from_contents — a character string, 3N-626
fn_string_from_str — a character string, 3N-626
fn_string_from_str_n — a character string, 3N-626
fn_string_from_strings — a character string, 3N-626
fn_string_from_substring — a character string, 3N-626
fn_string_is_empty — a character string, 3N-626
fn_string_next_substring — a character string, 3N-626
fn_string_prev_substring — a character string, 3N-626
fn_string_str — a character string, 3N-626
FN_string_t — a character string, 3N-626
fn_valuelist_destroy — return values of an attribute, 3N-549
fn_valuelist_next — return values of an attribute, 3N-549
FN_valuelist_t — return values of an attribute, 3N-549
fnmatch — match filename or path name, 3C-605
FNS component names spanning multiple naming systems, See FN_composite_name_t
fn_attr_bind — bind a reference to a name and associate attributes with named object, 3N-538
fn_attr_create_subcontext — create subcontext and associate attributes, 3N-539
fn_attr_ext_search — search for names whose attributes satisfy filter, 3N-540
fn_attr_search — search for atomic name with specified attributes in single context, 3N-562
fn_ctx Equivalent_name — construct equivalent name in same context, 3N-583
fn_ext_searchlist_destroy — search for names whose attributes satisfy filter, 3N-540
fn_ext_searchlist_next — search for names whose attributes satisfy filter, 3N-540
FN_ext_searchlist_t — search for names
whose

FNS, continued
attributes satisfy filter, 3N-540
FN_search_control_t — options for attribute search, 3N-612
FN_search_filter_t — filter expression for attribute search, 3N-615
fn_searchlist_destroy — terminate search for atomic name with specified attributes in single context, 3N-562
fn_searchlist_next — search for next atomic name with specified attributes in single context, 3N-562
FN_searchlist_t — search for atomic name with specified attributes in single context, 3N-562
fopen — open a stream, 3S-631, 3B-629
form library, See also curses library
formatted input conversion — wscanf, 3C-1805
formatted output conversion
— fprintf, 3B-1171
— printf, 3B-1171
— sprintf, 3B-1171
— vsprintf, 3B-1171
forms — character based forms package, 3X-634
forms field attributes, set and get
— field_buffer, 3X-644
— field_status, 3X-644
— form_field_buffer, 3X-644
— set_field_buffer, 3X-644
— set_field_status, 3X-644
— set_max_field, 3X-644
forms field characteristics
— dynamic_field_info, 3X-645
— field_info, 3X-645
— form_field_info, 3X-645
forms field data type validation
— field_arg, 3X-653
— field_type, 3X-653
— form_field_validation, 3X-653
— set_field_type, 3X-653
forms field option routines
— field_opts, 3X-648
forms field option routines, continued
— field_opts_off, 3X-648
— field_opts_on, 3X-648
— form_field_opts, 3X-648
— set_field_opts, 3X-648
forms field, off-screen data ahead or behind
— dataAhead, 3X-638
— dataBehind, 3X-638
— form_data, 3X-638
forms fields, create and destroy
— dup_field, 3X-647
— form_field_new, 3X-647
— free_field, 3X-647
— link_field, 3X-647
— new_field, 3X-647
forms fieldtype routines
— form_fieldtype, 3X-650
— free_fieldtype, 3X-650
— link_fieldtype, 3X-650
— new_fieldtype, 3X-650
— set_fieldtype_arg, 3X-650
— set_fieldtype_choice, 3X-650
forms option routines
— form_opts, 3X-658
— form_opts_off, 3X-658
— form_opts_on, 3X-658
— set_form_opts, 3X-658
forms pagination
— form_new_page, 3X-657
— new_page, 3X-657
— set_new_page, 3X-657
forms window and subwindow association routines
— form_sub, 3X-662
— form_win, 3X-662
— scale_form, 3X-662
— set_form_sub, 3X-662
— set_form_win, 3X-662
forms window cursor, position
— form_cursor, 3X-637
— pos_form_cursor, 3X-637
forms, application-specific routines
— field_init, 3X-654
— field_term, 3X-654
— form_hook, 3X-654
— form_init, 3X-654
Index—30
forms, application-specific routines, continued
  — form_term, 3X-654
  — set_field_init, 3X-654
  — set_field_term, 3X-654
  — set_form_init, 3X-654
  — set_form_term, 3X-654
forms, associate application data
  — field_userptr, 3X-652
  — form_field_userptr, 3X-652
  — form_userptr, 3X-661
  — set_field_userptr, 3X-652
  — set_form_userptr, 3X-661
forms, command processor
  — form_driver, 3X-639
forms, connect fields
  — field_count, 3X-642
  — form_field, 3X-642
  — form_fields, 3X-642
  — move_field, 3X-642
  — set_form_fields, 3X-642
forms, create and destroy
  — form_new, 3X-656
  — free_form, 3X-656
  — new_form, 3X-656
forms, format general appearance
  — form_field_just, 3X-646
forms, format general display attributes
  — field_back, 3X-643
  — field_fore, 3X-643
  — field_pad, 3X-643
  — form_field_attributes, 3X-643
  — set_field_back, 3X-643
  — set_field_fore, 3X-643
  — set_field_pad, 3X-643
forms, set current page and field
  — current_field, 3X-659
  — field_index, 3X-659
  — form_page, 3X-659
  — set_current_field, 3X-659
  — set_form_page, 3X-659
forms, write/erase from associated subwindows
  — form_post, 3X-660
  — post_form, 3X-660
forms, write/erase from associated subwindows, continued
  — unpost_form, 3X-660
fpgetmask — IEEE floating-point environment control, 3C-663
fpgetround — IEEE floating-point environment control, 3C-663
fpgetsticky — IEEE floating-point environment control, 3C-663
fprintf — formatted output conversion, 3B-1171, 3S-1175
fpsetmask — IEEE floating-point environment control, 3C-663
fpsetround — IEEE floating-point environment control, 3C-663
fpsetsticky — IEEE floating-point environment control, 3C-663
fputwc — put wide-character code on a stream, 3S-665
fputws — put wide character string on a stream, 3S-667
free — memory allocator, 3X-179
free space pointed to by terminal
  — del_curterm, 3XC-358
  — restartterm, 3XC-358
  — set_curterm, 3XC-358
  — settterm, 3XC-358
  — setupterm, 3XC-358
freopen — open a stream, 3S-670, 3B-629
frexp — extract mantissa and exponent from double precision number, 3C-672
fscanf — convert formatted input, 3S-1360
fseek — reposition a file-position indicator in a stream, 3S-673
fseeko — reposition a file-position indicator in a stream, 3S-673
fsetpos — reposition a file pointer in a stream, 3S-676
fsync — synchronize a file’s in-memory state with that on the physical medium, 3C-677
ftell — return a file offset in a stream, 3S-678
ftello — return a file offset in a stream, 3S-678
ftime — get date and time, 3C-679
ftruncate — set a file to a specified length, 3C-1715
ftw — walk a file tree, 3C-682
func_to_decimal — decimal record from character function, 3-1499
functions to manage lockfile(s) for user’s mailbox
— maillock, 3X-943
— mailunlock, 3X-943
— touchlock, 3X-943
funlockfile — acquire and release stream lock, 3S-528

g

gamma — log gamma function, 3M-909
gamma_r — log gamma function, 3M-909
gconvert — convert number to ASCII, 3-441
gcvt — convert number to ASCII, 3-441
general terminal interface
— termios, 3-1595
generate path name for controlling terminal
— ctermid, 3S-238
— ctermid_r, 3S-238
generate path names matching a pattern
— glob, 3C-806
— globfree, 3C-806
generic transport name-to-address translation
— netdir, 3N-1045
— netdir_free, 3N-1045
— netdir_getbyaddr, 3N-1045
— netdir_getbyname, 3N-1045
— netdir_mergeaddr, 3N-1045
— netdir_options, 3N-1045
— netdir_perxor, 3N-1045
— netdir_sperxor, 3N-1045
— taddr2uaddr, 3N-1045
— uaddr2taddr, 3N-1045
get (or push back) wchar_t characters from curses terminal keyboard
— curs_getwch, 3X-297
— getwch, 3X-297
— mvgetwch, 3X-297
— mvwgetwch, 3X-297
get (or push back) wchar_t characters from curses terminal keyboard, continued
— ungetwch, 3X-297
— wgetwch, 3X-297
get a multibyte character string from terminal
— getnstr, 3XC-738
— getstr, 3XC-738
— mvgetnstr, 3XC-738
— mvgetstr, 3XC-738
— mvwgetnstr, 3XC-738
— mvwgetstr, 3XC-738
— wgetnstr, 3XC-738
— wgetstr, 3XC-738
get a single-byte character from terminal
— getch, 3XC-697
— mvgetch, 3XC-697
— mvwgetch, 3XC-697
— wgetch, 3XC-697
get a string of wchar_t characters (and attributes)
from a curses window — curs_inwchstr, 3X-317
inwchnstr, 3X-317
inwchstr, 3X-317
mvinwchnstr, 3X-317
mvinwchstr, 3X-317
mvwvinwchnstr, 3X-317
mvwvinwchstr, 3X-317
wvinwchnstr, 3X-317
wvinwchstr, 3X-317
get a string of wchar_t characters from a curses window — curs_inwstr, 3X-318
innwstr, 3X-318
inwstr, 3X-318
mvinwstr, 3X-318
mvinwstr, 3X-318
mvwvinwstr, 3X-318
mvwvinwstr, 3X-318
winnwstr, 3X-318
winwstr, 3X-318
get a synchronization object handle from a synchronization object’s address —
td_ta_map_addr2sync, 3T-1573
get a thread’s thread-specific data for libthread_db library of interfaces — td_thr_tsd, 3T-1590

Index–32
get a wchar_t character and its attributes from a
curses window — curs_inwch, 3X-316
    inwch, 3X-316
    mvwinwch, 3X-316
    winwch, 3X-316
get a wide character from terminal
    — get_wch, 3X-799
    — mvget_wch, 3X-799
    — mvwget_wch, 3X-799
    — wget_wch, 3X-799
get a wide character string (with rendition) from a
cchar_t — getcchar, 3X-696
get a wide character string from terminal
    — get_wstr, 3X-739
    — getn_wstr, 3X-739
    — mvget_wstr, 3X-739
    — mvwget_wstr, 3X-739
    — mvwget_wstr, 3X-739
    — wget_wstr, 3X-739
    — wgetn_wstr, 3X-739
get a wide-character code from a stream —
    fgetwc, 3S-521
get address of symbol in shared object —
    dlsym, 3X-395
get and set media attributes
    — media_getattr, 3X-966
    — media_setattr, 3X-966
get configurable variables — confstr, 3C-223
get current file position information —
    fgetpos, 3S-520
get cursor or window coordinates
    — getbegyx, 3X-693
    — getmaxyx, 3X-693
    — getparyx, 3X-693
    — getyx, 3X-693
get diagnostic information —
    derror, 3X-388
get error message string —
    t_strerror, 3N-1732
get foreground process group ID —
    tcgetpgrp, 3-1548
get input baud rate
    — cfgetispeed, 3-203
get list of attribute identifiers —
    fn_attr_get_ids, 3N-548
get name of current host —
    gethostname, 3XN-723
get number of bytes in a character —
    mblen, 3C-959
get or set process scheduling priority
    — getpriority, 3C-747
    — setpriority, 3C-747
get output baud rate
    — cfgetospeed, 3-203
get process group ID for session leader for controlling terminal —
    tcgetsid, 3-1549
get service entry —
    getservbyname, 3N-765
    endservent, 3N-765
    getservbyname_r, 3N-765
    getservbyport, 3N-765
    getservbyport_r, 3N-765
    getservent, 3N-765
    getservent_r, 3N-765
    setservent, 3N-765
get the name of the peer socket —
    getpeername, 3XN-746
get the parameters associated with the terminal —
    tcgetattr, 3-1547
get the socket name —
    getsockname, 3XN-770
get the socket options —
    getssockopt, 3XN-774
get the trace attributes from a tnfctl handle —
    tnfctl_trace_attrs_get, 3X-1661
get thread information in libthread_db library of interfaces —
    td_thr_get_info, 3T-1583
get wchar_t character strings from curses terminal
    keyboard — curs_getwstr, 3X-301
    getnwsr, 3X-301
    getwsr, 3X-301
    mvgetnwsr, 3X-301
    mvwgetnwsr, 3X-301
    wgetnwsr, 3X-301
    wgetwstr, 3X-301
get wide character from a stream —
    getwc, 3S-798
get wide character from stdin stream —
    getwchar, 3S-801

Index—33
get_wch — get a wide character from terminal, 3XC-799
get_wstr — get a wide character string from terminal, 3XC-739
getacdir — get audit control file information, 3-684
getacflg — get audit control file information, 3-684
getacinfo — get audit control file information, 3-684
getacmin — get audit control file information, 3-684
getacna — get audit control file information, 3-684
getauclassent — get audit_class database entry, 3-686
getauclassent_r — get audit_class database entry, 3-686
getauclassnam — get audit_class database entry, 3-686
getauclassnam_r — get audit_class database entry, 3-686
getauditflags() — generate process audit state, 3-711
getauditflagsbin() — convert audit flag specifications, 3-688
getauditflagschar() — convert audit flag specifications, 3-688
getauevent — get audit_event database entry, 3-689
getauevent_r — get audit_event database entry, 3-689
getauevnam — get audit_event database entry, 3-689
getauevnam_r — get audit_event database entry, 3-689
getauevnonam — get audit_event database entry, 3-689
getauevnum — get audit_event database entry, 3-689
getauevnum_r — get audit_event database entry, 3-689
getauuserent — get audit_user database entry, 3-691
getauuserent_r — get audit_user database entry, 3-691
getauusernam — get audit_user database entry, 3-691
getauusernam_r — get audit_user database entry, 3-691
getbegyx — get cursor or window coordinates, 3XC-693
getbkgrnd — set or get the background character (and rendition) of window using a complex character, 3XC-173
getcchar — get a wide character string (with rendition) from a cchar_t, 3XC-696
getch — get a single-byte character from terminal, 3XC-697
getcwd — get pathname of current working directory, 3C-701
getdate — convert user format date and time, 3C-703
getenv — return value for environment name, 3C-709
getexcename — return pathname of executable, 3C-710
getgrent — get group entry from database, 3C-712
getgrent_r — get group entry from database, 3C-712
getgrgid — get group entry from database, 3C-712
getgrgid_r — get group entry from database, 3C-712
getgrnam — get group entry from database, 3C-712
getgrnam_r — get group entry from database, 3C-712
gethostbyaddr — network host database functions, 3XN-490
gethostbyname — network host database func-
ations, 3XN-490
gethostent — network host database functions, 3XN-490
gethostid — get unique identifier of current host, 3C-721
gethostname — get name of current host, 3C-722, 3XN-723
gethrtime — get high resolution real time, 3C-724
gethrvtime — get high resolution virtual time, 3C-724
getlogin — get login name, 3C-725
getlogin_r — get login name, 3C-725
getmaxyx — get cursor or window coordinates, 3XC-693
getmntany — get mnttab file information, 3C-727
getmntent — get mnttab file information, 3C-727
getn_wstr — get a wide character string from terminal, 3XC-739
getnetbyaddr — network database functions, 3XN-492
getnetbyname — network database functions, 3XN-492
getnetent — network database functions, 3XN-492
getnstr — get a multibyte character string from terminal, 3XC-738
getnwstr — get wchar_t character strings from curses terminal keyboard, 3X-301
g getopt — get option letter from argument vector, 3C-740
getpagesize — get system page size, 3C-743
getparyx — get cursor or window coordinates, 3XC-693
getpass — read a string of characters without echo, 3C-744
getpassphrase — read a string of characters without echo, 3C-744
getpeername — get name of peer connected to socket, 3N-745, 3XN-746
getpriority — get or set process scheduling priority, 3C-747
getprotobynumber — network protocol database functions, 3XN-494
getprotoent — network protocol database functions, 3XN-494
getpublickey — retrieve public or secret key, 3N-752
getpw — get passwd entry from UID, 3C-753
getpwnent — get password entry from user database, 3C-754
getpwnent_r — get password entry from user database, 3C-754
getpwnam — get password entry from user database, 3C-754
getpwnam_r — get password entry from user database, 3C-754
getpwuid — get password entry from user database, 3C-754
getpwuid_r — get password entry from user database, 3C-754
getrusage — get information about resource utilization, 3C-761
get the total number of threads in a process for libthread_db — td_ta_get_nthreads, 3T-1572
getsecretkey — retrieve public or secret key, 3N-752
getservbyname — get service entry, 3N-765, 3XN-496
getservbyname_r — get service entry, 3N-765
getservbyport — get service entry, 3N-765, 3XN-496
getservbyport_r — get service entry, 3N-765
getservent — get service entry, 3N-765, 3XN-496
getservent_r — get service entry, 3N-765
getservbyname — get the socket name, 3XN-770
getssockopt — get the socket options, 3XN-774
gets spent — get shadow password database entry, 3C-776
gets spent_r — get shadow password database entry (reentrant), 3C-776
gets pnam — get shadow password database entry,
getspnam_r — get shadow password database entry (reentrant), 3C-776
getstr — get a multibYTE character string from terminal, 3XC-738
getsubopt — parse suboptions from a string, 3C-780
gettext — message handling function, 3C-783
gettimeofday — get date and time, 3C-787, 3B-786
gettext — retrieve a text string, 3C-788
getusershell() — get legal user shells, 3C-790
getutent — access utmp file entry, 3C-791
getutid — access utmp file entry, 3C-791
getutline — access utmp file entry, 3C-791
getutmp — access utmpx file entry, 3C-793
getutmpx — access utmpx file entry, 3C-793
getutxent — access utmpx file entry, 3C-793
endutxent(), 3C-793
getutmp(), 3C-793
getutmpx(), 3C-793
getutxent(), 3C-793
getutxid(), 3C-793
getutxline(), 3C-793
pututxline(), 3C-793
setutxent(), 3C-793
updutxmp(), 3C-793
updutxmpx(), 3C-793
utmxpname(), 3C-793
getutxid — access utmpx file entry, 3C-793
getutxline — access utmpx file entry, 3C-793
getvfsany — get vfstab file entry, 3C-796
getvfSENT — get vfstab file entry, 3C-796
getvfSFILE — get vfstab file entry, 3C-796
getvfSSPEC — get vfstab file entry, 3C-796
getwc — get wide character from a stream, 3S-798
getwchar — get (or push back) wchar_t characters from curses terminal keyboard, 3X-297
getwchar — get wide character from stdin stream, 3S-801
getwd — get current working directory pathname, 3C-802
getwidth — get codeset information, 3C-803
getwin — read a window from, and write a window to, a file, 3XC-804
getws — convert a string of EUC characters from the stream to Process Code, 3S-805
getwstr — get wchar_t character strings from curses terminal keyboard, 3X-301
getyx — get cursor or window coordinates, 3XC-693
glob — generate path names matching a pattern, 3C-806
global_variables — variables used for X/Open Curses, 3XC-810
globfree — generate path names matching a pattern, 3C-806
gmatch — shell global pattern matching, 3G-811
grantpt — grant access to the slave pseudo-terminal device, 3C-812
graphics interface
— arc, 3-1166
— box, 3-1166
— circle, 3-1166
— closepl, 3-1166
— closevt, 3-1166
— cont, 3-1166
— erase, 3-1166
— label, 3-1166
— line, 3-1166
— linmod, 3-1166
— move, 3-1166
— openpl, 3-1166
— openvt, 3-1166
— plot, 3-1166
— point, 3-1166
— space, 3-1166
group IDs
set terminal foreground process group id — tcsetpgrp, 3C-1560
group IDs, supplementary
initialize — initgroups, 3C-837
groups
— endgrent, 3C-712
— fgetgrent, 3C-712
— fgetgrent_r, 3C-712
Index–36
groups, continued
  — getgrent, 3C-712
  — getgrent_r, 3C-712
  — getgrgid, 3C-712
  — getgrgid_r, 3C-712
  — getgrnam, 3C-712
  — getgrnam_r, 3C-712
  — setgrent, 3C-712

H
halfdelay — enable/disable half-delay mode, 3XC-813
halt system processor
  — reboot, 3C-1269
has_colors — manipulate color information, 3XC-193
has_ic — determine insert/delete character/line capability, 3XC-814
has_il — determine insert/delete character/line capability, 3XC-814
hash-table search routine
  — hsearch, 3C-817
hasmntopt — get mnttab file information, 3C-727
have Volume Management check for media —
  volmgt_check, 3X-1756
hcreate — create hash table, 3C-817
hdestroy — destroy hash table, 3C-817
hline — use single-byte characters (and renditions) to draw lines, 3XC-815
hline_set — use complex characters (and renditions) to draw lines, 3XC-816
host ID
  get unique identifier of current host —
    gethostid, 3C-721
host machines, remote
  return information about users — rusers, rusers, 3N-1352
host name
  get name of current host — gethostname, 3C-722
  set name of current host — sethostname, 3C-722
hsearch — hash-table search routine, 3C-817

htonl — convert values between host and network byte order, 3XN-820
htons — convert values between host and network byte order, 3XN-820
hyperbolic cosine function — cosh, 3M-235
hyperbolic sine function — sinh, 3M-1455
hyperbolic tangent function — tanh, 3M-1539
hypot — Euclidean distance function, 3M-821

I
I/O asynchronous
  read and write operations — aio_read, aio_write, 3R-140
I/O multiplexing, synchronous
  — select, 3C-1380
I/O package
  standard buffered I/O — stdio, 3S-1479
I/O, asynchronous
  cancel request — aio_cancel, 3R-133
  file synchronization — aio_sync, 3R-135
  retrieve error status — aio_error, 3R-143
  retrieve return status — aio_return, 3R-143
  wait for request — aio_suspend, 3R-146
I/O, requests
  list — lio_listio, 3R-921
iconv — code conversion function, 3-822
iconv_close — code conversion deallocation function, 3-824
iconv_open — code conversion allocation function, 3-825
idcok — enable/disable hardware insert-character and delete-character features, 3XC-826
idlok — set terminal output controls, 3XC-207
IEEE arithmetic
  convert floating-point number to string —
    ecvt, 3C-443
IEEE floating-point environment control
  — fpgetmaks, 3C-663
  — fpgetround, 3C-663
  — fpgetsticky, 3C-663
  — fpsetmask, 3C-663
  — fpsetround, 3C-663
  — fpsetsticky, 3C-663
ilogb — returns an unbiased exponent, 3M-827
immedok — call refresh on changes to window, 3XC-828
in_wch — retrieve a complex character (with rendition), 3XC-853
in_wchnstr — retrieve complex character string (with rendition), 3XC-854
in_wchstr — retrieve complex character string (with rendition), 3XC-854
inch — return a single-byte character (with rendition), 3XC-829
inchnstr — retrieve a single-byte character string (with rendition), 3XC-830
inchstr — retrieve a single-byte character string (with rendition), 3XC-830
index — string operations, 3C-832
inet — Internet address manipulation, 3N-833
inet_addr — Internet address manipulation, 3N-833, 3XN-835
inet_lnaof — Internet address manipulation, 3N-833, 3XN-835
inet_makeaddr — Internet address manipulation, 3N-833, 3XN-835
inet_netof — Internet address manipulation, 3N-833, 3XN-835
inet_network — Internet address manipulation, 3N-833, 3XN-835
inet_ntoa — Internet address manipulation, 3N-833, 3XN-835
insert a multicolumn text string — wscrtcpprintf, 3C-1805
insert a character
— insch, 3XC-843
— mvinsch, 3XC-843
— mvwinsch, 3XC-843
— winsch, 3XC-843
insert a complex character
— ins_wch, 3XC-851
— mvins_wch, 3XC-851
— mvwins_wch, 3XC-851
— wins_wch, 3XC-851
insert a line in a window
— insertln, 3XC-845
— winsertln, 3XC-845
insert a multibyte character string
— insnstr, 3XC-846
— insstr, 3XC-846
— mvinsnstr, 3XC-846
— mvinsstr, 3XC-846
— mvwinsnstr, 3XC-846
— mvwinsstr, 3XC-846
— winsnstr, 3XC-846
— winsstr, 3XC-846
insert a wchar_t character before the character under the cursor in a curses window —
curs_inswch, 3X-313
input conversion
convert from wchar_t string — wsscanf, 3C-1805
input/output package
standard buffered I/O — stdio, 3S-1479
ins_nwstr — insert a wide character string, 3XC-848
ins_wch — insert a complex character, 3XC-851
ins_wstr — insert a wide character string, 3XC-848
insch — insert a character, 3XC-843
insdelln — insert/delete lines to/from the window, 3XC-844
innwstr — get a string of wchar_t characters from a curses window, 3X-318, 3XC-841
innstr — retrieve a multibyte character string (without rendition), 3XC-839
inswstr — get a string of wchar_t characters from a curses window, 3X-318, 3XC-841
Index—38
insert a wchar_t character before the character under the cursor in a curses window
  — curs_inswch, continued
    inswch, 3X-313
    mvinswch, 3X-313
    mvswch, 3X-313

insert a wide character string
  — insnwstr, 3XC-848
  — inswstr, 3XC-848
  — mvinsnwstr, 3XC-848
  — mvinswstr, 3XC-848
  — mvswstr, 3XC-848
  — mvwinswstr, 3XC-848
  — mvwinsnwstr, 3XC-848
  — winsnwstr, 3XC-848
  — winswstr, 3XC-848

insert wchar_t string before character under the cursor in a curses window — curs_inswstr,
  3X-314
    insnwstr, 3X-314
    inswstr, 3X-314
    mvinsnwstr, 3X-314
    mvinswstr, 3X-314
    mvwinswstr, 3X-314
    mvwinsnwstr, 3X-314
    winsnwstr, 3X-314
    winswstr, 3X-314

insert/delete lines to/from the window
  — insdelln, 3XC-844
  — winsdelln, 3XC-844

insertln — insert a line in a window, 3XC-845

insnstr — insert a multibyte character string, 3XC-846

insnwstr — insert wchar_t string before character under the cursor in a curses window, 3X-314

intrflush — flush output in tty on interrupt, 3XC-852

introduction and overview of X/Open Curses — curses, 3XC-257

inverse hyperbolic functions
  — acosh, 3M-117
  — asinh, 3M-117
  — atanh, 3M-117

invoke the function associated with a door descriptor — door_call, 3X-422

inwch — get a wchar_t character and its attributes from a curses window, 3X-316

inswstr — insert wchar_t string before character under the cursor in a curses window, 3X-314

interfaces for direct probe and process control for another process
  — tnfctl_continue, 3X-1647
  — tnfctl_exec_open, 3X-1647
  — tnfctl_pid_open, 3X-1647

interfaces in libthread_db that target process memory access
  — ps_pread, 3T-1187
  — ps_pwrite, 3T-1187
  — ps_ptread, 3T-1187
  — ps_ptwrite, 3T-1187

interfaces to query and to change the state of a probe
  — tnfctl_probe_connect, 3X-1655
  — tnfctl_probe_disable, 3X-1655
  — tnfctl_probe_disconnect_all, 3X-1655
  — tnfctl_probe_enable, 3X-1655
  — tnfctl_probe_state_get, 3X-1655
  — tnfctl_probe_trace, 3X-1655
  — tnfctl_probe_untrace, 3X-1655

Internet address manipulation — inet, 3N-833
  — inet_addr, 3XN-835, 3N-833
  — inet_lnaof, 3XN-835, 3N-833
  — inet_makeaddr, 3XN-835, 3N-833
  — inet_netof, 3XN-835, 3N-833
  — inet_network, 3XN-835, 3N-833
  — inet_ntoa, 3XN-835, 3N-833

Interprocess Communication
  create a new FIFO — mkfifo, 3C-997

Index—39
inwchnstr — get a string of wchar_t characters (and attributes) from a curses window, 3X-317
inwchstr — get a string of wchar_t characters (and attributes) from a curses window, 3X-317
inwstr — get a string of wchar_t characters from a curses window, 3X-318, 3XC-841
is_linetouched — control window refresh, 3XC-858
is_wintouched — control window refresh, 3XC-858
isalnum — character handling, 3C-243
isalpha — character handling, 3C-243
isascii — character handling, 3C-243
isattym — test for a terminal device, 3C-856
isdigit — character handling, 3C-243
isencrypt — determine whether a buffer of characters is encrypted, 3G-857
isendwin — restore initial terminal environment, 3XC-498
isenglish — wide-character code classification functions, 3C-863
isgraph — character handling, 3C-243
isideogram — wide-character code classification functions, 3C-863
islower — character handling, 3C-243
isnan — test for NaN, 3M-862
isnumber — wide-character code classification functions, 3C-863
isphonogram — wide-character code classification functions, 3C-863
isprint — character handling, 3C-243
ispunct — character handling, 3C-243
isspace — character handling, 3C-243
isspecial — wide-character code classification functions, 3C-863
isupper — character handling, 3C-243
iswalnum — wide-character code classification functions, 3C-863
iswalpha — wide-character code classification functions, 3C-863
iswascii — wide-character code classification functions, 3C-863
iswcntrl — wide-character code classification functions, 3C-863
iswctype — test character for specified class, 3C-865
iswdigit — wide-character code classification functions, 3C-863
iswgraph — wide-character code classification functions, 3C-863
iswlower — wide-character code classification functions, 3C-863
iswprint — wide-character code classification functions, 3C-863
iswpunct — wide-character code classification functions, 3C-863
iswspace — wide-character code classification functions, 3C-863
iswupper — wide-character code classification functions, 3C-863
iswxdigit — wide-character code classification functions, 3C-863
isxdigit — character handling, 3C-243
iterate over probes
— tnfctl_probe_apply, 3X-1652
— tnfctl_probe_apply_ids, 3X-1652
iterate over the set of locks owned by a thread —
  td_thr_lockowner, 3T-1586
iterator functions on process handles from libthread_db library of interfaces
— td_ta_sync_iter, 3T-1578
— td_ta_thr_iter, 3T-1578
— td_ta_tsd_iter, 3T-1578

J
j0 — Bessel functions of the first kind, 3M-867
j1 — Bessel functions of the first kind, 3M-867
jn — Bessel functions of the first kind, 3M-867

K
Kerberos authentication library
— kerberos, 3N-868
— krb_get_cred, 3N-868
— krb_kntoln, 3N-868
<table>
<thead>
<tr>
<th>Kerberos authentication library, continued</th>
</tr>
</thead>
<tbody>
<tr>
<td>— krb_mk_err, 3N-868</td>
</tr>
<tr>
<td>— krb_mk_req, 3N-868</td>
</tr>
<tr>
<td>— krb_mk_safe, 3N-868</td>
</tr>
<tr>
<td>— krb_rd_err, 3N-868</td>
</tr>
<tr>
<td>— krb_rd_req, 3N-868</td>
</tr>
<tr>
<td>— krb_rd_safe, 3N-868</td>
</tr>
<tr>
<td>— krb_set_key, 3N-868</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Kerberos authentication routines for RPC</th>
</tr>
</thead>
<tbody>
<tr>
<td>— authkerb_getucred, 3N-872</td>
</tr>
<tr>
<td>— authkerb_seccreate, 3N-872</td>
</tr>
<tr>
<td>— kerberos_rpc, 3N-872</td>
</tr>
<tr>
<td>— svc_kerb_reg, 3N-872</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Kerberos ticket cache file name</th>
</tr>
</thead>
<tbody>
<tr>
<td>— krb_set_tkt_string, 3N-883</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Kerberos utility routines</th>
</tr>
</thead>
<tbody>
<tr>
<td>— krb_get_admhst, 3N-878</td>
</tr>
<tr>
<td>— krb_get_krbhst, 3N-878</td>
</tr>
<tr>
<td>— krb_get_lrealm, 3N-878</td>
</tr>
<tr>
<td>— krb_get_phost, 3N-878</td>
</tr>
<tr>
<td>— krb_realmofhost, 3N-878</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>kernel virtual memory functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>— copy data from kernel image or running system</td>
</tr>
<tr>
<td>— kvm_read, kvm_kread,</td>
</tr>
<tr>
<td>kvm_uread, 3K-901</td>
</tr>
<tr>
<td>— copy data to kernel image or running system</td>
</tr>
<tr>
<td>— kvm_write, kvm_kwrite,</td>
</tr>
<tr>
<td>kvm_uwrite, 3K-901</td>
</tr>
<tr>
<td>— get entries from kernel symbol table</td>
</tr>
<tr>
<td>— kvm_nlist, 3K-898</td>
</tr>
<tr>
<td>— get invocation argument for process</td>
</tr>
<tr>
<td>— kvm_getcmd, 3K-894</td>
</tr>
<tr>
<td>— get u-area for process</td>
</tr>
<tr>
<td>— kvm_getu, 3K-894</td>
</tr>
<tr>
<td>— kstat — kernel statistics facility</td>
</tr>
<tr>
<td>— kvm_getu, 3K-894</td>
</tr>
<tr>
<td>— kstat_chain_update — update the kstat</td>
</tr>
<tr>
<td>header chain, 3K-890</td>
</tr>
<tr>
<td>— kstat_close — initialize kernel</td>
</tr>
<tr>
<td>statistics facility, 3K-892</td>
</tr>
<tr>
<td>— kstat_data_lookup — find a kstat by name,</td>
</tr>
<tr>
<td>3K-892</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>kernel virtual memory functions, continued</th>
</tr>
</thead>
<tbody>
<tr>
<td>— kstat_lookup — find a kstat by name,</td>
</tr>
<tr>
<td>3K-891</td>
</tr>
<tr>
<td>— kstat_open — initialize kernel statistics facility, 3K-892</td>
</tr>
<tr>
<td>— kstat_read — read or write kstat data, 3K-893</td>
</tr>
<tr>
<td>— kstat_write — read or write kstat data, 3K-893</td>
</tr>
<tr>
<td>— kvm_close — specify kernel to examine, 3K-899</td>
</tr>
<tr>
<td>— kvm_getproc — read system process structures, 3K-896</td>
</tr>
<tr>
<td>— kvm_getu — get u-area for process, 3K-894</td>
</tr>
<tr>
<td>— kvm_kread — copy data from a kernel image or running system, 3K-901</td>
</tr>
</tbody>
</table>

| key_name — return character string used as key name, 3XC-875 |
| keyname — return character string used as key name, 3XC-875 |
| keypad — enable/disable keypad handling, 3XC-876 |
| killchar — return current ERASE or KILL characters, 3XC-499 |
| killpg — send signal to a process group, 3C-877 |
| killwchar — return current ERASE or KILL characters, 3XC-499 |
| kstat — kernel statistics facility, 3K-884 |
| kstat_chain_update — update the kstat header chain, 3K-890 |
| kstat_close — initialize kernel statistics facility, 3K-892 |
| kstat_data_lookup — find a kstat by name, 3K-891 |
| kstat_lookup — find a kstat by name, 3K-891 |
| kstat_open — initialize kernel statistics facility, 3K-892 |
| kstat_read — read or write kstat data, 3K-893 |
| kstat_write — read or write kstat data, 3K-893 |
| kvm_close — specify kernel to examine, 3K-899 |
| kvm_getproc — read system process structures, 3K-896 |
| kvm_getu — get u-area for process, 3K-894 |
| kvm_kread — copy data from a kernel image or running system, 3K-901 |
kvm_kwrite — copy data to a kernel image or running system, 3K-901
kvm_nextproc — read system process structures, 3K-896
kvm_nlist — get entries from kernel symbol table, 3K-898
kvm_open — specify kernel to examine, 3K-899
kvm_read — copy data from kernel image or running system, 3K-901
kvm_setproc — read system process structures, 3K-896
kvm_uread — copy data from a kernel image or running system, 3K-901
kvm_uwrite — copy data to a kernel image or running system, 3K-901

L
label — graphics interface, 3-1166
labs — return absolute value of long integer, 3C-104
language information — nl_langinfo, 3C-1094
ldexp — load exponent of a floating point number, 3C-904
ldiv — compute quotient and remainder, 3C-377
leaveok — set terminal output controls, 3XC-207
lfmt — display error message in standard format and pass to logging and monitoring services, 3C-905
lgamma — log gamma function, 3M-909
lgamma_r — log gamma function, 3M-909
library for TNF probe control in a process or the kernel — libtnfctl, 3X-917
library of interfaces for monitoring and manipulating threads-related aspects of multithreaded programs — libthread_db, 3T-91
library routines for client side calls, continued
— clnt_perror, 3N-1315
— clnt_sperrno, 3N-1315
— clnt_sperror, 3N-1315
— rpc_broadcast, 3N-1315
— rpc_broadcast_exp, 3N-1315
— rpc_call, 3N-1315
— rpc_clnt_calls, 3N-1315
library routines for dealing with creation and manipulation of CLIENT handles
— clnt_control, 3N-1318
— clnt_create, 3N-1318
— clnt_create_timed, 3N-1318
— clnt_create_vers, 3N-1318
— clnt_create_vers_timed, 3N-1318
— clnt_destroy, 3N-1318
— clnt_dg_create, 3N-1318
— clnt_pcreateerror, 3N-1318
— clnt_raw_create, 3N-1318
— clnt_screateerror, 3N-1318
— clnt_tli_create, 3N-1318
— clnt_tp_create, 3N-1318
— clnt_tp_create_timed, 3N-1318
— clnt_vc_create, 3N-1318
— rpc_clnt_create, 3N-1318
— rpc_createerror, 3N-1318
library routines for RPC servers
— rpc_svc_calls, 3N-1338
— svc_dg_enablecache, 3N-1338
— svc_done, 3N-1338
— svc_exit, 3N-1338
— svc_fdset, 3N-1338
— svc_freeargs, 3N-1338
— svc_getargs, 3N-1338
— svc_getreq_common, 3N-1338
— svc_getreqset, 3N-1338
— svc_getrpcsetter, 3N-1338
— svc_max_pollfd, 3N-1338
— svc_pollfd, 3N-1338
— svc_run, 3N-1338
— svc_sendreply, 3N-1338
libthread_db — library of interfaces for monitoring and manipulating threads-related aspects of multithreaded programs, 3T-912

Index—42
libntfctl — library for TNF probe control in a process or the kernel, 3X-917
line — graphics interface, 3-1166
linear search and update routine
  — lfind, 3C-939
  — lsearch, 3C-939
linmod — graphics interface, 3-1166
lio_listio — list directed I/O, 3R-921
list directed I/O — lio_listio, 3R-921
list the atomic names and references bound in a context
  — fn_bindinglist_destroy, 3N-592
  — fn_bindinglist_next, 3N-592
  — FN_bindinglist_t, 3N-592
  — fn_ctx_list_bindings, 3N-592
list the atomic names bound in a context
  — fn_ctx_list_names, 3N-593
  — fn_namelist_destroy, 3N-593
  — fn_namelist_next, 3N-593
  — FN_namelist_t, 3N-593
listen — listen for connections on a socket, 3N-924, 3XN-925
listen for socket connections and limit the queue of incoming connections — listen, 3XN-925
llabs — return absolute value of long long integer, 3C-104
lldiv — compute quotient and remainder, 3C-377
lltostr — string conversion routines, 3C-1508
load exponent of a floating point number — ldexp, 3C-904
load exponent of a radix-independent floating-point number — scalb, 3M-1357, 3M-1358
locale
  modify and query a program’s locale — setlocale, 3C-1419
localeconv — get numeric formatting information, 3C-926
lock
  apply or remove advisory lock on open file — flock, 3B-526
lock address space
  — mlockall, 3C-1004
lock memory pages, continued
  — mlock, 3C-1002
lockf — allows sections of file to be locked, 3C-930
log — natural logarithm function, 3M-935
log gamma function
  — gamma, 3M-909
  — gamma_r, 3M-909
  — lgamma, 3M-909
  — lgamma_r, 3M-909
log10 — base 10 logarithm function, 3M-933
log1p — compute natural logarithm, 3M-934
logb — radix-independent exponent, 3M-936
login name
  — getlogin, 3C-725
  — getlogin_r, 3C-725
longjmp — non-local goto, 3B-1411, 3C-1414
longname — return full terminal type name, 3X-938
look up the link reference bound to a name — fn_ctx_lookup_link, 3N-597
looks up the symbol in the symbol table of the load object in the target process — ps_pglist_global_lookup, 3T-1188
M
madvise — provide advice to VM system, 3-941
maillock — functions to manage lockfile(s) for user’s mailbox, 3X-943
mailunlock — functions to manage lockfile(s) for user’s mailbox, 3X-943
make modified instructions executable —
  sync_instruction_memory, 3C-1517
makecontext — manipulate user contexts, 3C-945
malloc — memory allocator, 3X-179
manage thread signals for libthread_db
  — td_thr_setsigpending, 3T-1588
  — td_thr_sigsetmask, 3T-1588
Management Interface database administration functions
  — DmiAddComponent, 3X-397
  — DmiAddGroup, 3X-397
  — DmiAddLanguage, 3X-397
  — DmiDeleteComponent, 3X-397

Index—43
Management Interface database administration functions, continued
   — DmiDeleteGroup, 3X-397
   — DmiDeleteLanguage, 3X-397

Management Interface initialization functions
   — DmiGetConfig, 3X-406
   — DmiGetVersion, 3X-406
   — DmiRegister, 3X-406
   — DmiSetConfig, 3X-406
   — DmiUnregister, 3X-406

Management Interface listing functions
   — DmiListAttributes, 3X-409
   — DmiListClassNames, 3X-409
   — DmiListComponents, 3X-409
   — DmiListComponentsByClass, 3X-409
   — DmiListGroup, 3X-409
   — DmiListLanguages, 3X-409

Management Interface operation functions
   — DmiAddRow, 3X-401
   — DmiDeleteRow, 3X-401
   — DmiGetAttribute, 3X-401
   — DmiGetMultiple, 3X-401
   — DmiSetAttribute, 3X-401
   — DmiSetMultiple, 3X-401

manipulate color information
   — can_change_color, 3XC-193
   — color_content, 3XC-193
   — COLOR_PAIR, 3XC-193
   — has_colors, 3XC-193
   — init_color, 3XC-193
   — init_pair, 3XC-193
   — pair_content, 3XC-193
   — PAIR_NUMBER, 3XC-193
   — start_color, 3XC-193

manipulate sets of signals — sigsetops, 3C-1443
   sigaddset, 3C-1443
   sigdelset, 3C-1443
   sigemptyset, 3C-1443
   sigfillset, 3C-1443
   sigismember, 3C-1443

manipulate soft labels
   — slk_attr_off, 3XC-1458
   — slk_attr_on, 3XC-1458
   — slk_attr_set, 3XC-1458
   — slk_attroff, 3XC-1458

manipulate soft labels, continued
   — slk_attron, 3XC-1458
   — slk_attrset, 3XC-1458
   — slk_clear, 3XC-1458
   — slk_color, 3XC-1458
   — slk_init, 3XC-1458
   — slk_label, 3XC-1458
   — slk_noutrrefresh, 3XC-1458
   — slk_refresh, 3XC-1458
   — slk_restore, 3XC-1458
   — slk_set, 3XC-1458
   — slk_touch, 3XC-1458
   — slk_wset, 3XC-1458

map a tnftcl error code to a string —
   tnftcl_strerror, 3X-1660

map area of parent window to subwindow —
   mvderwin, 3XC-1039

match filename or path name — fnmatch, 3C-605

math library exception-handling — matherr, 3M-954
   mathematical functions
   — gamma, 3M-909
   — gamma_r, 3M-909
   — igamma, 3M-909
   — igamma_r, 3M-909

matherr — math library exception-handling, 3M-954

mblen — get number of bytes in a character, 3C-959

mbstowcs — convert a character string to a wide-character string, 3C-960

mbtowc — convert a character to a wide-character code, 3C-961

mctl — memory management control, 3B-962

media_findname — convert a supplied name into an absolute pathname that can be used to access removable media, 3X-964

media_getattr — get and set media attributes, 3X-966

media_setattr — get and set media attributes, 3X-966

memory — memory operations, 3C-968
   optimizing usage of user mapped memory —
   madvise, 3-941

Index—44
memory allocator — bsdmalloc, 3X-179
   — alloca, 3C-947
   — calloc, 3C-947, 3X-949, 3X-952
   — free, 3C-947, 3X-949, 3X-952, 3X-179
   — mallinfo, 3X-949
   — malloc, 3C-947, 3X-949, 3X-952, 3X-179
   — malloc, 3C-947, 3X-949, 3X-952, 3X-179
   — memalign, 3C-947
   — realloc, 3C-947, 3X-949, 3X-952, 3X-179
   — valloc, 3C-947
memory lock or unlock
   calling process — plock, 3C-1165
memory management — mctl, 3B-962
   copy a file into memory — copylist, 3G-230
   get system page size — getpagesize, 3C-743
   lock address space — mlockall, 3C-1004
   lock pages in memory — mlock, 3C-1002
   synchronize memory with physical storage — 
      msync, 3C-1025
   unlock address space — munlockall, 3C-1004
   unlock pages in memory — munlock, 3C-1002
memory object, shared
   open — shm_open, 3R-1425
   remove — shm_unlink, 3R-1428
memory operations
   — memcpy, 3C-968
   — memchr, 3C-968
   — memcmp, 3C-968
   — memcopy, 3C-968
   — memmove, 3C-968
   — memory, 3C-968
   — memset, 3C-968
menu library, See also curses library
menus — character based menus package, 3X-969
menus cursor
   — menu_cursor, 3X-974
   — pos_menu_cursor, 3X-974
menus display attributes
   — menu_attributes, 3X-972
   — menu_back, 3X-972
   — menu_fore, 3X-972
   — menu_grey, 3X-972
   — menu_pad, 3X-972
   — set_menu_back, 3X-972
   — set_menu_fore, 3X-972
menus display attributes, continued
   — set_menu_grey, 3X-972
   — set_menu_pad, 3X-972
menus from associated subwindows, write/erase
   — menu_post, 3X-992
   — post_menu, 3X-992
   — unpost_menu, 3X-992
menus item name and description
   — item_description, 3X-981
   — item_name, 3X-981
   — menu_item_name, 3X-981
menus item options routines
   — item_opts, 3X-983
   — item_opts_off, 3X-983
   — item_opts_on, 3X-983
   — menu_item_opts, 3X-983
   — set_item_opts, 3X-983
menus item values, set and get
   — item_value, 3X-986
   — menu_item_value, 3X-986
   — set_item_value, 3X-986
menus item, visibility
   — item_visible, 3X-987
   — menu_item_visible, 3X-987
menus items, associate application data
   — item_userptr, 3X-985
   — menu_item_userptr, 3X-985
   — set_item_userptr, 3X-985
menus items, connect and disconnect
   — item_count, 3X-984
   — menu_items, 3X-984
   — set_menu_items, 3X-984
menus items, create and destroy
   — free_item, 3X-982
   — menu_item_new, 3X-982
   — new_item, 3X-982
menus items, get and set
   — current_item, 3X-980
   — item_index, 3X-980
   — menu_item_current, 3X-980
   — set_current_item, 3X-980
   — set_top_row, 3X-980
   — top_row, 3X-980
menus mark string routines
   — menu_mark, 3X-988
menus mark string routines, continued
— set_menu_mark, 3X-988

menus options routines
— menu_opts, 3X-990
— menu_opts_off, 3X-990
— menu_opts_on, 3X-990
— set_menu_opts, 3X-990

menus pattern match buffer
— menu_pattern, 3X-991
— set_menu_pattern, 3X-991

menus subsystem, command processor
— menu_driver, 3X-975

menus window and subwindow association routines
— menu_sub, 3X-994
— menu_win, 3X-994
— scale_menu, 3X-994
— set_menu_sub, 3X-994
— set_menu_win, 3X-994

menus, application-specific routines
— item_init, 3X-978
— item_term, 3X-978
— menu_hook, 3X-978
— menu_init, 3X-978
— menu_term, 3X-978
— set_item_init, 3X-978
— set_item_term, 3X-978
— set_menu_init, 3X-978
— set_menu_term, 3X-978

menus, associate application data
— menu_userptr, 3X-993
— set_menu_userptr, 3X-993

menus, create and destroy
— free_menu, 3X-989
— menu_new, 3X-989
— new_menu, 3X-989

menus, rows and columns
— menu_format, 3X-977
— set_menu_format, 3X-977

message catalog
open/catalog — catopen, catclose, 3C-197
read a program message — catgets, 3C-196

message handling functions, continued
— bindtextdomain, 3C-783
— dgettext, 3C-783
— gettext, 3C-783
— textdomain, 3C-783

message queue
close — mq_close, 3R-1012
get attributes — mq_getattr, 3R-1022
notify process (or thread) — mq_notify, 3R-1013
open — mq_open, 3R-1015
receive a message from — mq_receive, 3R-1018
remove — mq_unlink, 3R-1024
send message to — mq_send, 3R-1020
set attributes — mq_setattr, 3R-1022

messages
display a message on stderr or system console
— fmtmsg, 3C-533
print system error messages — perror, 3C-1161
system signal messages — psignal, 3C-1185

meta
— enable/disable meta keys, 3XC-995

mkdirp — create directories in a path, 3G-996
mkfifo — create a new FIFO, 3C-997
mkstemp — make a unique file name, 3C-998
mktemp — make a unique file name, 3C-999

mktime — converts a tm structure to a calendar time, 3C-1000

mnttab file
— getmntany, 3C-727
— getmntent, 3C-727
— hasmntpt, 3C-727
— putmntent, 3C-727

modf — decompose floating-point number, 3C-1006
modff — decompose floating-point number, 3C-1006

modify multiple attributes associated with named object — fn_attr_multi_modify, 3N-561

modify specified attribute associated with name — fn_attr_modify, 3N-553

modify/delete user credentials for an authentication service — pam_setcred, 3-1126
monitor — prepare process execution profile, 3C-1007
move — graphics interface, 3-1166, 3XC-1009
move cursor in window
    — move, 3XC-1009
    — wmove, 3XC-1009
move the cursor — mvcur, 3XC-1038
move window — mvwin, 3XC-1042
movenextch — moving the cursor by character, 3X-278
moveprevch — moving the cursor by character, 3X-278
mp — multiple precision integer arithmetic, 3M-1010
mp_gcd — multiple precision integer arithmetic, 3M-1010
mp_itom — multiple precision integer arithmetic, 3M-1010
mp_madd — multiple precision integer arithmetic, 3M-1010
mp_mcmp — multiple precision integer arithmetic, 3M-1010
mp_mdiv — multiple precision integer arithmetic, 3M-1010
mp_mfree — multiple precision integer arithmetic, 3M-1010
mp_min — multiple precision integer arithmetic, 3M-1010
mp_mout — multiple precision integer arithmetic, 3M-1010
mp_msub — multiple precision integer arithmetic, 3M-1010
mp_mtox — multiple precision integer arithmetic, 3M-1010
mp_mult — multiple precision integer arithmetic, 3M-1010
mp_pow — multiple precision integer arithmetic, 3M-1010
mp_rpow — multiple precision integer arithmetic, 3M-1010
mp_xtom — multiple precision integer arithmetic, 3M-1010
mq_close — close a message queue, 3R-1012
mq_getattr — set/get message queue attributes, 3R-1022
mq_notify — notify process (or thread) that a message is available on a queue, 3R-1013
mq_open — open a message queue, 3R-1015
mq_receive — receive a message from a message queue, 3R-1018
mq_send — send a message to a message queue, 3R-1020
mq_setattr — set/get message queue attributes, 3R-1022
mq_unlink — remove a message queue, 3R-1024
msync — synchronize memory with physical storage, 3C-1025
multiple precision integer arithmetic
    — mp, 3M-1010
    — mp_gcd, 3M-1010
    — mp_itom, 3M-1010
    — mp_madd, 3M-1010
    — mp_mcmp, 3M-1010
    — mp_mdiv, 3M-1010
    — mp_mfree, 3M-1010
    — mp_min, 3M-1010
    — mp_mout, 3M-1010
    — mp_msub, 3M-1010
    — mp_mtox, 3M-1010
    — mp_mult, 3M-1010
    — mp_pow, 3M-1010
    — mp_rpow, 3M-1010
    — mp_xtom, 3M-1010
mutex — mutual exclusion locks, 3T-1027
Destroy, 3T-1031
Dynamically Allocated Mutexes, 3T-1035
Initialize, 3T-1027
Interprocess Locking, 3T-1033
Lock and Unlock, 3T-1030
Multiple Instruction Single Data, 3T-1032
POSIX, 3T-1027
POSIX Initialize, 3T-1028
POSIX/Solaris Locking, 3T-1030
Single Gate, 3T-1031
Solaris, 3T-1027
Solaris Initialize, 3T-1029
mutex_destroy — mutual exclusion locks,
mutex_init — mutual exclusion locks, 3T-1027
mutex_lock — mutual exclusion locks, 3T-1027
mutex_unlock — mutual exclusion locks, 3T-1027
mutex_destroy, 3T-1027
mutex_init, 3T-1027
mutex_lock, 3T-1027
mutex_unlock, 3T-1027
pthread_mutex_destroy, 3T-1027
pthread_mutex_init, 3T-1027
pthread_mutex_lock, 3T-1027
pthread_mutex_trylock, 3T-1027
pthread_mutex_unlock, 3T-1027
mvget_wch — get a wide character from terminal, 3XC-799
mvget_wstr — get a wide character string from terminal, 3X-738
mvgetn_str — get a multibyte character string from terminal, 3X-738
mvgetnwstr — get wchar_t character strings from curses terminal keyboard, 3X-301
mvgetstr — get a multibyte character string from terminal, 3X-738
mvgetwch — get (or push back) wchar_t characters from curses terminal keyboard, 3X-297
mvgetwstr — get wchar_t character strings from curses terminal keyboard, 3X-301
mvhline — use single-byte characters (and renditions) to draw lines, 3X-815
mvhline_set — use complex characters (and renditions) to draw lines, 3X-816
mvget_str — retrieve a complex character string (with rendition), 3X-853
mvget_str — retrieve complex character string (with rendition), 3X-854
mvgetwch — retrieve a single-byte character (with rendition), 3X-829
mvgetchnstr — retrieve a single-byte character string (with rendition), 3X-830

3T-1027

Index—48
mvinchstr — retrieve a single-byte character string (with rendition), 3XC-830
mvinnstr — retrieve a multibyte character string (without rendition), 3XC-839
mvinnwstr — get a string of wchar_t characters from a curses window, 3X-318, 3XC-841
mvins_nwstr — insert a wide character string, 3XC-848
mvins_wch — insert a complex character, 3XC-851
mvins_wstr — insert a wide character string, 3XC-848
mvinsch — insert a character, 3XC-843
mvinsnstr — insert a multibyte character string, 3XC-846
mvinsnswstr — insert wchar_t string before character under the cursor in a curses window, 3X-314
mvinsstr — insert a multibyte character string, 3XC-846
mvinstr — retrieve a multibyte character string (without rendition), 3XC-839
mvinswch — insert a wchar_t character before the character under the cursor in a curses window, 3X-313
mvinswstr — insert wchar_t string before character under the cursor in a curses window, 3X-314
mvinwch — get a wchar_t character and its attributes from a curses window, 3X-316
mvinwchnstr — get a string of wchar_t characters (and attributes) from a curses window, 3X-317
mvinwchstr — get a string of wchar_t characters (and attributes) from a curses window, 3X-317
mvinwstr — get a string of wchar_t characters from a curses window, 3X-318, 3XC-841
mvprintw — write formatted output to window, 3XC-1040
mvscanw — read formatted input from window, 3XC-1041
mvvline — use single-byte characters (and renditions) to draw lines, 3XC-815
mvvline_set — use complex characters (and renditions) to draw lines, 3XC-816
mvwaddr_wch — add a complex character (with rendition) to a window, 3XC-128
mvwaddr_wchnstr — copy a string of complex characters (with renditions) to a window, 3XC-130
mvwadd_wchstr — copy a string of complex characters (with rendition) to a window, 3XC-130
mvwaddch — add a character (with rendition) to a window, 3XC-118
mvwaddchnstr — copy a character string (with renditions) to a window, 3XC-119
mvwaddchstr — copy a character string (with renditions) to a window, 3XC-119
mvwaddnwstr — add a string of wchar_t characters to a curses window and advance cursor, 3X-277, 3XC-123
mvwaddrstr — add a multi-byte character string (without rendition) to a window, 3XC-121
mvwaddwch — add a wchar_t character (with attributes) to a curses window and advance cursor, 3X-272
mvwaddwchnstr — add string of wchar_t characters (and attributes) to a curses window, 3X-275
mvwaddwchstr — add string of wchar_t characters (and attributes) to a curses window, 3X-275
mvwaddwstr — add a string of wchar_t characters to a curses window and advance cursor, 3X-277, 3XC-123
mvwchgat — change the rendition of characters in a window, 3XC-205
mvwdelch — remove a character, 3XC-357
mvwget_wch — get a wide character from terminal, 3XC-799
mvwget_wstr — get a wide character string from terminal, 3XC-739
mvwgetch — get a single-byte character from terminal, 3XC-697
mvwgetn_wstr — get a wide character string from terminal, 3XC-739
mvwgetnstr — get a multibyte character string from terminal, 3XC-738

Index—49
mvwgetnwstr — get wchar_t character strings from curses terminal keyboard, 3X-301
mvwgetstr — get a multibyte character string from terminal, 3XC-738
mvwgetwch — get (or push back) wchar_t characters from curses terminal keyboard, 3X-297
mvwgetwstr — get wchar_t character strings from curses terminal keyboard, 3X-301
mvwhline — use single-byte characters (and renditions) to draw lines, 3XC-815
mvwhline_set — use complex characters (and renditions) to draw lines, 3XC-816
mvwin — move window, 3XC-1042
mvwin_wch — retrieve a complex character (with rendition), 3XC-853
mvwin_wchnstr — retrieve complex character string (with rendition), 3XC-854
mvwin_wchstr — retrieve complex character string (with rendition), 3XC-854
mvwinch — return a single-byte character (with rendition), 3XC-829
mvwinchnstr — retrieve a single-byte character string (with rendition), 3XC-830
mvwinchstr — retrieve a single-byte character string (with rendition), 3XC-830
mvwinnstr — retrieve a multibyte character string (without rendition), 3XC-839
mvwinnwstr — get a string of wchar_t characters from a curses window, 3X-318, 3XC-841
mvwinsnstr — insert a wide character string, 3XC-848
mvwinsnwstr — insert a wide character string, 3XC-848
mvwinsnch — insert a complex character, 3XC-851
mvwinsnchstr — insert a character, 3XC-843
mvwinsnstr — insert a multibyte character string, 3XC-846
mvwinsnswstr — insert wchar_t string before character under the cursor in a curses window, 3X-314
mvwinsstr — insert a multibyte character string, 3XC-846
mvwinstr — retrieve a multibyte character string (without rendition), 3XC-839
mvwinswch — insert a wchar_t character before the character under the cursor in a curses window, 3X-313
mvwinswstr — insert wchar_t string before character under the cursor in a curses window, 3X-314
mvwinwch — get a wchar_t character and its attributes from a curses window, 3X-316
mvwinwnchnstr — get a string of wchar_t characters (and attributes) from a curses window, 3X-317
mvwinwchstr — get a string of wchar_t characters (and attributes) from a curses window, 3X-317
mvwinwstr — get a string of wchar_t characters from a curses window, 3X-318, 3XC-841
mvwprintw — write formatted output to window, 3XC-1040
mvwscanw — read formatted input from window, 3XC-1041
mvwvline — use single-byte characters (and renditions) to draw lines, 3XC-815
mvwvline_set — use complex characters (and renditions) to draw lines, 3XC-816
mwwaddnstr — add a multi-byte character string (without rendition) to a window, 3XC-121

N
named pipe
create a new one — mkfifo, 3C-997
nanosleep — high resolution sleep, 3R-1043
napms — sleep process for a specified length of time, 3XC-1044
natural logarithm function — log, 3M-935
netdir — generic transport name-to-address translation, 3N-1045
netdir_free — generic transport name-to-address translation, 3N-1045
netdir_getbyaddr — generic transport name-to-address translation, 3N-1045

Index—50
netdir_getbyname — generic transport name-to-address translation, 3N-1045
netdir_mergeaddr — generic transport name-to-address translation, 3N-1045
netdir_options — generic transport name-to-address translation, 3N-1045
netdir_perror — generic transport name-to-address translation, 3N-1045
netdir_sperror — generic transport name-to-address translation, 3N-1045
network configuration database entry
   — endnetconfig, 3N-732
   — freenetconfig, 3N-732
   — getnetconfig, 3N-732
   — getnetconfigent, 3N-732
   — nc_perror, 3N-732
   — nc_sperror, 3N-732
   — setnetconfig, 3N-732
network configuration entry corresponding to NET-PATH
   — endnetpath, 3N-736
   — getnetpath, 3N-736
   — setnetpath, 3N-736
network database functions
   — endnetent, 3XN-492
   — getnetbyaddr, 3XN-492
   — getnetbyname, 3XN-492
   — getnetent, 3XN-492
   — getnetent_r, 3XN-492
   — setnetent, 3XN-492
network entry
   — endnetent, 3N-729
   — getnetbyaddr, 3N-729
   — getnetbyaddr_r, 3N-729
   — getnetbyname, 3N-729
   — getnetbyname_r, 3N-729
   — getnetent, 3N-729
   — getnetent_r, 3N-729
network group entry
   — endnetgrent, 3N-734
   — getnetgrent, 3N-734
   — getnetgrent_r, 3N-734
   — innetgr, 3N-734
   — setnetgrent, 3N-734
network host database functions
   — endhostent, 3XN-490
   — gethostbyaddr, 3XN-490
   — gethostbyname, 3XN-490
   — gethostent, 3XN-490
   — sethostent, 3XN-490
network host entry
   — endhostent, 3N-716
   — gethostbyaddr, 3N-716
   — gethostbyaddr_r, 3N-716
   — gethostbyname, 3N-716
   — gethostbyname_r, 3N-716
   — gethostent, 3N-716
   — gethostent_r, 3N-716
   — sethostent, 3N-716
network listener service
   format and send listener service request message — nlsrequest, 3N-1097
   get client’s data passed via the listener —
      nlsgetcall, 3N-1095
   get name of transport provider — nlsprovider, 3N-1096
network protocol database functions
   — endprotoent, 3XN-494
   — getprotobyname, 3XN-494
   — getprotobynumber, 3XN-494
   — getprotoent, 3XN-494
   — setprotoent, 3XN-494
network protocol entry
   — endprotoent, 3N-749
   — getprotobyname, 3N-749
   — getprotobyname_r, 3N-749
   — getprotobynumber, 3N-749
   — getprotobynumber_r, 3N-749
   — getprotoent, 3N-749
   — getprotoent_r, 3N-749
   — setprotoent, 3N-749
network services database functions
   — endservent, 3XN-496
   — getservbyname, 3XN-496
   — getservbyport, 3XN-496
   — getservent, 3XN-496
   — setservent, 3XN-496
newpad — create and display curses pads, 3X-325, 3XC-1049

Index—51
newterm — screen initialization functions, 3XC-838
newwin — create a new window or subwindow, 3XC-364
next representable double-precision floating-point number — nextafter, 3M-1051
nextafter — next representable double-precision floating-point number, 3M-1051
nextkey — data base subroutines, 3B-348
nftw — walk a file tree, 3C-682
nice — change priority of a process, 3B-1052

NIS client interface
— yp_all, 3N-1836
— yp_bind, 3N-1836
— yp_first, 3N-1836
— yp_get_default_domain, 3N-1836
— yp_master, 3N-1836
— yp_match, 3N-1836
— yp_next, 3N-1836
— yp_order, 3N-1836
— yp_unbind, 3N-1836
— ypclnt, 3N-1836
— yperr_string, 3N-1836
— ypprot_err, 3N-1836

NIS+ database functions
— db_add_entry, 3N-1053
— db_checkpoint, 3N-1053
— db_create_table, 3N-1053
— db_destroy_table, 3N-1053
— db_first_entry, 3N-1053
— db_free_result, 3N-1053
— db_initialize, 3N-1053
— db_list_entries, 3N-1053
— db_next_entry, 3N-1053
— db_remove_entry, 3N-1053
— db_reset_next_entry, 3N-1053
— db_standby, 3N-1053
— db_table_exists, 3N-1053
— dbUnload_table, 3N-1053
— nis_db, 3N-1053

NIS+ error messages, continued
nis_sperror, 3N-1058
nis_sperror_r, 3N-1058

NIS+ group manipulation functions
— nis_addmember, 3N-1059
— nis_creategroup, 3N-1059
— nis_destroygroup, 3N-1059
— nis_groups, 3N-1059
— nis_ismember, 3N-1059
— nis_print_group_entry, 3N-1059
— nis_removemember, 3N-1059
— nis_verifygroup, 3N-1059

NIS+ local names
— nis_freenames, 3N-1081
— nis_getnames, 3N-1081
— nis_local_directory, 3N-1062
— nis_local_group, 3N-1062
— nis_local_host, 3N-1062
— nis_local_names, 3N-1062
— nis_local_principal, 3N-1062

NIS+ log administration functions
— nis_checkpoint, 3N-1078
— nis_ping, 3N-1078

NIS+ miscellaneous functions
— nis_freeservelist, 3N-1079
— nis_freetags, 3N-1079
— nis_getservlist, 3N-1079
— niskdir, 3N-1079
— nis_rmdir, 3N-1079
— nis_server, 3N-1079
— nis_servstate, 3N-1079
— nis_stats, 3N-1079

NIS+ namespace functions
— nis_add, 3N-1063
— nis_freeresult, 3N-1063
— nis_lookup, 3N-1063
— nismodify, 3N-1063
— nis_names, 3N-1063
— nis_remove, 3N-1063

NIS+ object formats
— nis_objects, 3N-1069

NIS+ subroutines
— nis_clone_object, 3N-1081
— nis_destroy_object, 3N-1081
— nis_dir_cmp, 3N-1081

Index—52
NIS+ subroutines, continued
- nis_domain_of, 3N-1081
- nis_leaf_of, 3N-1081
- nis_name_of, 3N-1081
- nis_print_object, 3N-1081
- nis_subr, 3N-1081

NIS+ table functions
- nis_add_entry, 3N-1083
- nis_first_entry, 3N-1083
- nis_list, 3N-1083
- nis_modify_entry, 3N-1083
- nis_next_entry, 3N-1083
- nis_remove_entry, 3N-1083
- nis_tables, 3N-1083

NIS, change information
- yp_update, 3N-1841

NIS+ table functions, 3N-1083
- nis_tables — NIS+ table functions, 3N-1083
- nis_tables — NIS+ table functions, 3N-1083
- nis_tables — NIS+ table functions, 3N-1083

nl — enable/disable newline control, 3XC-1091
nl_langinfo — language information, 3C-1094
nlist — get entries from symbol table, 3B-1092

nocbreak — set input mode controls, 3XC-200

nodelay — set blocking or non-blocking read, 3XC-1099

noecho — enable/disable terminal echo, 3XC-438

non-local goto — set jmp, 3B-1411, 3C-1414
- _longjmp, 3C-937, 3B-1411
- _setjmp, 3C-937, 3B-1411
- longjmp, 3B-1411, 3C-1414
- siglongjmp, 3C-1414
- sigsetjmp, 3C-1414

nonl — enable/disable newline control, 3XC-1091

noqiflush — control flush of input and output on interrupt, 3XC-1100

noraw — set input mode controls, 3XC-200

NOTE — annotate source code with info for tools, 3X-1101
- NOTE vs _NOTE, 3X-1101
- NotInfo Argument, 3X-1102

notimeout — set timed blocking or non-blocking read, 3XC-1103

ntohl — convert values between host and network byte order, 3XN-820

ntohs — convert values between host and network byte order, 3XN-820

numbers, convert to strings — econvert, 3-441

O

offsetof — offset of structure member, 3C-1104
open a shared object — dlopen, 3X-391
open a stream — fopen, 3S-631, 3S-670
open directory — opendir, 3C-1105
opendir — open directory, 3C-1105
openlog — control system log, 3-1525
openpl — graphics interface, 3-1166
openvt — graphics interface, 3-1166

operations on a synchronization object in libthread_db
- td_sync_get_info, 3T-1563
- td_sync_setstate, 3T-1563
- td_sync_waiters, 3T-1563

output conversion
- wscanf — convert to wchar_t string, 3C-1804

output conversion, formatted
- fprintf, 3B-1171
- printf, 3B-1171
- sprintf, 3B-1171
- vfprintf, 3B-1171
- vprintf, 3B-1171
- vsprintf, 3B-1171

overlap or overwrite windows
- overlay, 3XC-1107
- overwrite, 3XC-1107

overlay — overlap or overwrite windows, 3XC-1107

overlay or overwrite any portion of window — copywin, 3XC-232

overview of concepts related to POSIX thread cancellation — cancellation, 3T-187

overview of the XFN interface — xfn, 3N-1821

an overview of XFN attribute operations — xfn_attributes, 3N-1822

XFN compound syntax: an overview of XFN model for compound name parsing —

Index—53
overwrite — overlap or overwrite windows, 3X-1107

P
p2close — close pipes to and from a command, 3G-1111
p2open — open pipes to and from a command, 3G-1111
page size, system
get — getpagesize, 3C-743
pair_content — manipulate color information, 3XC-193
PAIR_NUMBER — manipulate color information, 3XC-193
PAM — PAM Service Module APIs, 3-1131, 3-1113
pam — Pluggable Authentication Module
Administrative Interface, 3-1114
Interface Overview, 3-1113
Stacking Multiple Schemes, 3-1114
Stateful Interface, 3-1114
PAM error messages
get string — pam_strerror, 3-1147
PAM routines to maintain module specific state
— pam_get_data, 3-1127
— pam_set_data, 3-1127
PAM Service Module APIs
— pam, 3-1131
pam_acct_mgmt — perform PAM account validation procedures, 3-1116
pam_authenticate — perform authentication within the PAM framework, 3-1117
pam_chauthtok — perform password related functions within the PAM framework, 3-1119
pam_close_session — perform PAM session creation and termination operations, 3-1124
pam_end — authentication transaction routines for PAM, 3-1144
pam_get_data — PAM routines to maintain module specific state, 3-1127
pam_get_item — authentication information routines for PAM, 3-1129
pam_getenv — returns the value for a PAM environment name, 3-1121
pam_getenvlist — returns a list of all the PAM environment variables, 3-1122
pam_open_session — perform PAM session creation and termination operations, 3-1124
pam_putenv — change or add a value to the PAM environment, 3-1125
pam_set_data — PAM routines to maintain module specific state, 3-1127
pam_set_item — authentication information routines for PAM, 3-1129
pam_setcred — modify/delete user credentials for an authentication service, 3-1126
pam_sm — PAM Service Module APIs
Interaction with the User, 3-1132
Interface Overview, 3-1131
Stateful Interface, 3-1131
pam_sm_acct_mgmt — service provider implementation for pam_acct_mgmt, 3-1135
pam_sm_authenticate — service provider implementation for pam_authenticate, 3-1137
pam_sm_chauthtok — service provider implementation for pam_chauthtok, 3-1139
pam_sm_close_session — Service provider implementation for pam_open_session and pam_close_session, 3-1141
pam_sm_open_session — Service provider implementation for pam_open_session and pam_close_session, 3-1141
pam_sm_setcred — service provider implementation for pam_setcred, 3-1142
pam_start — authentication transaction routines for PAM, 3-1144
panel library, See also curses library
panels — character based panels package, 3X-1148
panels deck manipulation routines
— bottom_panel, 3X-1154
— hide_panel, 3X-1153
— panel_hidden, 3X-1153
— panel_show, 3X-1153
— panel_top, 3X-1154

Index—54
panels deck manipulation routines, continued
— show_panel, 3X-1153
— top_panel, 3X-1154

panels deck traversal primitives
— panel_above, 3X-1150
— panel_below, 3X-1150

panels panel, associate application data
— panel_userptr, 3X-1156
— set_panel_userptr, 3X-1156

panels panel, get or set current window
— panel_window, 3X-1157
— replace_panel, 3X-1157

panels virtual screen refresh routine
— panel_update, 3X-1155
— update_panel, 3X-1155

panels window on virtual screen, move
— move_panel, 3X-1151
— panel_move, 3X-1151

panels, create and destroy
— del_panel, 3X-1152
— new_panel, 3X-1152
— panel_new, 3X-1152

password databases
lock the lock file — lckpwdf, 3C-903
unlock the lock file — ulckpwdf, 3C-903

passwords
get passwd entry from UID — getpw, 3C-753
get password entry from a file — fgetpwent, 3C-754
write password file entry — putpwent, 3C-1245

passwords, shadow
get shadow password database entry —
endspent, 3C-776
get shadow password database entry (reen-
trant) — fgetspent_r, 3C-776
write shadow password file entry —
putspent, 3C-1247

path name
return last element — path name, 3C-164

pathfind — search for named file in named direct-
ories, 3G-1158

pclose — initiate pipe to/from a process, 3S-1168
pecho_wchar — add character and refresh win-
dow, 3XC-1160
pechowchar — add character and refresh window,
3X-1160, 3X-325
pechowchar — create and display curses pads,
3X-325
perform authentication within the PAM framework
— pam_authenticate, 3-1117
perform PAM account validation procedures —
pam_acct_mgmt, 3-1116
perform PAM session creation and termination
operations
— pam_close_session, 3-1124
— pam_open_session, 3-1124
perform password related functions within the
PAM framework — pam_chauthtok, 3-1119
perform word expansions
— wordexp, 3C-1801
— wordfree, 3C-1801
perror — print system error messages, 3C-1161
pfmt — display error message in standard format,
3C-1162

pipes
initiate to/from a process — pclose, 3S-1168
open, close to and from a command —
p2open, p2close, 3G-1111
placeholder for future logging functionality —
td_log, 3T-1562
plock — lock or unlock into memory process, text,
or data, 3C-1165
plot — graphics interface, 3-1166
Link Editor, 3-1167
Pluggable Authentication Module
— PAM, 3-1113
pnoutrefresh — create and display curses pads,
3X-325, 3XC-1049
point — graphics interface, 3-1166
pop a thread cancellation cleanup handler —
pthread_cleanup_pop, 3T-1199
popen — initiate pipe to/from a process, 3S-1168
pow — power function, 3M-1170
power function — pow, 3M-1170
preemption control

Index—55
preemption control, continued
   — schedctl_exit, 3X-1365
   — schedctl_init, 3X-1365
   — schedctl_lookup, 3X-1365
   — schedctl_start, 3X-1365
   — schedctl_stop, 3X-1365

prefresh — create and display curses pads, 3X-325, 3XC-1049

print formatted output
   — fprintf, 3S-1175
   — printf, 3S-1175
   — sprintf, 3S-1175
   — vfprintf, 3S-1767
   — vprintf, 3S-1767
   — vsnprintf, 3S-1767
   — vsprintf, 3S-1767

print formatted output conversion, 3B-1171, 3S-1175

printw — write formatted output to window, 3XC-1040

probe insertion interface
   — TNF_DEBUG, 3X-1669
   — TNF_PROBE_0, 3X-1669
   — TNF_PROBE_0_DEBUG, 3X-1669
   — TNF_PROBE_1, 3X-1669
   — TNF_PROBE_1_DEBUG, 3X-1669
   — TNF_PROBE_2, 3X-1669
   — TNF_PROBE_2_DEBUG, 3X-1669
   — TNF_PROBE_3, 3X-1669
   — TNF_PROBE_3_DEBUG, 3X-1669
   — TNF_PROBE_4, 3X-1669
   — TNF_PROBE_4_DEBUG, 3X-1669
   — TNF_PROBE_5, 3X-1669
   — TNF_PROBE_5_DEBUG, 3X-1669

proc_service — process service interfaces, 3T-1182
   SPARC Platform Only, 3T-1181
   x86 Platform Only, 3T-1182

process and LWP control in libthread_db, continued
   — ps_pcontinue, 3T-1189
   — ps_pstop, 3T-1189

Process Code string operations — wstring, 3C-1806
   wscasecmp, 3C-1806
   wscol, 3C-1806
   wsdup, 3C-1806
   wsnccasecmp, 3C-1806

process service interfaces — proc_service, 3T-1182

process statistics
   prepare execution profile — monitor, 3C-1007

processes
   change priority — nice, 3B-1052
   duplicate an open file descriptor — dup2, 3C-436
   generate path name for controlling terminal — ctermid, ctermid_r, 3S-238
   get character-string representation — cuserid, 3S-347
   initiate pipe to/from a process — popen, pclose, 3S-1168
   manipulate user contexts — makecontext, swapcontext, 3C-945
   memory lock or unlock — plock, 3C-1165
   prepare execution profile — monitor, 3C-1007
   report CPU time used — clock, 3C-209
   send signal to a process group — killpg, 3C-877
   send signal to program — raise, 3C-1253
   set terminal foreground process group id — tcsetpgrp, 3C-1560
   suspend execution for interval — sleep, 3B-1456, 3C-1457
   terminate process — exit, 3C-504
   terminate the process abnormally — abort, 3C-103
   wait for process to terminate or stop — WIFEXITED, 3B-1774

profiling utilities
   prepare process execution profile — monitor, 3C-1007

program assertion
   verify — assert, 3C-150

Index—56
program messages
   open/close a message catalog — catopen,
       catclose, 3C-197
   read — catgets, 3C-196
programs
   last locations — end, etext, edata, 3C-489
ps_kill — process and LWP control in
       libthread_db, 3T-1189
ps_lcontinue — process and LWP control in
       libthread_db, 3T-1189
ps_lgetfregs — routines that access the target
   process register in libthread_db, 3T-1186
ps_lgetregs — routines that access the target
   process register in libthread_db, 3T-1186
ps_lgetxregs — routines that access the target
   process register in libthread_db, 3T-1186
ps_lgetxregsize — routines that access the target
   process register in libthread_db, 3T-1186
ps_lrolltoaddr — process and LWP control in
       libthread_db, 3T-1189
ps_lsetfregs — routines that access the target
   process register in libthread_db, 3T-1186
ps_lsetregs — routines that access the target
   process register in libthread_db, 3T-1186
ps_lsetxregs — routines that access the target
   process register in libthread_db, 3T-1186
ps_lstop — process and LWP control in
       libthread_db, 3T-1189
ps_pcontinue — process and LWP control in
       libthread_db, 3T-1189
ps_pdread — interfaces in libthread_db that target
   process memory access, 3T-1187
ps_pdwrite — interfaces in libthread_db that target
   process memory access, 3T-1187
ps_pglobal_lookup — looks up the symbol in
   the symbol table of the load object in the target
   process, 3T-1188
ps_pstop — process and LWP control in
       libthread_db, 3T-1189
ps_ptread — interfaces in libthread_db that target
   process memory access, 3T-1187
ps_ptwrite — interfaces in libthread_db that tar-
       get process memory access, 3T-1187
pseudo-terminal device
   get name of the slave pseudo-terminal device
       — ptsname, 3C-1240
   grant access to the slave pseudo-terminal dev-
       ice — grantpt, 3C-812
pseudorandom number functions
   — initstate, 3C-1256
   — random, 3C-1256
   — setstate, 3C-1256
   — srand, 3C-1256
psiginfo — system signal messages, 3C-1185
psignal — system signal messages, 3B-1184,
       3C-1185
pthread_atfork — register fork handlers,
       3T-1191
pthread_cancel — cancel execution of a thread,
       3T-1198
pthread_cleanup_pop — pop a thread cancella-
       tion cleanup handler, 3T-1199
pthread_cleanup_push — push a thread cancel-
       lation cleanup handler, 3T-1200
pthread_create — thread creation, 3T-1203
   Attributes, 3T-1204
   POSIX, 3T-1203
   Solaris, 3T-1203
pthread_getspecific — thread-specific-data
   functions, 3T-1217
pthread_key_create — thread-specific-data
   functions, 3T-1217
   Create Key, 3T-1217
   POSIX, 3T-1218
   POSIX Delete Key, 3T-1218
   POSIX Get Value, 3T-1218
   Set Value, 3T-1217
   Solaris, 3T-1217
   Solaris Get Value, 3T-1218
pthread_key_delete — thread-specific-data
   functions, 3T-1217
pthread_mutex_destroy — mutual exclusion
   locks, 3T-1027
pthread_mutex_init — mutual exclusion locks,
       3T-1027
Index—57
pthread_mutex_lock — mutual exclusion locks, 3T-1027
pthread_mutex_trylock — mutual exclusion locks, 3T-1027
pthread_mutex_unlock — mutual exclusion locks, 3T-1027
pthread_setcancelstate — enable or disable cancellation, 3T-1227
pthread_setcanceltype — set the cancellation type of a thread, 3T-1229
pthread_setspecific — thread-specific-data functions, 3T-1217
pthread_sigmask — change and/or examine calling thread’s signal mask (POSIX), 3T-1234
pthread_testcancel — create cancellation point in the calling thread., 3T-1239
ptsname — get name of the slave pseudo-terminal device, 3C-1240
publickey — retrieve public or secret key, 3N-752
push a thread cancellation cleanup handler —
    pthread_cleanup_push, 3T-1200
push character back onto the input queue —
    unget_wch, 3XC-1745
    ungetch, 3XC-1745
push wide-character code back into input stream —
    ungetwc, 3S-1250
put wide-character code back into input stream —
    ungetwc, 3S-1250
put wide character on a stream — putwc, 3S-1248
put wide character on stdout stream — putwchar, 3S-1249
put wide character string on a stream — fputws, 3S-667
put wide-character code on a stream — fputwc, 3S-665
putenv — change or add value to environment, 3C-1243
putmntent — get mnttab file information, 3C-727
putp — apply padding information and output string, 3XC-1244
putpwent — write password file entry, 3C-1245
putspent — write shadow password file entry, 3C-1247
pututxline — access utmpx file entry, 3C-793
pututxline — access utmpx file entry, 3C-793
putwc — put wide character on a stream, 3S-1248
putwchar — put wide character on stdout stream, 3S-1249
putwin — read a window from, and write a window to, a file, 3XC-804
putws — convert a string of Process Code characters to EUC characters and put it on a stream, 3S-1250

Q
qefconvert — convert number to ASCII, 3-441
qgconvert — convert number to ASCII, 3-441
qhconvert — convert number to ASCII, 3-441
qiflush — control flush of input and output on interrupt, 3XC-1100
qsort — quick sort, 3C-1251
quadruple_to_decimal — decimal record from quadruple-precision floating, 3-524
queues
    insert/remove element from a queue —
    insque, remque, 3C-850

R
rac_drop() — remote asynchronous calls, 3N-1326
rac_poll() — remote asynchronous calls, 3N-1326
rac_recv() — remote asynchronous calls, 3N-1326
rac_send() — remote asynchronous calls, 3N-1326
radix-independent exponent — logb, 3M-936
raise — send signal to program, 3C-1253
rand — simple random number generator, 3B-1254, 3C-1255
random — pseudorandom number functions, 3C-1256
random number generator
    — drand48, 3C-434
    — erand48, 3C-434
    — jrand48, 3C-434

Index—58
random number generator, continued
— lcong48, 3C-434
— lrand48, 3C-434
— mrand48, 3C-434
— nrand48, 3C-434
— rand, 3B-1254
— seed48, 3C-434
— srand48, 3C-434
random number generator, simple
— rand, 3C-1255
— srand, 3C-1255
raw — set input mode controls, 3XC-200
rcmd — execute command remotely, 3N-1259
re_comp — compile and execute regular expressions, 3C-1270
re_exec — compile and execute regular expressions, 3C-1270
read a directory entry — readdir, 3B-1261
read a string of characters without echo — getpass, 3C-744
— getpassphrase, 3C-744
read a window from, and write a window to, a file
— getwin, 3XC-804
— putwin, 3XC-804
read and write a disk’s VTOC — read_vtoc, 3X-1266
— write_vtoc, 3X-1266
read directory
— readdir, 3C-1263
— readdir_r, 3C-1263
read formatted input from window
— mvscanw, 3XC-1041
— mvwscanw, 3XC-1041
— scanw, 3XC-1041
— vw_scanw, 3XC-1041
— vwscanw, 3XC-1041
— wscanw, 3XC-1041
read or write kstat data
— kstat_read, 3K-893
— kstat_write, 3K-893
read system process structures
— kvm_getproc, 3K-896
— kvm_nextproc, 3K-896
— kvm_setproc, 3K-896
— read_vtoc — read and write a disk’s VTOC, 3X-1266
— readdir — read a directory entry, 3B-1261, 3C-1263
— POSIX, 3C-1263
— readdir_r — read directory, 3C-1263
— reading and writing thread registers in libthread_db
— td_thr_getfpregs, 3T-1581
— td_thr_getgregs, 3T-1581
— td_thr_getxregs, 3T-1581
— td_thr_getxregsize, 3T-1581
— td_thr_setfpregs, 3T-1581
— td_thr_setgregs, 3T-1581
— td_thr_setxregs, 3T-1581
— realloc — memory allocator, 3X-179
— realpath — resolve pathname, 3C-1267
— reboot — reboot system or halt processor, 3C-1269
— receive a message from a connected socket —
— recv, 3XN-1273
— receive a message from a socket —
— recv, 3N-1271, 3XN-1275, 3XN-1278
— recvfrom, 3N-1271
— recvmsg, 3N-1271
— recv — receive a message from a connected socket,
— 3XN-1273, 3N-1271
— recvfrom — receive a message from a socket,
— 3N-1271, 3XN-1275
— recvmsg — receive a message from a socket,
— 3N-1271, 3XN-1275
— redraw screen or portion of screen
— redrawwin, 3XC-1281
— wredrawln, 3XC-1281
— redrawwin — redraw screen or portion of screen,
— 3XC-1281
— refresh — refresh windows and lines, 3XC-433
— refresh windows and lines
— dupupdate, 3XC-433
— refresh, 3XC-433
— wnoutrefresh, 3XC-433
— wrefresh, 3XC-433
— regcmp — compile regular expression, 3C-1282
— regcomp — regular expression matching, 3C-1284
— regerror — regular expression matching, 3C-1284

Index—59
regex — execute regular expression, 3C-1282
regexec — regular expression matching, 3C-1284
regexp — regular expression compile and match routines, 3G-1290
regfree — regular expression matching, 3C-1284
register callbacks for probe creation and destruction
— tnfctl_register_funcs, 3X-1659
register fork handlers — pthread_atfork, 3T-1191
regular expression compile and match routines
— advance, 3G-1290
— compile, 3G-1290
— regexpr, 3G-1290
— step, 3G-1290
regular expression matching
— regcomp, 3C-1284
— regerror, 3C-1284
— regexec, 3C-1284
— regfree, 3C-1284
regular expressions
compile and execute — regcmp, regex, 3C-1282
release removable media device reservation —
volmgt_release, 3X-1759
remainder — remainder function, 3M-1293
remainder function — remainder, 3M-1293
remote command, return stream to
— rcmd, 3N-1259
remote procedure calls, library routines for — rpc, 3N-1303
remote system
return information about users — rusers, rnusers, 3N-1352
write to — rstat, 3N-1351, 3N-1353
remove — remove file, 3C-1294
remove a character
— delch, 3XC-357
— mvdelch, 3XC-357
— mvwdelch, 3XC-357
— wdelch, 3XC-357
remove a line
— deleteln, 3XC-360
— wdeleteln, 3XC-360
remque — remove element from a queue, 3C-850
rename the name of a binding — fn_ctx_rename, 3N-598
reposition a file-position indicator in a stream
— fseek, 3S-673
— fseeko, 3S-673
res_init — resolver routines, 3N-1296
res_mkquery — resolver routines, 3N-1296
res_query — resolver routines, 3N-1296
res_search — resolver routines, 3N-1296
res_send — resolver routines, 3N-1296
reserve removable media device —
volmgt_acquire, 3X-1753
reserve screen line for dedicated purpose — rip-offline, 3XC-1302
reset file position indicator in a stream — rewind, 3S-1298
reset position of directory stream to the beginning of a directory — rewinddir, 3C-1299
reset_prog_mode — save/restore terminal modes, 3XC-355
reset_shell_mode — save/restore terminal modes, 3XC-355
resetty — restore/save terminal modes, 3XC-1295
resolve pathname — realpath, 3C-1267
resolver — resolver routines, 3N-1296
resolver routines — resolver, 3N-1296
— dn_comp, 3N-1296
— dn_expand, 3N-1296
— res_init, 3N-1296
— res_mkquery, 3N-1296
— res_search, 3N-1296
— res_send, 3N-1296
resource utilization
get information — getrusage, 3C-761
restartterm — free space pointed to by terminal, 3XC-358
restore initial terminal environment
— endwin, 3XC-498
— isendwin, 3XC-498
restore/save terminal modes
— resetty, 3X-1295
— savetty, 3X-1295
retrieve a complex character (with rendition)
- in_wch, 3XC-853
- mvin_wch, 3XC-853
- mvwin_wch, 3XC-853
- win_wch, 3XC-853

retrieve a multibyte character string (without rendition)
- innstr, 3XC-839
- instr, 3XC-839
- mvinnstr, 3XC-839
- mvinstr, 3XC-839
- mvwinnstr, 3XC-839
- mvwinstr, 3XC-839
- winnstr, 3XC-839
- winstr, 3XC-839

retrieve a single-byte character string (with rendition)
- inchnstr, 3XC-830
- inchstr, 3XC-830
- mvinchnstr, 3XC-830
- mvinchstr, 3XC-830
- mvwinchnstr, 3XC-830
- mvwinchstr, 3XC-830
- winchnstr, 3XC-830
- winchstr, 3XC-830

retrieve a wide character string (without rendition)
- innwstr, 3XC-841
- inwstr, 3XC-841
- mvinnwstr, 3XC-841
- mvinwstr, 3XC-841
- mvwinnwstr, 3XC-841
- mvwinwstr, 3XC-841
- winnwstr, 3XC-841
- winwstr, 3XC-841

retrieve archive symbol table — elf_getarsym, 3E-470

retrieve complex character string (with rendition)
- in_wchnstr, 3XC-854
- inwchstr, 3XC-854
- mvin_wchnstr, 3XC-854
- mvinwchstr, 3XC-854
- mvwin_wchnstr, 3XC-854
- mvwinwchstr, 3XC-854
- win_wchnstr, 3XC-854
- winwchstr, 3XC-854

retrieve public or secret key — getpublickey, 3N-752
getsecretkey, 3N-752
publickey, 3N-752

return a file offset for a file descriptor — tell, 3C-1592
return a file offset in a stream
- ftell, 3S-678
- ftello, 3S-678

return a handle to the Initial Context —
fn_ctx_handle_from_initial, 3N-588

returns a list of all the PAM environment variables
- pam_getenvlist, 3-1122

return a single-byte character (with rendition)
- inch, 3XC-829
- mvinch, 3XC-829
- mvwinch, 3XC-829
- winch, 3XC-829

return character string used as key name
- key_name, 3XC-875
- keyname, 3XC-875

return credential information associated with the client —
door_cred, 3X-427

return current ERASE or KILL characters
- erasechar, 3XC-499
- erasewchar, 3XC-499
- killchar, 3XC-499
- killwchar, 3XC-499

return from a door invocation — door_return, 3X-429

return full terminal type name — longname, 3XC-938

return information associated with a door descriptor —
door_info, 3X-428

return magnitude of first argument and sign of second argument — copysign, 3M-231

return multiple attributes associated with named object
- fn_attr_multi_get, 3N-557
- fn_multigetlist_destroy, 3N-557
- fn_multigetlist_next, 3N-557
- FN_multigetlist_t, 3N-557

return pathname of executable — getexecname,
return physical memory information — sysmem, 3-1529
asysmem, 3-1529
return specified attribute associated with name —
fn_attr_get, 3N-547
return stream to a remote command — rexec, 3N-1300
return terminal baud rate — baudrate, 3XC-165
return the synchronization handle for the object on
which a thread is blocked —
td_thr_sleepinfo, 3T-1589
returns the value for a PAM environment name —
pam_getenv, 3-1121
return the value of a terminfo capability
— tigetflag, 3XC-1622
— tigetnum, 3XC-1622
— tigetstr, 3XC-1622
— tpam, 3XC-1622
return the value of the environmental variable
TERM — termname, 3XC-1596
return the video attributes supported by the terminal —
termattr, 3XC-1594
return the Volume Management root directory
— volmgt_root, 3X-1761
return values of an attribute
— fn_attr_get_values, 3N-549
— fn_valuelist_destroy, 3N-549
— fn_valuelist_next, 3N-549
— fn_valuelist_t, 3N-549
return whether or not Volume Management is running —
vollmg_running, 3X-1762
returns an unbiased exponent — ilogb, 3M-827
revoke access to a door descriptor —
door_revoke, 3X-430
rewind — reset file position indicator in a stream, 3S-1298
rewinddir — reset position of directory stream to the
beginning of a directory, 3C-1299
reexec — return stream to a remote command, 3N-1300
rindex — string operations, 3C-832
rint — round-to-nearest integral value, 3M-1301
ripoffline — reserve screen line for dedicated purpose, 3XC-1302
rmdirp — remove directories in a path, 3G-996
rnusers — return information about users on remote machines, 3N-1352
round-to-nearest integral value — rint, 3M-1301
routines that access the target process register in libthread_db
— ps_lgetfregs, 3T-1186
— ps_lgetregs, 3T-1186
— ps_lgetxregs, 3T-1186
— ps_lgetxregsize, 3T-1186
— ps_lsetfregs, 3T-1186
— ps_lsetregs, 3T-1186
— ps_lsetxregs, 3T-1186
rpc — library routines for remote procedure calls, 3N-1303
RPC data transmission using XDR routines — xdr, 3N-1303
RPC bind service library routines
— rpc_getmaps, 3N-1311
— rpcb_getaddr, 3N-1311
— rpcb_gettime, 3N-1311
— rpcb_rmtcall, 3N-1311
— rpcb_set, 3N-1311
— rpcb_unset, 3N-1311
— rpcbind, 3N-1311
RPC entry
— endrpcent, 3N-758
— getrpcbyname, 3N-758
— getrpcbyname_r, 3N-758
— getrpcbyname, 3N-758
— getrpcbyname_r, 3N-758
— getrpcbyname, 3N-758
— getrpcbyname, 3N-758
— getrpcbyname, 3N-758
— getrpcbyname, 3N-758
— setrpcent, 3N-758
RPC library routine for manipulating global RPC
attributes for client and server applications
— rpc_control, 3N-1324
RPC library routines for creation and manipulation of server handles
— rpc_svc_create, 3N-1342
Index — 62
<table>
<thead>
<tr>
<th><strong>RPC library routines for creation and manipulation of server handles, continued</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>- svc_create, 3N-1342</td>
<td></td>
</tr>
<tr>
<td>- svc_destroy, 3N-1342</td>
<td></td>
</tr>
<tr>
<td>- svc_dg_create, 3N-1342</td>
<td></td>
</tr>
<tr>
<td>- svc_fd_create, 3N-1342</td>
<td></td>
</tr>
<tr>
<td>- svc_raw_create, 3N-1342</td>
<td></td>
</tr>
<tr>
<td>- svc_tli_create, 3N-1342</td>
<td></td>
</tr>
<tr>
<td>- svc_tp_create, 3N-1342</td>
<td></td>
</tr>
<tr>
<td>- svc_vc_create, 3N-1342</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>RPC library routines for registering servers</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>- rpc_reg, 3N-1347</td>
<td></td>
</tr>
<tr>
<td>- rpc_svc_reg, 3N-1347</td>
<td></td>
</tr>
<tr>
<td>- svc_auth_reg, 3N-1347</td>
<td></td>
</tr>
<tr>
<td>- svc_reg, 3N-1347</td>
<td></td>
</tr>
<tr>
<td>- svc_unreg, 3N-1347</td>
<td></td>
</tr>
<tr>
<td>- xprt_register, 3N-1347</td>
<td></td>
</tr>
<tr>
<td>- xprt_unregister, 3N-1347</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>RPC library routines for server side errors</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>- rpc_svc_err, 3N-1345</td>
<td></td>
</tr>
<tr>
<td>- svcerr_auth, 3N-1345</td>
<td></td>
</tr>
<tr>
<td>- svcerr_decode, 3N-1345</td>
<td></td>
</tr>
<tr>
<td>- svcerr_noproc, 3N-1345</td>
<td></td>
</tr>
<tr>
<td>- svcerr_noprog, 3N-1345</td>
<td></td>
</tr>
<tr>
<td>- svcerr_progvers, 3N-1345</td>
<td></td>
</tr>
<tr>
<td>- svcerr_systemerr, 3N-1345</td>
<td></td>
</tr>
<tr>
<td>- svcerr_weakauth, 3N-1345</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>RPC obsolete library routines</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>- authdes_create, 3N-1329</td>
<td></td>
</tr>
<tr>
<td>- authunix_create_default, 3N-1329</td>
<td></td>
</tr>
<tr>
<td>- callrpc, 3N-1329</td>
<td></td>
</tr>
<tr>
<td>- clnt_broadcast, 3N-1329</td>
<td></td>
</tr>
<tr>
<td>- clntraw_create, 3N-1329</td>
<td></td>
</tr>
<tr>
<td>- clnttcp_create, 3N-1329</td>
<td></td>
</tr>
<tr>
<td>- clntudp_bufcreate, 3N-1329</td>
<td></td>
</tr>
<tr>
<td>- clntudp_create, 3N-1329</td>
<td></td>
</tr>
<tr>
<td>- get_myaddress, 3N-1329</td>
<td></td>
</tr>
<tr>
<td>- getrpcport, 3N-1329</td>
<td></td>
</tr>
<tr>
<td>- pmap_getmaps, 3N-1329</td>
<td></td>
</tr>
<tr>
<td>- pmap_getport, 3N-1329</td>
<td></td>
</tr>
<tr>
<td>- pmap_rmtcall, 3N-1329</td>
<td></td>
</tr>
<tr>
<td>- pmap_set, 3N-1329</td>
<td></td>
</tr>
<tr>
<td>- pmap_unset, 3N-1329</td>
<td></td>
</tr>
<tr>
<td>- registerrpc, 3N-1329</td>
<td></td>
</tr>
<tr>
<td>- rpc_soc, 3N-1329</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>RPC, XDR library routines</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>- rpc_xdr, 3N-1349</td>
<td></td>
</tr>
<tr>
<td>- xdr_accepted_reply, 3N-1349</td>
<td></td>
</tr>
<tr>
<td>- xdr_authsys_parms, 3N-1349</td>
<td></td>
</tr>
<tr>
<td>- xdr_callhdr, 3N-1349</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>RPC obsolete library routines, continued</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>- svc_fds, 3N-1329</td>
<td></td>
</tr>
<tr>
<td>- svc_getcaller, 3N-1329</td>
<td></td>
</tr>
<tr>
<td>- svc_getreq, 3N-1329</td>
<td></td>
</tr>
<tr>
<td>- svc_register, 3N-1329</td>
<td></td>
</tr>
<tr>
<td>- svc_unregister, 3N-1329</td>
<td></td>
</tr>
<tr>
<td>- svcfd_create, 3N-1329</td>
<td></td>
</tr>
<tr>
<td>- svcraw_create, 3N-1329</td>
<td></td>
</tr>
<tr>
<td>- svctcp_create, 3N-1329</td>
<td></td>
</tr>
<tr>
<td>- svcudp_bufcreate, 3N-1329</td>
<td></td>
</tr>
<tr>
<td>- svcudp_create, 3N-1329</td>
<td></td>
</tr>
<tr>
<td>- xdr_authunix_parms, 3N-1329</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>rpc routines</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>rac_drop() — remote asynchronous calls, 3N-1326</td>
<td></td>
</tr>
<tr>
<td>rac_poll() — remote asynchronous calls, 3N-1326</td>
<td></td>
</tr>
<tr>
<td>rac_recv() — remote asynchronous calls, 3N-1326</td>
<td></td>
</tr>
<tr>
<td>rac_send() — remote asynchronous calls, 3N-1326</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>RPC using Kerberos authentication routines</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>- authkerb_getucred, 3N-872</td>
<td></td>
</tr>
<tr>
<td>- authkerb_seccreate, 3N-872</td>
<td></td>
</tr>
<tr>
<td>- kerberos_rpc, 3N-872</td>
<td></td>
</tr>
<tr>
<td>- svc_kerb_reg, 3N-872</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>RPC, secure library routines</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>- authdes_getucred, 3N-1375</td>
<td></td>
</tr>
<tr>
<td>- authdes_seccreate, 3N-1375</td>
<td></td>
</tr>
<tr>
<td>- getnetname, 3N-1375</td>
<td></td>
</tr>
<tr>
<td>- host2netname, 3N-1375</td>
<td></td>
</tr>
<tr>
<td>- key_decryptsession, 3N-1375</td>
<td></td>
</tr>
<tr>
<td>- key_encryptsession, 3N-1375</td>
<td></td>
</tr>
<tr>
<td>- key_gendes, 3N-1375</td>
<td></td>
</tr>
<tr>
<td>- key_secretkey_is_set, 3N-1375</td>
<td></td>
</tr>
<tr>
<td>- key_setsecret, 3N-1375</td>
<td></td>
</tr>
<tr>
<td>- netname2host, 3N-1375</td>
<td></td>
</tr>
<tr>
<td>- netname2user, 3N-1375</td>
<td></td>
</tr>
<tr>
<td>- secure_rpc, 3N-1375</td>
<td></td>
</tr>
<tr>
<td>- user2netname, 3N-1375</td>
<td></td>
</tr>
</tbody>
</table>

Index—63
RPC, XDR library routines, continued
— xdr_callmsg, 3N-1349
— xdr_opaque_auth, 3N-1349
— xdr_rejected_reply, 3N-1349
— xdr_replymsg, 3N-1349
rpc_broadcast — library routines for client side calls, 3N-1315
rpc_broadcast_exp — library routines for client side calls, 3N-1315
rpc_call — library routines for client side calls, 3N-1315
rpc_clnt_auth — library routines for client side remote procedure call authentication, 3N-1313
rpc_clnt_calls — library routines for client side calls, 3N-1315
Routines, 3N-1315
rpc_clnt_create — library routines for dealing with creation and manipulation of CLIENT handles, 3N-1318
Routines, 3N-1318
rpc_createerr — library routines for dealing with creation and manipulation of CLIENT handles, 3N-1318
rpc_svc_calls — library routines for RPC servers, 3N-1338
Routines, 3N-1338
rresvport — get privileged socket, 3N-1259
rstat — get performance data from remote kernel, 3N-1351
ruserok — authenticate user, 3N-1259
rusers — return information about users on remote machines, 3N-1352
xdr_utmpidlearr, 3N-1352
rw_rdlock() — acquire a read lock, 3T-1354
rw_tryrdlock() — acquire a read lock, 3T-1354
rw_trywrlock() — acquire a write lock, 3T-1354
rw_unlock() — unlock a readers/writer lock, 3T-1354
rw_wrlock() — acquire a write lock, 3T-1354
rwall — write to specified remote machines, 3N-1353
rwlock_destroy() — destroy a readers/writer lock, 3T-1354
rwlock_init() — initialize a readers/writer lock, 3T-1354

S
save/restore terminal modes
— def_prog_mode, 3XC-355
— def_shell_mode, 3XC-355
— reset_prog_mode, 3XC-355
— reset_shell_mode, 3XC-355
savetty — restore/save terminal modes, 3XC-1295
scalb — load exponent of a radix-independent floating-point number, 3M-1357
scalbn — load exponent of a radix-independent floating-point number, 3M-1358
scan a directory
— alphasort, 3B-1359
— scandir, 3B-1359
scandir — scan a directory, 3B-1359
scanf — convert formatted input, 3S-1360
scanw — read formatted input from window, 3XC-1041
sched_get_priority_max — get scheduling parameter limits, 3R-1367
sched_get_priority_min — get scheduling parameter limits, 3R-1367
sched_getparam — get scheduling parameters, 3R-1368
sched_getscheduler — get scheduling policy and scheduling parameters, 3R-1370
sched_rr_get_interval — get scheduling parameter limits, 3R-1367
sched_setparam — set scheduling parameters, 3R-1368
sched_setscheduler — set scheduling policy and scheduling parameters, 3R-1370
sched_yield — yield processor, 3R-1372
schedctl_exit — preemption control, 3X-1365
schedctl_init — preemption control, 3X-1365
schedctl_lookup — preemption control, 3X-1365
schedctl_start — preemption control, 3X-1365
schedctl_stop — preemption control, 3X-1365

Index—64
scheduling parameters
  set/get — sched_setparam, sched_getparam, 3R-1368

scheduling parameters list
  — sched_get_priority_max, 3R-1367
  — sched_get_priority_min, 3R-1367
  — sched_rr_get_interval, 3R-1367

scheduling policy and parameters
  set/get — sched_setscheduler, sched_getscheduler, 3R-1370

scheduling priority
  change priority of a process — nice, 3B-1052

scr_dump — write screen contents to/from a file, 3XC-1373

scr_init — write screen contents to/from a file, 3XC-1373

scr_restore — write screen contents to/from a file, 3XC-1373

scr_set — write screen contents to/from a file, 3XC-1373

screen initialization functions
  — initscr, 3XC-838
  — newterm, 3XC-838

scrl — scroll a window, 3XC-1374

scroll — scroll a window, 3XC-1374

scroll a window
  — scrl, 3XC-1374
  — scroll, 3XC-1374
  — wscrl, 3XC-1374

scrollloc — set terminal output controls, 3XC-207

search for atomic name with specified attributes in single context
  — fn_attr_search, 3N-562
  — fn_searchlist_destroy, 3N-562
  — fn_searchlist_next, 3N-562
  — FN_searchlist_t, 3N-562

search for names whose attributes satisfy filter
  — fn_attr_ext_search, 3N-540
  — fn_ext_searchlist_destroy, 3N-540
  — fn_ext_searchlist_next, 3N-540
  — FN_ext_searchlist_t, 3N-540

search functions
  binary search a sorted table — bsearch, 3C-182

search functions, continued
  linear search and update routine — lsearch, lfind, 3C-939

manage hash search tables — hsearch, 3C-817

seconvert — convert number to ASCII, 3-441

secure, RPC
  See RPC, secure, 3N-1375

seekdir — set position of directory stream, 3C-1379

select — synchronous I/O multiplexing, 3C-1380

sem_close — close a named semaphore, 3R-1386

sem_destroy — destroy an unnamed semaphore, 3R-1387

sem_getvalue — get the value of a semaphore, 3R-1388

sem_init — initialize an unnamed semaphore, 3R-1389

sem_open — initialize/open a named semaphore, 3R-1390

sem_post — increment the count of a semaphore, 3R-1392

sem_trywait — acquire or wait for a semaphore, 3R-1394

sem_unlink — remove a named semaphore, 3R-1393

sem_wait — acquire or wait for a semaphore, 3R-1394

sema_destroy() — destroy a semaphore, 3T-1383

sema_init() — initialize a semaphore, 3T-1383

sema_post() — increment a semaphore, 3T-1383

sema_trywait() — decrement a semaphore, 3T-1383

sema_wait() — decrement a semaphore, 3T-1383

semaphore acquire or wait for — sem_wait, sem_trywait, 3R-1394

close a named one — sem_close, 3R-1386

destroy an unnamed one — sem_destroy, 3R-1387

get the value — sem_getvalue, 3R-1388

increment the count — sem_post, 3R-1392

Index—65
semaphore, continued
initialize an unnamed one — sem_init, 3R-1389
initialize/open a named one — sem_open, 3R-1390
remove a named one — sem_unlink, 3R-1393
send — send a message on a socket, 3XN-1398, 3N-1396
send a “break” for a specific duration — tcsendbreak, 3-1556
send a message on a socket — send, 3XN-1398, 3XN-1403
send a message on a socket using a message structure — sendmsg, 3XN-1400
sendmsg — send a message on a socket using a message structure, 3XN-1400, 3N-1396
sendto — send a message on a socket, 3XN-1403, 3N-1396
Service Access Facility library function
— doconfig, 3N-417
Service Provider functions for components
— DmiOriginateEvent, 3X-415
— DmiRegisterCi, 3X-415
— DmiUnRegisterCi, 3X-415
service provider implementation for
pam_acct_mgmt — pam_sm_acct_mgmt, 3-1135
service provider implementation for
pam_authenticate — pam_sm_authenticate, 3-1137
service provider implementation for pam_chauthtok
— pam_sm_chauthtok, 3-1139
Service provider implementation for
pam_open_session and pam_close_session
— pam_sm_close_session, 3-1141
— pam_sm_open_session, 3-1141
service provider implementation for pam_setcred —
pam_sm_setcred, 3-1142
set a char_t type character from a wide character
and rendition — setcchar, 3XC-1410
set and/or get alternate signal stack context —
sigstack, 3C-1446
set and/or get signal stack context — sigstack,
3B-1445
set blocking or non-blocking read — nodelay, 3X-1099
set concurrency level for target process —
td_ta_setconcurrency, 3T-1577
set encoding key — setkey, 3C-1417
set foreground process group ID — tcsetpgid, 3-1559
set input baud rate
— cfsetispeed, 3-204
set input mode controls
— cbreak, 3XC-200
— nobreak, 3XC-200
— noraw, 3XC-200
— raw, 3XC-200
set or get the background character (and rendition)
of window using a complex character
— bkgrnd, 3XC-173
— bkgrndset, 3XC-173
— getbkgrnd, 3XC-173
— wbkgrnd, 3XC-173
— wbkgrndset, 3XC-173
— wgetbkgrnd, 3XC-173
set output baud rate
— cfsetospeed, 3-204
set position of directory stream — seekdir, 3C-1379
set terminal output controls
— clearok, 3XC-207
— idlok, 3XC-207
— leaveok, 3XC-207
— scrollok, 3XC-207
— setscrreg, 3XC-207
— wsetscrreg, 3XC-207
set the background character (and rendition) of window
— bkgd, 3XC-172
— bkgdset, 3XC-172
— wbkgd, 3XC-172
— wbkgdset, 3XC-172
set the cancellation type of a thread —
 pthread_setcanceltype, 3T-1229
set the parameters associated with the terminal —
tcsetattr, 3-1557
Index–66
set the priority of a thread — td_thr_setprio, 3T-1587
set the socket options — setsockopt, 3XN-1421
set timed blocking or non-blocking read
— notimeout, 3XC-1103
— timeout, 3XC-1103
— wtimeout, 3XC-1103
set values of lines and columns — use_env, 3XC-1748
set visibility of cursor — curs_set, 3XC-333
set/clear window attributes
— standend, 3XC-1478
— standout, 3XC-1478
— wstandend, 3XC-1478
— wstandout, 3XC-1478
set_curterm — free space pointed to by terminal, 3XC-358
set_term — switch between terminals, 3XC-1424
setac — get audit control file information, 3-684
setauclass — rewind audit_class database file, 3-686
setauuser — rewind audit_event database file, 3-689
setauuser — get audit_user database entry, 3-691
setcchar — set a cchar_t type character from a wide character and rendition, 3XC-1410
setgrent — get group entry from database, 3C-712
sethostname — set name of current host, 3C-722
setjmp — non-local goto, 3B-1411, 3C-1414
setkey — set encoding key, 3C-1417
setlabel — define the label for pfmt() and lfmt(), 3C-1418
setlocale — modify and query a program’s locale, 3C-1419
setlogmask — control system log, 3-1525
setnetent — network database functions, 3XN-492
setpriority — get or set process scheduling priority, 3C-747
setprotoent — network protocol database functions, 3XN-494
setpwnam — get password entry from user database, 3C-754
setscrreg — set terminal output controls, 3XC-207
setservent — get service entry, 3N-765, 3XN-496
setsockopt — set the socket options, 3XN-1421
setspent — get shadow password database entry, 3C-776
setstate — pseudorandom number functions, 3C-1256
setterm — free space pointed to by terminal, 3XC-358
settimeofday — set date and time, 3C-787
setuptimeofday — set date and time, 3C-787
setutent — access utmp file entry, 3C-791
setuxent — access utmpx file entry, 3C-793
severity levels, applications
build a list for use with fmtmsg — addseverity, 3C-126
sfconvert — convert number to ASCII, 3-441
sgconvert — convert number to ASCII, 3-441
shared memory object
open — shm_open, 3R-1425
remove — shm_unlink, 3R-1428
shared object
close — dclose, 3X-380
get address of symbol — dladdr, 3X-395
get diagnostic information — dlerror, 3X-388
open — dlopen, 3X-391, 3X-378
shell command
issue one — system, 3S-1530
shell global pattern matching — gmatch, 3G-811
shm_open — open a shared memory object, 3R-1425
shm_unlink — remove a shared memory object, 3R-1428

Index—67
shut down socket send and receive operations — shutdown, 3XN-1430
shutdown — shut down part of a full-duplex connection, 3N-1249, 3XN-1430
sig2str — translation between signal name and signal number, 3C-1483
sigaddset — manipulate sets of signals, 3C-1443
sigdelset — manipulate sets of signals, 3C-1443
sigemptyset — manipulate sets of signals, 3C-1443
sigfillset — manipulate sets of signals, 3C-1443
sigfpe() function, 3-1432
sighold — adds sig to the calling process's signal mask, 3C-1438
sigignore — sets the disposition of sig to SIG_IGN, 3C-1438
siginterrupt — allow signals to interrupt functions, 3B-1435
sigismember — manipulate sets of signals, 3C-1443
siglongjmp — non-local goto, 3C-1443
signal — modify signal disposition, 3C-1438, 3B-1436
schedule after interval in microseconds — ualarm, 3C-1742
suspend execution for interval in microseconds — usleep, 3C-1749
wait for queued signals — sigwaitinfo, sigtimedwait, 3R-1452
simplified signal facilities — bsd_signal, 3C-181
signal management — simplified, for application processes — signal, 3C-1438
signal messages — get error message string — strsignal, 3C-1505
signal messages, system — psignal, 3B-1184, 3C-1185
signals, block — sigblock, 3B-1431
— sigmask, 3B-1431
— sigpause, 3B-1431
signals, block, continued — sigsetmask, 3B-1431
signals, software — gsignal, 3C-1477
— ssignal, 3C-1477
significand — significand function, 3M-1440
significand function — significand, 3M-1440
sigpause — removes sig from the calling process's signal mask and suspends the calling process until a signal is received, 3C-1438
sigqueue — queue a signal to a process, 3R-1441
sigrelse — removes sig from the calling process's signal mask, 3C-1438
sigset — modify signal disposition, 3C-1438
sigset jmp — non-local goto, 3C-1414
sigsetops — manipulate sets of signals, 3C-1443
sigstack — set and/or get alternate signal stack context, 3C-1446, 3B-1445
sigtimedwait — wait for queued signals, 3R-1452
sigvec — software signal facilities, 3B-1448
sigwaitinfo — wait for queued signals, 3R-1452
sin — sine function, 3M-1454
sine function — sin, 3M-1454
single_to_decimal — decimal record from single-precision floating, 3-524
sinh — hyperbolic sine function, 3M-1455
sleep — suspend execution for interval, 3B-1456, 3C-1457
— high resolution — nanosleep, 3R-1043
suspend execution for interval in microseconds — usleep, 3C-1749
sleep process for a specified length of time — napms, 3XC-1044
slk_attr_off — manipulate soft labels, 3XC-1548
slk_attr_on — manipulate soft labels, 3XC-1548
slk_attr_set — manipulate soft labels, 3XC-1548
slk_attroff — manipulate soft labels, 3XC-1548
slk_attron — manipulate soft labels, 3XC-1548
slk_attrset — manipulate soft labels, 3XC-1548
slk_clear — manipulate soft labels, 3XC-1548
slk_color — manipulate soft labels, 3XC-1548
slk_init — manipulate soft labels, 3XC-1548
slk_label — manipulate soft labels, 3XC-1458
slk_noutrefresh — manipulate soft labels, 3XC-1458
slk_refresh — manipulate soft labels, 3XC-1458
slk_restore — manipulate soft labels, 3XC-1458
slk_set — manipulate soft labels, 3XC-1458
slk_touch — manipulate soft labels, 3XC-1458
slk_wset — manipulate soft labels, 3XC-1458
socket — create an endpoint for communication, 3N-1460, 3XN-1463
accept a connection — accept, 3N-105
bind a name — bind, 3N-168
get name — getsockopt, 3N-769
get name of connected peer — getpeername, 3N-745
get options — getsockopt, 3N-771
initiate a connection — connect, 3N-224
listen for connections — listen, 3N-924
send message from — send, sendto, sendmsg, 3N-1396
set options — setsockopt, 3N-771
shut down part of a full-duplex connection — shutdown, 3N-1429
socketpair — create a pair of connected sockets, 3N-1465, 3XN-1466
software signals
— gsignal, 3C-1477
— ssignal, 3C-1477
sort
— qsort, 3C-1251
space — graphics interface, 3-1166
specify an alternative door server thread creation function — door_server_create, 3X-431
sprintf — formatted output conversion, 3B-1171
spray — scatter data in order to test the network, 3N-1468
sprintf — print formatted output, 3S-1175
sqrt — square root function, 3M-1470
square root function — sqrt, 3M-1470
srand — reset simple random number generator, 3B-1254
srandom — pseudorandom number functions, 3C-1256
SSAgentIsAlive — Sun Solstice Enterprise Agent registration and communication helper functions, 3X-1471
SSAGetTrapPort — Sun Solstice Enterprise Agent registration and communication helper functions, 3X-1471
SSAoidCmp — Sun Solstice Enterprise Agent OID helper functions, 3X-1474
SSAoidCpy — Sun Solstice Enterprise Agent OID helper functions, 3X-1474
SSAoidDup — Sun Solstice Enterprise Agent OID helper functions, 3X-1474
SSAoidFree — Sun Solstice Enterprise Agent OID helper functions, 3X-1474
SSAoidInit — Sun Solstice Enterprise Agent OID helper functions, 3X-1474
SSAoidNew — Sun Solstice Enterprise Agent OID helper functions, 3X-1474
SSAoidString — Sun Solstice Enterprise Agent OID helper functions, 3X-1474
SSAoidStrToOid — Sun Solstice Enterprise Agent OID helper functions, 3X-1474
SSAoidZero — Sun Solstice Enterprise Agent OID helper functions, 3X-1474
SSARegSubagent — Sun Solstice Enterprise Agent registration and communication helper functions, 3X-1471
SSARegSubtable — Sun Solstice Enterprise Agent registration and communication helper functions, 3X-1471
SSARegSubtree — Sun Solstice Enterprise Agent registration and communication helper functions, 3X-1471
SSASendTrap — Sun Solstice Enterprise Agent registration and communication helper functions, 3X-1471
SSAStringCpy — Sun Solstice Enterprise Agent string helper functions, 3X-1476
SSAStringInit — Sun Solstice Enterprise Agent string helper functions, 3X-1476
SSAStringToChar — Sun Solstice Enterprise Agent string helper functions, 3X-1476

Index—69
SSAStringZero — Sun Solstice Enterprise Agent
string helper functions, 3X-1476
SSASubagentOpen — Sun Solstice Enterprise
Agent registration and communication helper
functions, 3X-1471
sscanf — convert formatted input, 3S-1360
standend — curses character and window attri-
bute control routines, 3X-280, 3XC-1478
standout — curses character and window attri-
bute control routines, 3X-280, 3XC-1478
start_color — manipulate color information,
3XC-193
stdio — standard buffered input/output package,
3S-1479
step — regular expression compile and match rou-
tines, 3G-1290
string collation
— strcoll, 3C-1486
store — data base subroutines, 3B-348
strfind — string manipulations, 3G-1488
str2sig — translation between signal name and
signal number, 3C-1496
strcadd — copy strings, compressing or expand-
ing C language escape codes, 3G-1484
strccasecmp — string operations, 3C-1496
strcat — string operations, 3C-1496
strcpy — string operations, 3C-1496
strccpy — string operations, 3C-1496
strcspn — string operations, 3C-1496
strdup — string operations, 3C-1496
streadd — copy strings, compressing or expand-
ing C language escape codes, 3G-1484
stream
convert a string of EUC characters from the
stream to Process Code — getws,
3S-805
convert a string of EUC characters from the
stream to Process Code — getwfs,
3S-805
stream, continued
convert a string of Process Code characters to
EUC characters and put it on a stream
— putws, 3S-1250
open — fopen, 3B-629
push character back onto input stream —
ungetc, 3S-1744
stream status inquiries
— clearerr, 3S-516
— feof, 3S-516
— ferror, 3S-516
— fileno, 3S-516
stream, assign buffering
— setbuf, 3S-1406
— setvbuf, 3S-1406
stream, get character or word
— fgetc, 3S-694
— getc, 3S-694
— getc_unlocked, 3S-694
— getchar, 3S-694
— getchar_unlocked, 3S-694
— getw, 3S-694
stream, get string
— fgets, 3S-764
— gets, 3S-764
stream, put a string
— fputs, 3S-1246
— puts, 3S-1246
stream, put character or word
— fputc, 3S-1241
— putc, 3S-1241
— putc_unlocked, 3S-1241
— putchar, 3S-1241
— putchar_unlocked, 3S-1241
— putw, 3S-1241
STREAMS
accept a connection on a socket — accept,
3N-105
attach a STREAMS-based file descriptor to an
object in the file system name space —
fattach, 3C-508
bind a name to a socket — bind, 3N-168
buffered binary input/output — fread,
3S-668
create a pair of connected sockets — socket-
pair,
STREAMS, continued
3N-1465
create an endpoint for communication —
socket, 3N-1460
determine whether a buffer of characters is
encrypted — isencrypt, 3G-857
get and set socket options — getsockopt,
setsockopt, 3N-771
get name of peer connected to socket — get-
peername, 3N-745
get socket name — getsockname, 3N-769
initiate a connection on a socket — connect,
3N-224
listen for connections on a socket — listen,
3N-924
read stream up to next delimiter — bgets,
3G-167
return to remote command — rcmd, 3N-1259
send a message from a socket — send,
sendto, sendmsg, 3N-1396
shut down part of a full-duplex connection —
shutdown, 3N-1429
split buffer into fields — bufsplit, 3G-185
test file descriptor for a STREAMS file —
isastream, 3C-855
strepy — copy strings, compressing or expand-
ing C language escape codes, 3G-1484
strfind — string manipulations, 3G-1488
strfmon — convert monetary value to string,
3C-1489
strptime — convert date and time to string,
3C-1493
string — string operations, 3C-1496
string conversion routines
— atoi, 3C-1508
— atol, 3C-1508
— atoll, 3C-1508
— l1tostr, 3C-1508
— strtol, 3C-1508
— strtoll, 3C-1508
— u1tostr, 3C-1508
string encoding function — crypt, 3C-236
string manipulations — strfind, 3G-1488
string manipulations — strfind, continued
— strfind, 3G-1488
— strrspn, 3G-1488
— strtrns, 3G-1488
string operation
get error message string — strerror,
3C-1487
get PAM error message string —
pam_strerror, 3-1147
get TLI error message string — t_strerror,
3N-1732
string operations
bit and byte — bstring, 3C-184
— index, 3C-832
— rindex, 3C-832
— strcasecmp, 3C-1496
— strcmp, 3C-1496
— strchr, 3C-1496
— strcspn, 3C-1496
— strcpyp, 3C-1496
— strcsps, 3C-1496
— strdup, 3C-1496
— string, 3C-1496
— strlen, 3C-1496
— strncasecmp, 3C-1496
— strncat, 3C-1496
— strncmp, 3C-1496
— strnecpy, 3C-1496
— strnspn, 3C-1496
— strnstr, 3C-1496
— strspn, 3C-1496
— strrchr, 3C-1496
— strsrchr, 3C-1496
— strstr, 3C-1496
— strtok, 3C-1496
— strtok, 3C-1496
— strxfrm, 3C-1514
string transformation — strxfrm, 3C-1514
string_to_decimal — decimal record from
character string, 3-1499
strings
copy, compressing or expanding C language
escape codes, 3G-1484
strings, convert from numbers — econvert, 3-441
strlen — string operations, 3C-1496
strncasecmp — string operations, 3C-1496
strncat — string operations, 3C-1496
strncpy — string operations, 3C-1496
strncpy — string operations, 3C-1496
strpbrk — string operations, 3C-1496
strptime — date and time conversion, 3C-1502
strrchr — string operations, 3C-1496
strfind — string manipulations, 3G-1488
strspn — string operations, 3C-1496
strstr — string operations, 3C-1496
strtok — string manipulations, 3G-1488
strtof — convert string to double-precision number, 3C-1506
strtok — string operations, 3C-1496
strtok_r — string operations, 3C-1496
strtol — string conversion routines, 3C-1508
strtol1 — string conversion routines, 3C-1508
strtoull — convert string to unsigned long, 3C-1511
strtoq — code conversion for Process Code and File Code, 3C-1513
strfind — string manipulations, 3G-1488
strxfrm — string transformation, 3C-1514
subpad — create and display curses pads, 3X-325, 3XC-1049
subwin — create a new window or subwindow, 3XC-364

Sun Solstice Enterprise Agent OID helper functions
— SSAoidCmp, 3X-1474
— SSAoidCpy, 3X-1474
— SSAoidDup, 3X-1474
— SSAoidFree, 3X-1474
— SSAoidInit, 3X-1474
— SSAoidNew, 3X-1474
— SSAoidString, 3X-1474
— SSAoidToStrOid, 3X-1474
— SSAoidZero, 3X-1474

Sun Solstice Enterprise Agent registration and communication helper functions
— SSAgentIsAlive, 3X-1471
— SSAGetTrapPort, 3X-1471
— SSARegSubagent, 3X-1471
— SSARegSubtable, 3X-1471
— SSARegSubtree, 3X-1471
— SSASendTrap, 3X-1471
— SSASubagentOpen, 3X-1471

Index-72
switch between terminals — set_term, 3XC-1424
symbol address
get address in shared object — dlsym, 3X-395
symbol table
get entries — nlist, 3B-1092
sync_instruction_memory — make modified instructions executable, 3C-1517
synchronize a file’s data
— fdatasync, 3R-512
synchronize window with its parents or children
— synck, 3XC-1518
— wcursyncup, 3XC-1518
— wsyncdown, 3XC-1518
— wsyncup, 3XC-1518
synchronous I/O multiplexing
— FD_CLR, 3C-1380
— FD_ISSET, 3C-1380
— FD_SET, 3C-1380
— select, 3C-1380
synck — synchronize window with its parents or children, 3XC-1518
sys_siglist — system signal messages list, 3B-1184
syscall — indirect system call, 3B-1519
sysconf — get configurable system variables, 3C-1520
syslog — control system log, 3-1525
sysmem — return physical memory information, 3-1529
system — issue shell command, 3S-1530
system error messages
print — perror, 3C-1161
system log
log message with variable argument list — vsyslog, 3-1769
system signal messages
— psignal, 3C-1185
system variables
get configurable ones — sysconf, 3C-1520

T

t_accept — accept a connection request, 3N-1531
t_alloc — allocate memory for argument structures, 3N-1535
t_bind — bind an address to a transport endpoint, 3N-1540
t_close — close a transport endpoint, 3N-1550
t_connect — establish a connection with another transport user, 3N-1552
t_error — produce error message, 3N-1597
t_free — free allocated memory, 3N-1599
t_getinfo — get protocol-specific service information, 3N-1602
t_getstate — get the current state, 3N-1608
t_listen — listen for a connect indication, 3N-1629
t_look — look at the current event on a transport endpoint, 3N-1632
t_open — establish a transport endpoint, 3N-1677
t_optmgmt — manage options for a transport endpoint, 3N-1681
t_rcv — manage options for a transport endpoint, 3N-1698
t_rcvconnect — receive the confirmation from a connection request, 3N-1701
t_rcvdis — retrieve information from disconnect, 3N-1704
t_rcvrel — acknowledge receipt of an orderly release indication, 3N-1707
t_rcvuadata — receive a data unit, 3N-1709
t_rcvuderr — receive a unit data error indication, 3N-1712
t_snd — send data or expedited data over a connection, 3N-1720
t_snddis — send user-initiated disconnect request, 3N-1724
t_sndrel — initiate an orderly release of connection, 3N-1727
t_sndudata — send a data unit, 3N-1729
t_strerror — get error message string, 3N-1732
t_sync — synchronize transport library, 3N-1734
t_unbind — disable a transport endpoint, 3N-1739

Index—73
taddr2uaddr — generic transport name-to-address translation, 3N-1045
tan — tangent function, 3M-1538
tangent function — tan, 3M-1538
tanh — hyperbolic tangent function, 3M-1539
tcdrain — wait for transmission of output, 3-1544
tcflush — flush non-transmitted output data, non-read input data or both, 3-1546
tcgetattr — get the parameters associated with the terminal, 3-1547
tcgetpgrp — get foreground process group ID, 3-1548
tcgetsid — get process group ID for session leader for controlling terminal, 3-1549
tcsendbreak — send a “break” for a specific duration, 3-1556
tcsetattr — set the parameters associated with the terminal, 3-1557
tcsetpgrp — set foreground process group ID, 3-1559

td_event_addset — thread events in libthread_db, 3T-1568
td_event_delset — thread events in libthread_db, 3T-1568
td_event_emptyset — thread events in libthread_db, 3T-1568
td_event_fillset — thread events in libthread_db, 3T-1568
td_eventisempty — thread events in libthread_db, 3T-1568

td_event_ismember — thread events in libthread_db, 3T-1568
td_init — initialization function for libthread_db library of interfaces, 3T-1561
td_log — placeholder for future logging functionality, 3T-1562
td_sync_get_info — operations on a synchronization object in libthread_db, 3T-1563
td_sync_setstate — operations on a synchronization object in libthread_db, 3T-1563
td_sync_waiters — operations on a synchronization object in libthread_db, 3T-1563
td_ta_delete — allocate and deallocate process handles for libthread_db, 3T-1575
td_ta_enable_stats — collect target process statistics for libthread_db, 3T-1566
td_ta_event_addr — thread events in libthread_db, 3T-1568
Event Set Manipulation Macros, 3T-1570
td_ta_event_getmsg — thread events in libthread_db, 3T-1568
td_ta_get_nthreads — gets the total number of threads in a process for libthread_db, 3T-1572
td_ta_get_ph — allocate and deallocate process handles for libthread_db, 3T-1575
td_ta_get_stats — collect target process statistics for libthread_db, 3T-1566
td_ta_map_addr2sync — get a synchronization object handle from a synchronization object’s address, 3T-1573
td_ta_map_addr2thr — convert a thread id or thread address to a thread handle, 3T-1574
td_ta_map_id2thr — convert a thread id or thread address to a thread handle, 3T-1574
td_ta_new — allocate and deallocate process handles for libthread_db, 3T-1575
td_ta_reset_stats — collect target process statistics for libthread_db, 3T-1566
td_ta_set_event — thread events in libthread_db, 3T-1568

td_ta_setconcurrency — set concurrency level for target process, 3T-1577
td_ta_sync_iter — iterator functions on process handles from libthread_db library of interfaces, 3T-1578
td_ta_thr_iter — iterator functions on process handles from libthread_db library of interfaces, 3T-1578
td_ta_tsd_iter — iterator functions on process handles from libthread_db library of interfaces, 3T-1578

td_thr_clear_event — thread events in
libthread_db, 3T-1568

`td_thr_dbresume` — suspend and resume threads in libthread_db, 3T-1580

`td_thr_dbsuspend` — suspend and resume threads in libthread_db, 3T-1580

`td_thr_event_enable` — thread events in libthread_db, 3T-1568

`td_thr_event_getmsg` — thread events in libthread_db, 3T-1568

`td_thr_get_info` — get thread information in libthread_db library of interfaces, 3T-1583

`td_thr_getfpregs` — reading and writing thread registers in libthread_db, 3T-1581

`td_thr_getgregs` — reading and writing thread registers in libthread_db, 3T-1581

`td_thr_getxregs` — reading and writing thread registers in libthread_db, 3T-1581

`td_thr_getxregsize` — reading and writing thread registers in libthread_db, 3T-1581

`td_thr_lockowner` — iterate over the set of locks owned by a thread, 3T-1586

`td_thr_set_event` — thread events in libthread_db, 3T-1568

`td_thr_setfpregs` — reading and writing thread registers in libthread_db, 3T-1581

`td_thr_setgregs` — reading and writing thread registers in libthread_db, 3T-1581

`td_thr_setprio` — set the priority of a thread, 3T-1587

`td_thr_setsigpending` — manage thread signals for libthread_db, 3T-1588

`td_thr_setxregs` — reading and writing thread registers in libthread_db, 3T-1581

`td_thr_setxregsize` — reading and writing thread registers in libthread_db, 3T-1588

`td_thr_sleepinfo` — return the synchronization handle for the object on which a thread is blocked, 3T-1589

`td_thr_tsd` — get a thread’s thread-specific data for libthread_db library of interfaces, 3T-1590

`td_thr_validate` — test a thread handle for validity, 3T-1591

`tda_ta_clear_event` — thread events in libthread_db, 3T-1568

`tdelete` — manage binary search trees, 3C-1717

`tell` — return a file offset for a file descriptor, 3C-1592

`teldir` — current location of a named directory stream, 3C-1593

`tempnam` — create a name for a temporary file, 3S-1635

`termattr` — return the video attributes supported by the terminal, 3XC-1594

`terminal` — find the slot in the utmp file of the current user
  — `ttyslot`, 3C-1738

`terminal device, slave pseudo` —
  get name — `ptsname`, 3C-1240
  grant access — `grantpt`, 3C-812

`terminal ID` — generate path name for controlling terminal
  — `ctermid`, `ctermid_r`, 3S-238

`terminal line` — establish an outgoing connection — `dial`, 3N-369

`terminals` —
  set terminal foreground process group id — `tcsetpgrp`, 3C-1560

`termios` — general terminal interface, 3-1595

`termname` — return the value of the environmental variable TERM, 3XC-1596

`test a thread handle for validity` —
  `td_thr_validate`, 3T-1591

`test character for specified class` — `iswctype`, 3C-865

`test for a terminal device` — `isatty`, 3C-856

`test for NaN` — `isnan`, 3M-862

`text processing utilities` —
  compile and execute regular expressions — `regex`, `regex`, 3C-1282
  quick sort — `qsort`, 3C-1251

`text string` —
  `gettxt`, 3C-788

Index—75
textdomain — select domain of messages, 3C-783

tfind — manage binary search trees, 3C-1717

tgetent — emulate the termcap database, 3XC-1601

tgetflag — emulate the termcap database, 3XC-1601

tgetnum — emulate the termcap database, 3XC-1601

tgettext — emulate the termcap database, 3XC-1601

tgoto — emulate the termcap database, 3XC-1601

t_continue — continue thread execution, 3T-1620

thr_create — thread creation, 3T-1203

thr_getconcurrency — get thread concurrency level, 3T-1618

thr_getspecific — thread-specific-data functions, 3T-1217

thr_keycreate — thread-specific-data functions, 3T-1217

thr_main — identifies the calling thread as the main thread or not the main thread, 3T-1615

thr_setconcurrency — set thread concurrency level, 3T-1618

thr_setspecific — thread-specific-data functions, 3T-1217

thr_sigsetmask — change and/or examine calling thread’s signal mask, 3T-1234

thr_stksegment — get thread stack bottom and size, 3T-1619

thr_suspend — suspend thread execution, 3T-1620

thr_yield — thread yield to another thread, 3T-1621

thread creation
    — pthread_create, 3T-1203
    — thr_create, 3T-1203

thread events in libthread_db
    — td_event isempty, 3T-1568
    — td_event ismember, 3T-1568
    — td_taevent_addr, 3T-1568
    — td_taevent_getmsg, 3T-1568
    — td_taevent_setevent, 3T-1568
    — td_thr_clear_event, 3T-1568
    — td_thr_event_enable, 3T-1568
    — td_thr_event_getmsg, 3T-1568
    — td_thr_set_event, 3T-1568
    — tda_taclevent, 3T-1568

thread signal mask
    POSIX — pthread_sigmask, 3T-1234
    Solaris — thr_sigsetmask, 3T-1234

thread yield to another thread — thr_yield, 3T-1621

thread-specific-data functions
    — pthread_getspecific, 3T-1217
    — pthread_key_create, 3T-1217
    — pthread_key_delete, 3T-1217
    — pthread_setspecific, 3T-1217
    — thr_getspecific, 3T-1217
    — thr_keycreate, 3T-1217
    — thr_setspecific, 3T-1217

tigetflag — return the value of a terminfo capability, 3XC-1622

tigetnum — return the value of a terminfo capability, 3XC-1622

tgettext — return the value of a terminfo capability, 3XC-1622

time
    computes the difference between two calendar times — difftime, 3C-372

time accounting
    for current process — times, 3B-1628

time and date
    convert to string — asctime, 3C-239
    convert user format date and time — getdate, 3C-703
    get — ftime, 3C-679
    settimeofday, 3C-787

time of day
    get and set — gettimeofday, settimeofday, 3B-786
time, calendar
convert from a tm structure — mktime, 3C-1000

timeout — set timed blocking or non-blocking
read, 3XC-1103
timer_create — create a per-process, per-LWP,
or per-thread timer, 3R-1623
timer_delete — delete a per-process, per-LWP,
or per-thread timer, 3R-1625
timer_getoverrun — high-resolution timer
operations, 3R-1626
timer_gettime — high-resolution timer opera-
tions, 3R-1626
timer_settime — high-resolution timer opera-
tions, 3R-1626
times — get process times, 3B-1628
tmpfile — create a temporary file, 3S-1634
tmpnam — create a name for a temporary file,
3S-1635
TNF_DEBUG — probe insertion interface, 3X-1669
TNF_PROBE — probe insertion interface
arg_name_n, 3X-1671
arg_type_n, 3X-1671
arg_value_n, 3X-1671
detail, 3X-1670
keys, 3X-1670
name, 3X-1670
TNF_PROBE_0 — probe insertion interface, 3X-1669
TNF_PROBE_0_DEBUG — probe insertion interface,
3X-1669
TNF_PROBE_1 — probe insertion interface, 3X-1669
TNF_PROBE_1_DEBUG — probe insertion interface,
3X-1669
TNF_PROBE_2 — probe insertion interface, 3X-1669
TNF_PROBE_2_DEBUG — probe insertion interface,
3X-1669
TNF_PROBE_3 — probe insertion interface, 3X-1669
TNF_PROBE_3_DEBUG — probe insertion interface,
3X-1669
TNF_PROBE_4 — probe insertion interface, 3X-1669
TNF_PROBE_4_DEBUG — probe insertion interface,
3X-1669

Index—77
tnfctl_probe_apply — iterate over probes, 3X-1652

tnfctl_probe_apply_ids — iterate over probes, 3X-1652

tnfctl_probe_connect — interfaces to query and to change the state of a probe, 3X-1655

tnfctl_probe_disconnect_all — interfaces to query and to change the state of a probe, 3X-1655

tnfctl_probe_enable — interfaces to query and to change the state of a probe, 3X-1655

tnfctl_probe_disable — interfaces to query and to change the state of a probe, 3X-1655

tnfctl_probe_state_get — interfaces to query and to change the state of a probe, 3X-1655

tnfctl_probe_trace — interfaces to query and to change the state of a probe, 3X-1655

tnfctl_probe_untrace — interfaces to query and to change the state of a probe, 3X-1655

tnfctl_register_funcs — register callbacks for probe creation and destruction, 3X-1659

tnfctl_strerror — map a tnfctl error code to a string, 3X-1660

tnfctl_trace_attrs_get — get the trace attributes from a tnfctl handle, 3X-1661

tnfctl_trace_state_set — control kernel tracing and process filtering, 3X-1663

toascii — translate integer to a 7-bit ASCII character, 3C-1674

tolower — transliterate upper-case characters to lower-case, 3C-1676

touchline — control window refresh, 3X-1676

touchlock — functions to manage lockfile(s) for user’s mailbox, 3X-943

touchwin — control window refresh, 3X-1678

toupper — transliterate lower-case characters to upper-case, 3C-1689

towctrans — wide-character mapping, 3C-1690

towlower — transliterate upper-case wide-character code to lower-case, 3C-1691

towupper — transliterate lower-case wide-character code to upper-case, 3C-1692

tparm — return the value of a terminfo capability, 3X-1622

tputs — apply padding information and output string, 3X-1244

tracing — overview of routines for tnf tracing

Tracing a Process, 3X-1694

Tracing the Kernel, 3X-1695

translate address to symbolic information — dladdr, 3X-378

translate integer to a 7-bit ASCII character — toascii, 3C-1674

translation between signal name and signal number — str2sig, 3C-1483

sig2str, 3C-1483

transliterate lower-case characters to upper-case — _toupper, 3C-1688, 3C-1689

transliterate lower-case wide-character code to upper-case — towupper, 3C-1692

transliterate upper-case characters to lower-case — _tolower, 3C-1675, 3C-1676

transliterate upper-case wide-character code to lower-case — towlower, 3C-1691

transport functions

accept a connection request — t_accept, 3N-1531

acknowledge receipt of an orderly release indication — t_rcvrel, 3N-1707

allocate memory for argument structures — t_alloc, 3N-1535

bind an address to a transport endpoint — t_bind, 3N-1540

close a transport endpoint — t_close, 3N-1550

disable a transport endpoint — t_unbind, 3N-1739

establish a connection with another transport user — t_connect, 3N-1552

establish a transport endpoint — t_open, 3N-1677

free allocated memory — t_free, 3N-1599

get protocol-specific service information — t_getinfo, 3N-1602

Index−78
transport functions, continued
get the current state — t_getstate, 3N-1608
initiate an orderly release of connection —
t_sndrel, 3N-1727
listen for a connect indication — t_listen,
3N-1629
look at the current event on a transport end-
point — t_look, 3N-1632
manage options for a transport endpoint —
t_optmgmt, 3N-1681
produce error message — t_error, 3N-1597
receive a data unit — t_rcvudata, 3N-1709
receive a unit data error indication —
t_rcvuderr, 3N-1712
receive data or expedited data sent over a con-
nection — t_rcv, 3N-1698
receive the confirmation from a connection
request — t_rcvconnect, 3N-1701
retrieve information from disconnect —
t_rcvdis, 3N-1704
send a data unit — t_sndudata, 3N-1729
send data or expedited data over a connection
— t_snd, 3N-1720
send user-initiated disconnect request —
t_snddis, 3N-1724
synchronize transport library — t_sync,
3N-1734
truncate — set a file to a specified length, 3C-1715
tsearch — manage binary search trees, 3C-1717
ttynamel — find pathname of a terminal, 3C-1736
POSIX, 3C-1736
ttynamel_r — find pathname of a terminal,
3C-1736
ttyslot — find the slot in the utmp file of the
current user, 3C-1738
twalk — manage binary search trees, 3C-1717
typeahead — check for type-ahead characters,
3X-1741

U
uaddr2taddr — generic transport name-to-
address translation, 3N-1045
ualarm — schedule signal after interval in
microseconds, 3C-1742
ulltostr — string conversion routines, 3C-1508
unctrl — convert character to printable form,
3X-1743
unget_wch — push character back onto the input
queue, 3X-1745
ungetch — push character back onto the input
queue, 3X-1745
ungetwc — push wide-character code back into
input stream, 3S-1746
ungetwch — get (or push back) wchar_t characters
from curses terminal keyboard, 3X-297
unlock a pseudo-terminal master/slave pair —
unlockpt, 3C-1747
unlock address space
— munlockall, 3C-1004
unlock memory pages
— munlock, 3C-1002
unlockpt — unlock a pseudo-terminal
master/slave pair, 3C-1747
untouchwin — control window refresh, 3XC-858
updwtmp — access utmpx file entry, 3C-793
updwtmpx — access utmpx file entry, 3C-793
use complex characters (and renditions) to draw
borders
— border_set, 3XC-177
— box_set, 3XC-177
— wborder_set, 3XC-177
use complex characters (and renditions) to draw
lines
— hline_set, 3XC-816
— mvhline_set, 3XC-816
— mvvline_set, 3XC-816
— mvwhline_set, 3XC-816
— mvwvline_set, 3XC-816
— vline_set, 3XC-816
— whline_set, 3XC-816
— wvline_set, 3XC-816
use single-byte characters (and renditions) to draw
lines
— hline, 3XC-815
— mvhline, 3XC-815
— mvvline, 3XC-815
— mvwhline, 3XC-815
Index—79
use single-byte characters (and renditions) to draw lines, continued
— mvwvline, 3XC-815
— vline, 3XC-815
— whline, 3XC-815
— vwline, 3XC-815

use_env — set values of lines and columns, 3XC-1748

user context
— makecontext, 3C-945
— swapcontext, 3C-945

user IDs
get character-string representation — cuserid, 3S-347

users
return information from remote machines — rusers, rnusers, 3N-1352

usleep — suspend execution for interval in microseconds, 3C-1749

utmp file
access entry — getutent, 3C-791
find the slot of current user — ttyslot, 3C-1738

utmpname — access utmp file entry, 3C-791

utmpx file
access entry — getutxent, 3C-793

utmpxname — access utmpx file entry, 3C-793

V
variables used for X/Open Curses —
global_variables, 3XC-810

vfprintf — formatted output conversion, 3B-1171

vstab file
getvfsent, 3C-796

vid_attr — display string with video attributes, 3XC-1750

vid_puts — display string with video attributes, 3XC-1750

vidattr — display string with video attributes, 3XC-1750

vidputs — display string with video attributes, 3XC-1750

virtual memory, continued
optimizing usage of user mapped memory — madvise, 3-941

vlfmt — display error message in standard format and pass to logging and monitoring services, 3C-1751

vline — use single-byte characters (and renditions) to draw lines, 3XC-815

vline_set — use complex characters (and renditions) to draw lines, 3XC-816

volmgt_acquire — reserve removable media device, 3X-1753

volmgt_check — have Volume Management check for media, 3X-1756

volmgt_feature_enabled — check whether specific Volume Management features are enabled, 3X-1757

volmgt_inuse — check whether or not Volume Management is managing a pathname, 3X-1758

volmgt_release — release removable media device reservation, 3X-1759

volmgt_root — return the Volume Management root directory, 3X-1761

volmgt_running — return whether or not Volume Management is running, 3X-1762

volmgt_symdev — convert between Volume Management symbolic names, and the devices that correspond to them, 3X-1763

volmgt_symname — convert between Volume Management symbolic names, and the devices that correspond to them, 3X-1763

vpfmt — display error message in standard format and pass to logging and monitoring services, 3C-1765

vprintf — formatted output conversion, 3B-1171

vsprintf — formatted output conversion, 3B-1171

vsyslog() — log message with variable argument list, 3-1769

VTOC, disk’s
read a disk’s VTOC — read_vtoc, 3X-1266
write a disk’s VTOC — write_vtoc, 3X-1266

vw_printw — write formatted output to window,
vw_scanw — read formatted input from window, 3XC-1041
vwprintw — write formatted output to window, 3XC-1041
vwscanf — read formatted input from window, 3XC-1041

wait for transmission of output — tcdrain, 3-1544
watof — convert wide character string to doubleprecision number, 3C-1783
watoi — convert wide character string to long integer, 3C-1785
watol — convert wide character string to long integer, 3C-1785
watoll — convert wide character string to long integer, 3C-1785
wattr_get — control window attributes, 3XC-154
wattr_off — control window attributes, 3XC-154
wattr_on — control window attributes, 3XC-154
wattr_set — control window attributes, 3XC-154
wattron — change foreground window attributes, 3XC-156, 3X-280
wattrset — change foreground window attributes, 3XC-156, 3X-280
wbkgd — set the background character (and rendition) of window, 3XC-172
wbkgdset — set the background character (and rendition) of window, 3XC-172
wbkgnd — set or get the background character (and rendition) of window using a complex character, 3XC-173
wbkgndset — set or get the background character (and rendition) of window using a complex character, 3XC-173
wborder — add a single-byte border to a window, 3XC-175
wborder_set — use complex characters (and renditions) to draw borders, 3XC-177
wchar_t string number conversion — scanf, 3C-1805
wchgat — change the rendition of characters in a window, 3XC-205
wclear — clear a window, 3XC-206
wclrbot — clear to the end of a window, 3XC-213
wclrtoeol — clear to the end of a line, 3XC-214
wcolor_set — control window attributes, 3XC-154

wadd_wch — add a complex character (with rendition) to a window, 3XC-128
wadd_wchnstr — copy a string of complex characters (with renditions) to a window, 3XC-130
wadd_wchstr — copy a string of complex characters (with renditions) to a window, 3XC-130
waddch — add a character (with rendition) to a window, 3XC-118
waddchnstr — copy a character string (with renditions) to a window, 3XC-119
waddchstr — copy a character string (with renditions) to a window, 3XC-119
waddnstr — add a multi-byte character string (without rendition) to a window, 3XC-121
waddnwstr — add a string of wchar_t characters to a curses window and advance cursor, 3X-277, 3XC-123
waddstr — add a multi-byte character string (without rendition) to a window, 3XC-121
waddwch — add a wchar_t character (with attributes) to a curses window and advance cursor, 3X-272
waddwchnstr — add string of wchar_t characters (and attributes) to a curses window, 3X-275
waddwchstr — add string of wchar_t characters (and attributes) to a curses window, 3X-275
waddwstr — add a string of wchar_t characters to a curses window and advance cursor, 3X-277, 3XC-123
wadjcurspos — moving the cursor by character, 3X-278
wait3 — wait for process to terminate or stop, 3B-1774, 3C-1771

Index—81
wcscat — wide character string operations, 3C-1791
wcscchr — wide character string operations, 3C-1792
wcscmp — wide character string operations, 3C-1791
wcscoll — wide character string comparison using collating information, 3C-1781
wcscpy — wide character string operations, 3C-1791
wcscspn — wide character string operations, 3C-1792
wcsetno — get information on EUC codesets, 3C-237
wcsftime — convert date and time to wide character string, 3C-1782
wcslen — wide character string operations, 3C-1792
wcscncat — wide character string operations, 3C-1791
wcscncmp — wide character string operations, 3C-1791
wcscncpy — wide character string operations, 3C-1791
wcspbrk — wide character string operations, 3C-1792
wcsrchr — wide character string operations, 3C-1792
wcscspn — wide character string operations, 3C-1792
wcstod — convert wide character string to double-precision number, 3C-1783
wcstok — wide character string operations, 3C-1792
wcstol — convert wide character string to long integer, 3C-1785
wcstombs — convert a wide-character string to a character string, 3C-1787
wcstoul — convert wide character string to unsigned long, 3C-1788
wcstring — wide character string operations, 3C-1790
wcswcs — wide character string operations, 3C-1792
wcswidth — number of column positions of a wide-character string, 3C-1794
wcsxfrm — wide character string transformation, 3C-1795
wctomb — convert a wide-character code to a character, 3C-1797
wctrans — define wide-character mapping, 3C-1798
wctype — define character class, 3C-1799
wcursyncup — synchronize window with its parents or children, 3XC-1518
wcwidth — number of column positions of a wide-character code, 3C-1800
wdech — remove a character, 3XC-357
wdeleteln — remove a line, 3XC-360
wecho_wchar — add a complex character and refresh window, 3XC-440
wechochar — add a single-byte character and refresh window, 3XC-439
wechowchar — add a wchar_t character (with attributes) to a curses window and advance cursor, 3X-272
werase — clear a window, 3XC-206
wget_wch — get a wide character from terminal, 3XC-799
wget_wstr — get a wide character string from terminal, 3XC-739
wgetbgnd — set or get the background character (and rendition) of window using a complex character, 3XC-173
wgetch — get a single-byte character from terminal, 3XC-697
wgetn_wstr — get a wide character string from terminal, 3XC-739
wgetnstr — get a multibyte character string from terminal, 3XC-738
wgetnwstr — get wchar_t character strings from curses terminal keyboard, 3X-301
wgetstr — get a multibyte character string from terminal, 3XC-738

Index—82
wgetwch — get (or push back) wchar_t characters from curses terminal keyboard, 3X-297
wgetwstr — get wchar_t character strings from curses terminal keyboard, 3X-301
whline — use single-byte characters (and renditions) to draw lines, 3XC-815
whline_set — use complex characters (and renditions) to draw lines, 3XC-816
wide character string comparison using collating information
   — wcscoll, 3C-1781
   — wscoll, 3C-1781
wide character string operations
   — wcscat, 3C-1791
   — wcscmp, 3C-1791
   — wcscpy, 3C-1791
   — wcscspn, 3C-1792
   — wcslen, 3C-1792
   — wcscnmp, 3C-1791
   — wcsspn, 3C-1791
   — wcspbrk, 3C-1792
   — wcscrbp, 3C-1792
   — wcscspn, 3C-1792
   — wcstrbk, 3C-1792
   — wcstok, 3C-1792
   — wcstrch, 3C-1792
   — wcsncmp, 3C-1791
   — wcstok, 3C-1792
   — wcstrch, 3C-1792
   — wcstok, 3C-1792
wide character string to long integer, convert
   — watoi, 3C-1785
   — watol, 3C-1785
   — watoll, 3C-1785
   — wcsstol, 3C-1785
   — wstol, 3C-1785
wide character string transformation
   — wcsxfrm, 3C-1795
   — wsxfrm, 3C-1795
wide-character code classification functions
   — isenglish, 3C-863
   — isideogram, 3C-863
   — isnumber, 3C-863
   — isphonogram, 3C-863
wide-character code classification functions, continued
   — isspecial, 3C-863
   — iswalnum, 3C-863
   — iswalpha, 3C-863
   — iswascii, 3C-863
   — iswcntrl, 3C-863
   — iswdigit, 3C-863
   — iswgraph, 3C-863
   — iswlower, 3C-863
   — iswprint, 3C-863
   — iswpunct, 3C-863
   — iswspace, 3C-863
   — iswupper, 3C-863
   — iswxdigit, 3C-863
wide-character mapping — towctrans, 3C-1690
win_wch — retrieve a complex character (with rendition), 3XC-853
win_wchnstr — retrieve complex character string (with rendition), 3XC-854
win_wchstr — retrieve complex character string (with rendition), 3XC-854
winch — return a single-byte character (with rendition), 3XC-829
winchwnstr — retrieve a single-byte character string (with rendition), 3XC-830
winchstr — retrieve a single-byte character string (with rendition), 3XC-830
windex — wide character string operations, 3C-1792
winnstr — retrieve a multibyte character string (without rendition), 3XC-839
winnwstr — get a string of wchar_t characters from a curses window, 3X-318, 3XC-841
wins_nwstr — insert a wide character string, 3XC-848
wins_wch — insert a wide character string, 3XC-851
wins_wstr — insert a complex character, 3XC-851
wins_wch — insert a wide character string, 3XC-848
winsch — insert a character, 3XC-843
winsdelln — insert/delete lines to/from the window, 3XC-844
winsertln — insert a line in a window, 3XC-845

Index–83
winsnstr — insert a multibyte character string, 3XC-846
winsnwstr — insert wchar_t string before character under the cursor in a curses window, 3X-314
winsstr — insert a multibyte character string, 3XC-846
winstr — retrieve a multibyte character string (without rendition), 3XC-839
winswc — insert a wchar_t character before the character under the cursor in a curses window, 3X-313
winswstr — insert wchar_t string before character under the cursor in a curses window, 3X-314
winwch — get a wchar_t character and its attributes from a curses window, 3X-316
winwchnstr — get a string of wchar_t characters (and attributes) from a curses window, 3X-317
winwchstr — get a string of wchar_t characters (and attributes) from a curses window, 3X-317
winwstr — get a string of wchar_t characters from a curses window, 3X-318, 3XC-841
wmove — move cursor in window, 3XC-1009
wmoveprevch — moving the cursor by character, 3X-278
wmoveprevch — moving the cursor by character, 3X-278
wnoutrefresh — refresh windows and lines, 3XC-433
wordexp — perform word expansions, 3C-1801
wordfree — perform word expansions, 3C-1801
working directory
get pathname — getwd, 3C-802
wprintw — write formatted output to window, 3XC-1040
wredrawln — redraw screen or portion of screen, 3XC-1281
wrefresh — refresh windows and lines, 3XC-433
windex — wide character string operations, 3C-1792
write formatted output to window
— mvprintw, 3XC-1040
write formatted output to window, continued
— mvwprintw, 3XC-1040
— printw, 3XC-1040
— vwwprintw, 3XC-1040
— wprintw, 3XC-1040
write screen contents to/from a file
— scr_dump, 3XC-1373
— scr_init, 3XC-1373
— scr_restore, 3XC-1373
— scr_set, 3XC-1373
write_vtoc — read and write a disk’s VTOC, 3X-1266
wscanf — read formatted input from window, 3XC-1041
wcasecmp — Process Code string operations, 3C-1806
wcol — Process Code string operations, 3C-1806
wscoll — wide character string comparison using collating information, 3C-1781
wscr — scroll a window, 3XC-1374
wsdup — Process Code string operations, 3C-1806
wsetscrreg — set terminal output controls, 3XC-207
wcasecmp — Process Code string operations, 3C-1806
wprintf — formatted output conversion, 3C-1804
wscanf — formatted input conversion, 3C-1805
wstandend — curses character and window attribute control routines, 3X-280, 3XC-1478
wstandout — curses character and window attribute control routines, 3X-280, 3XC-1478
wstod — convert wide character string to double-precision number, 3C-1783
wstol — convert wide character string to long integer, 3C-1785
wstoustr — code conversion for Process Code and File Code, 3C-1513
wsxfrm — wide character string transformation, 3C-1795
wsyncdown — synchronize window with its parents or children, 3XC-1518

Index—84
wsyncup — synchronize window with its parents or children, 3XC-1518
wtimeout — set timed blocking or non-blocking read, 3XC-1103
wtouchln — control window refresh, 3XC-858
wunctrl — convert a wide character to printable form, 3XC-1807
wvline — use single-byte characters (and renditions) to draw lines, 3XC-815
wvline_set — use complex characters (and renditions) to draw lines, 3XC-816

X

XDR library routines
— xdr, 3N-1808
— xdr_admin, 3N-1810
— xdr_control, 3N-1810
— xdr_getpos, 3N-1810
— xdr_inline, 3N-1810
— xdr_setpos, 3N-1810
— xdr_sizeof, 3N-1810
— xdrrec_endofrecord, 3N-1810
— xdrrec_eof, 3N-1810
— xdrrec_readbytes, 3N-1810
— xdrrec_skiprecord, 3N-1810

XDR library routines for complex data structures
— xdr_array, 3N-1813
— xdr_bytes, 3N-1813
— xdr_complex, 3N-1813
— xdrOpaque, 3N-1813
— xdr_pointer, 3N-1813
— xdr_reference, 3N-1813
— xdr_string, 3N-1813
— xdr_union, 3N-1813
— xdr_vector, 3N-1813
— xdr_wrapstring, 3N-1813

XDR library routines for RPC
— rpc_xdr, 3N-1349
— xdr_accepted_reply, 3N-1349
— xdr_authsys_parms, 3N-1349
— xdr_callhdr, 3N-1349
— xdr_callmsg, 3N-1349
— xdrOpaque_author, 3N-1349
— xdr_rejected_reply, 3N-1349

XDR library routines for RPC, continued
— xdr_replymsg, 3N-1349

XDR library routines for simple data structures
— xdr_bool, 3N-1818
— xdr_char, 3N-1818
— xdr_double, 3N-1818
— xdr_enum, 3N-1818
— xdr_float, 3N-1818
— xdr_free, 3N-1818
— xdr_hyper, 3N-1818
— xdr_int, 3N-1818
— xdr_long, 3N-1818
— xdr_longlong_t, 3N-1818
— xdr_quadruple, 3N-1818
— xdr_short, 3N-1818
— xdr_simple, 3N-1818
— xdr_u_char, 3N-1818
— xdr_u_hyper, 3N-1818
— xdr_u_int, 3N-1818
— xdr_u_long, 3N-1818
— xdr_u_longlong_t, 3N-1818
— xdr_u_short, 3N-1818
— xdr_u_void, 3N-1818

XDR stream creation library routines
— xdr_create, 3N-1816
— xdr_destroy, 3N-1816
— xdrmem_create, 3N-1816
— xdrrec_create, 3N-1816
— xdrstdio_create, 3N-1816

xdr_statstime — get performance data from remote kernel, 3N-1351
xdr_statsvar — get performance data from remote kernel, 3N-1351
xfn — overview of the XFN interface, 3N-1821

XFN attribute
— fn_attribute_add, 3N-551
— fn_attribute_assign, 3N-551
— fn_attribute_copy, 3N-551
— fn_attribute_create, 3N-551
— fn_attribute_destroy, 3N-551
— fn_attribute_first, 3N-551
— fn_attribute_identifier, 3N-551
— fn_attribute_next, 3N-551
— fn_attribute_remove, 3N-551
— fn_attribute_syntax, 3N-551

Index–85
XFN attribute, continued
  — FN_attribute_t, 3N-551
  — fn_attribute_valuecount, 3N-551

XFN attributes, a set of
  — fn_attrset_add, 3N-567
  — fn_attrset_assign, 3N-567
  — fn_attrset_copy, 3N-567
  — fn_attrset_count, 3N-567
  — fn_attrset_create, 3N-567
  — fn_attrset_destroy, 3N-567
  — fn_attrset_first, 3N-567
  — fn_attrset_get, 3N-567
  — fn_attrset_next, 3N-567
  — fn_attrset_remove, 3N-567
  — FN_attrset_t, 3N-567

XFN compound name
  — fn_compound_name_append_comp, 3N-575
  — fn_compound_name_assign, 3N-575
  — fn_compound_name_copy, 3N-575
  — fn_compound_name_count, 3N-575
  — fn_compound_name_delete_all, 3N-575
  — fn_compound_name_delete_comp, 3N-575
  — fn_compound_name_destroy, 3N-575
  — fn_compound_name_first, 3N-575
  — fn_compound_name_from_syntax_attrs, 3N-575
  — fn_compound_name_get_syntax_attrs, 3N-575
  — fn_compound_name_insert_comp, 3N-575
  — fn_compound_name_is_empty, 3N-575
  — fn_compound_name_is_equal, 3N-575
  — fn_compound_name_is_prefix, 3N-575
  — fn_compound_name_is_suffix, 3N-575
  — fn_compound_name_last, 3N-575
  — fn_compound_name_next, 3N-575
  — fn_compound_name_prefix, 3N-575
  — fn_compound_name_prepend_comp, 3N-575
  — fn_compound_name_prev, 3N-575
  — fn_compound_name_suffix, 3N-575

XFN compound name, continued
  — FN_compound_name_t, 3N-575
  — fn_string_from_compound_name, 3N-575

an XFN context — FN_ctx_t, 3N-601
an XFN identifier — FN_identifier_t, 3N-604

XFN reference
  — fn_ref_addrcount, 3N-609
  — fn_ref_append_addr, 3N-609
  — fn_ref_assign, 3N-609
  — fn_ref_copy, 3N-609
  — fn_ref_create, 3N-609
  — fn_ref_create_link, 3N-609
  — fn_ref_delete_addr, 3N-609
  — fn_ref_delete_all, 3N-609
  — fn_ref_description, 3N-609
  — fn_ref_destroy, 3N-609
  — fn_ref_first, 3N-609
  — fn_ref_insert_addr, 3N-609
  — fn_ref_is_link, 3N-609
  — fn_ref_link_name, 3N-609
  — fn_ref_next, 3N-609
  — fn_ref_prepend_addr, 3N-609
  — FN_ref_t, 3N-609
  — fn_ref_type, 3N-609

XFN status object
  — fn_status_advance_by_name, 3N-622
  — fn_status_append_remaining_name, 3N-622
  — fn_status_append_resolved_name, 3N-622
  — fn_status_assign, 3N-622
  — fn_status_code, 3N-622
  — fn_status_copy, 3N-622
  — fn_status_create, 3N-622
  — fn_status_description, 3N-622
  — fn_status_destroy, 3N-622
  — fn_status_diagnostic_message, 3N-622
  — fn_status_is_success, 3N-622
  — fn_status_link_code, 3N-622
  — fn_status_link_diagnostic_message, 3N-622
  — fn_status_link_remaining_name, 3N-622
3N-622
XFN status object, continued

- fn_status_link_resolved_name, 3N-622
- fn_status_link_resolved_ref, 3N-622
- fn_status_remaining_name, 3N-622
- fn_status_resolved_name, 3N-622
- fn_status_resolved_ref, 3N-622
- fn_status_set, 3N-622
- fn_status_set_code, 3N-622
- fn_status_set_diagnostic_message, 3N-622
- fn_status_set_link_code, 3N-622
- fn_status_set_link_diagnostic_message, 3N-622
- fn_status_set_link_remaining_name, 3N-622
- fn_status_set_link_resolved_name, 3N-622
- fn_status_set_link_resolved_ref, 3N-622
- fn_status_set_remaining_name, 3N-622
- fn_status_set_resolved_name, 3N-622
- fn_status_set_resolved_ref, 3N-622
- fn_status_set_success, 3N-622
- FN_status_t, 3N-622

xfn_attributes — an overview of XFN attribute operations, 3N-1822

xfn_compound_names — XFN compound syntax: an overview of XFN model for compound name parsing, 3N-1826

xfn_status_codes — descriptions of XFN status codes, 3N-1831

XFN Status Codes, 3N-1831

Y

y₀ — Bessel functions of the second kind, 3M-1835
y₁ — Bessel functions of the second kind, 3M-1835
yn — Bessel functions of the second kind, 3M-1835