SunOS Reference Manual
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.

**AVAILABILITY**

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

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

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

**MT-LEVEL**

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

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

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 greater 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.

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
FN_attribute_t, fn_attribute_create, fn_attribute_destroy, fn_attribute_copy, fn_attribute_assign, fn_attribute_identifier, fn_attribute_syntax, fn_attribute_valuecount, fn_attribute_first, fn_attribute_next, fn_attribute_add, fn_attribute_remove – an XFN attribute

SYNOPSIS
cc [ flag ... ] file ... -lxfn [ library ... ]
#include <xfn/xfn.h>
FN_attribute_t *fn_attribute_create( const FN_identifier_t *attribute_id,
    const FN_identifier_t *attribute_syntax);
void fn_attribute_destroy(FN_attribute_t *attr);
FN_attribute_t *fn_attribute_copy( const FN_attribute_t *attr);
FN_attribute_t *fn_attribute_assign( FN_attribute_t *dst,
    const FN_attribute_t *src);
const FN_identifier_t *fn_attribute_identifier( const FN_attribute_t *attr);
const FN_identifier_t *fn_attribute_syntax( const FN_attribute_t *attr);
unsigned int fn_attribute_valuecount( const FN_attribute_t *attr);
const FN_attrvalue_t *fn_attribute_first( const FN_attribute_t *attr,
    void **iter_pos);
const FN_attrvalue_t *fn_attribute_next( const FN_attribute_t *attr,
    void **iter_pos);
int fn_attribute_add( FN_attribute_t *attr, const FN_attrvalue_t *attribute_value,
    unsigned int exclusive);
int fn_attribute_remove( FN_attribute_t *attr, const FN_attrvalue_t *attribute_value);

MT-LEVEL
Safe.

DESCRIPTION
An attribute has an attribute identifier, a syntax, and a set of distinct values. Each value is a sequence of octets. The operations associated with objects of type FN_attribute_t allow the construction, destruction, and manipulation of an attribute and its value set.

The attribute identifier and its syntax are specified using an FN_identifier_t.
fn_attribute_create() creates a new attribute object with the given identifier and syntax, and an empty set of values. fn_attribute_destroy() releases the storage associated with attr. fn_attribute_copy() returns a copy of the object pointed to by attr.
fn_attribute_assign() makes a copy of the attribute object pointed to by src and assigns it to dst, releasing any old contents of dst. A pointer to the same object as dst is returned.
fn_attribute_identifier() returns the attribute identifier of attr. fn_attribute_syntax() returns the attribute syntax of attr. fn_attribute_valuecount() returns the number of attribute values in attr.

modified 4 Nov 1994
**fn_attribute_first()** and **fn_attribute_next()** are used to enumerate the values of an attribute. Enumeration of the values of an attribute may return the values in any order. **fn_attribute_first()** returns an attribute value from **attr** and sets the iteration marker **iter_pos**. Subsequent calls to **fn_attribute_next()** returns the next attribute value identified by **iter_pos** and advances **iter_pos**. Adding or removing values from an attribute invalidates any iteration markers that the caller holds.

**fn_attribute_add()** adds a new value **attribute_value** to **attr**. The operation succeeds (but no change is made) if **attribute_value** is already in **attr** and **exclusive** is zero; the operation fails if **attribute_value** is already in **attr** and **exclusive** is nonzero. **fn_attribute_remove()** removes **attribute_value** from **attr**. The operation succeeds even if **attribute_value** is not amongst **attr**’s values.

**RETURN VALUE**

- **fn_attribute_first()** returns 0 if the attribute contains no values.
- **fn_attribute_next()** returns 0 if there are no more values to be returned in the attribute (as identified by the iteration marker) or if the iteration marker is invalid.
- **fn_attribute_add()** and **fn_attribute_remove()** return 1 if the operation succeeds, 0 if it fails.

**APPLICATION 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)**.

**SEE ALSO**

- **FN_attrvalue_t(3N)**, **FN_attrset_t(3N)**, **FN_identifier_t(3N)**, **fn_attr_get(3N)**, **fn_attr_modify(3N)**, **xfn_attributes(3N)**, **xfn(3N)**
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);

MT-LEVEL

Safe.

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() 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() 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.
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 VALUE**

- `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.

**APPLICATION**

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)`.

**SEE ALSO**

`FN_attribute_t(3N)`, `FN_attrset_t(3N)`, `FN_identifier_t(3N)`, `fn_attr_multi_modify(3N)`, `fn_attr_modify(3N)`, `xFn_attributes(3N)`, `xfn(3N)`
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 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);

MT-LEVEL
Safe.

DESCRIPTION
An attribute set is a set of attribute objects with distinct identifiers. The
fn_attr_multi_get() operation takes an attribute set as parameter and returns an attribute
set. The fn_attr_get_ids() 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 enumeration 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

modified 4 Nov 1994
from the set and sets \textit{iter_pos} after the first attribute. \texttt{fn_attrset_next()} returns the attribute following \textit{iter_pos} and advances \textit{iter_pos}.

\texttt{fn_attrset_add()} adds the attribute \textit{attr} to the attribute set \textit{aset}, replacing the attribute’s values if the identifier of \textit{attr} is not distinct in \textit{aset} and \textit{exclusive} is zero. If \textit{exclusive} is nonzero and the identifier of \textit{attr} is not distinct in \textit{aset}, the operation fails.

\texttt{fn_attrset_remove()} removes the attribute with the identifier \textit{attr_id} from \textit{aset}. The operation succeeds even if no such attribute occurs in \textit{aset}.

\begin{itemize}
\item \textbf{RETURN VALUE}
\texttt{fn_attrset_first()} returns 0 if the attribute set is empty. \texttt{fn_attrset_next()} returns 0 if there are no more attributes in the set.
\texttt{fn_attrset_add()} and \texttt{fn_attrset_remove()} return 1 if the operation succeeds, and 0 if the operation fails.
\end{itemize}

\begin{itemize}
\item \textbf{APPLICATION 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)}.
\end{itemize}

\begin{itemize}
\item \textbf{SEE ALSO}
\texttt{FN_attribute_t(3N)}, \texttt{FN_attrvalue_t(3N)}, \texttt{FN_identifier_t(3N)}, \texttt{fn_attr_multi_get(3N)}, \texttt{fn_attr_get_ids(3N)}, \texttt{xfn_attributes(3N)}, \texttt{xfn(3N)}
\end{itemize}
NAME       FN_attrvalue_t – an XFN attribute value

SYNOPSIS   cc [ flag ...] file ... -Ixfn [ 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:

```c
typedef struct {
    size_t length;
    void *contents;
} FN_attrvalue_t;
```

SEE ALSO   FN_attribute_t(3N), fn_attr_get_values(3N), xfn(3N)
NAME

FN_composite_name_t, fn_composite_name_create, fn_composite_name_destroy,
fn_composite_name_from_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_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_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_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);

MT-LEVEL Safe.

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_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 (i.e. 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.

modified 4 Nov 1994 3N-13
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()`, `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. (So for example, a subsequent call to `fn_composite_name_suffix()` will return the remainder of the name.) Otherwise it returns `0` and value of the iteration marker is undefined. The function `fn_composite_name_is_suffix()` is similar. It tests if a one composite name is a suffix of another. If so it returns `1` and sets the iteration marker immediately preceding the suffix.

`fn_composite_name_prepend_comp()` and `fn_composite_name_append_comp()` prepends and appends a single component to the given composite name, respectively. These operations invalidate any iteration marker the client holds for that object. `fn_composite_name_insert_comp()` inserts a single component before `iter_pos` to the given composite name and sets `iter_pos` to be immediately after the component just inserted. `fn_composite_name_delete_comp()` deletes the component located before `iter_pos` from the given composite name and sets `iter_pos` back one component.
The functions `fn_composite_name_prepend_name()`, `fn_composite_name_append_name()` and `fn_composite_name_insert_name()` perform the same update functions as their `_comp` counterparts, respectively, except that multiple components are being added, rather than single components. For `fn_composite_name_insert_name()` sets `iter_pos` to be immediately after the name just added.

**RETURN VALUE**

The functions `fn_composite_name_is_empty()`, `fn_composite_name_is_equal()`, `fn_composite_name_is_suffix()` and `fn_composite_name_is_prefix()` return 1 if the test indicated is true; 0 otherwise.

The update functions `fn_composite_name_prepend_comp()`, `fn_composite_name_append_comp()`, `fn_composite_name_insert_comp()`, `fn_composite_name_delete_comp()` and their `_name` counterparts return 1 if the update was successful; 0 otherwise.

If a function is expected to return a pointer to an object, a null pointer (0) is returned if the function fails.

**ERRORS**

Code set mismatches that occur during the composition of the string form or during comparisons of composite names are resolved in an implementation-dependent way. `fn_string_from_composite_name()`, `fn_composite_name_is_equal()`, `fn_composite_name_is_suffix()` and `fn_composite_name_is_prefix()` sets `status` to `FN_E_INCOMPATIBLE_CODE_SETS` for composite names whose components have code sets that are determined by the implementation to be incompatible.

**SEE ALSO**

`FN_string_t(3N)`, `xfn(3N)`
NAME

FN_compound_name_t, fn_compound_name_from_syntax_attrs,
fn_compound_name_get_syntax_attrs, fn_compound_name_destroy,
fn_string_from_compound_name, fn_compound_name_copy,
fn_compound_name_assign, fn_compound_name_count, fn_compound_name_first,
fn_compound_name_next, fn_compound_name_prev, fn_compound_name_last,
fn_compound_name_prefix, fn_compound_name_suffix, fn_compound_name_is_empty,
fn_compound_name_is_equal, fn_compound_name_is_prefix,
fn_compound_name_is_suffix, fn_compound_name_prepend_comp,
fn_compound_name_append_comp, fn_compound_name_insert_comp,
fn_compound_name_delete_comp, fn_compound_name_delete_all – an XFN compound name

SYNOPSIS

cc [ flag ... ] file ... -lxfn [ library ... ]

#include <xfn/xfn.h>

FN_compound_name_t *fn_compound_name_from_syntax_attrs(
    const FN_attrset_t *aset, const FN_string_t *name, FN_status_t *status);
FN_attrset_t *fn_compound_name_get_syntax_attrs(
    const FN_compound_name_t *name);
void fn_compound_name_destroy(FN_compound_name_t *name);
FN_string_t *fn_string_from_compound_name(const FN_compound_name_t *name);
FN_compound_name_t *fn_compound_name_copy(const FN_compound_name_t *name);
FN_compound_name_t *fn_compound_name_assign(FN_compound_name_t *dst,
    const FN_compound_name_t *src);
unsigned int fn_compound_name_count(const FN_compound_name_t *name);
const FN_string_t *fn_compound_name_first(const FN_compound_name_t *name,
    void **iter_pos);
const FN_string_t *fn_compound_name_next(const FN_compound_name_t *name,
    void **iter_pos);
const FN_string_t *fn_compound_name_prev(const FN_compound_name_t *name,
    void **iter_pos);
const FN_string_t *fn_compound_name_last(const FN_compound_name_t *name,
    void **iter_pos);
FN_compound_name_t *fn_compound_name_prefix(const FN_compound_name_t *name,
    const void *iter_pos);
FN_compound_name_t *fn_compound_name_suffix(const FN_compound_name_t *name,
    const void *iter_pos);
int fn_compound_name_is_empty(const FN_compound_name_t *name);
int fn_compound_name_is_equal(" const FN_compound_name_t *name1,
    const FN_compound_name_t *name2, unsigned int *status);

int fn_compound_name_is_prefix(const FN_compound_name_t *name,
    const FN_compound_name_t *pre, void **iter_pos, unsigned int *status);

int fn_compound_name_is_suffix(const FN_compound_name_t *name,
    const FN_compound_name_t *suffix, void **iter_pos, unsigned int *status);

int fn_compound_name_prepend_comp( FN_compound_name_t *name,
    const FN_string_t *atomic_comp, unsigned int *status);

int fn_compound_name_append_comp( FN_compound_name_t *name,
    const FN_string_t *atomic_comp, unsigned int *status);

int fn_compound_name_insert_comp( FN_compound_name_t *name,
    void **iter_pos, const FN_string_t *atomic_comp, unsigned int *status);

int fn_compound_name_delete_comp( FN_compound_name_t *name,
    void **iter_pos);

int fn_compound_name_delete_all( FN_compound_name_t *name);

MT-LEVEL Safe.

DESCRIPTION Most applications treat names as opaque data and 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_syntax.

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" (FN_ID_STRING syntax) attribute identifying the namespace syntax of the string name. The value "standard" (FN_ID_STRING syntax) 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 using 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 function fn_compound_name_is_equal(), fn_compound_name_is_prefix(), fn_compound_name_is_suffix(), tests 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. (So for example, a subsequent call to fn_compound_name_suffix() will return the remainder of the name.) Otherwise it returns 0 and value of the iteration marker is undefined. The function fn_compound_name_is_suffix() is similar. It tests if a one compound name is a suffix of another. If so it returns 1 and sets the iteration marker immediately preceding the suffix.

fn_compound_name_prepend_comp() and fn_compound_name_append_comp() prepends and appends a single atomic component to the given compound name, respectively. These operations invalidates any iteration marker the client holds for that object. fn_compound_name_insert_comp() inserts an atomic component before iter_pos to the given compound name and sets iter_pos to be immediately after the component just inserted. fn_compound_name_delete_comp() deletes the atomic component located before iter_pos from the given compound name and sets iter_pos back one component. fn_compound_name_delete_all() deletes all the atomic components from name.
RETURN VALUE

The functions `fn_compound_name_is_empty()`, `fn_compound_name_is_equal()`, `fn_compound_name_is_suffix()` and `fn_compound_name_is_prefix()` return 1 if the test indicated is true; 0 otherwise.

The update functions `fn_compound_name_prepend_comp()`, `fn_compound_name_append_comp()`, `fn_compound_name_insert_comp()`, `fn_compound_name_delete_comp()` and `fn_compound_name_delete_all()` return 1 if the update was successful; 0 otherwise.

If a function is expected to return a pointer to an object, a null pointer (0) is returned if the function fails.

ERRORS

When the function `fn_compound_name_from_syntax_attrs()` fails, it returns in `status` a status code. 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.

`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()` and `fn_compound_name_insert_comp()` may return in `status` when the code set of the given string is incompatible with that of the compound name.

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_compound_names(3N)`, `xfn(3N)`
NAME   FN_ctx_t – an XFN context

MT-LEVEL Safe.

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.

These are the interfaces:

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

#include <xfn/xfn.h>

FN_ctx_t *fn_ctx_handle_from_initial(FN_status_t *status);
FN_ctx_t *fn_ctx_handle_from_ref(const FN_ref_t *ref, 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);
```

3N-20 modified 4 Nov 1994
The following contains a brief summary of these operations.

\texttt{fn\_ctx\_handle\_from\_initial()} returns a pointer to an Initial Context that provides a starting point for resolution of composite names. \texttt{fn\_ctx\_handle\_from\_ref()} returns a handle to an FN\_ctx\_t object using the given reference \texttt{ref}. \texttt{fn\_ctx\_get\_ref()} returns the reference of the context \texttt{ctx}. \texttt{fn\_ctx\_handle\_destroy()} releases the resources associated with the FN\_ctx\_t object \texttt{ctx}; it does not affect the state of the context itself.

\texttt{fn\_ctx\_lookup()} returns the reference bound to \texttt{name} resolved relative to \texttt{ctx}.

\texttt{fn\_ctx\_list\_names()} is used to enumerate the atomic names bound in the context named by \texttt{name} resolved relative to \texttt{ctx}. \texttt{fn\_ctx\_list\_bindings()} is used to enumerate the atomic names and their references in the context named by \texttt{name} resolved relative to \texttt{ctx}.

\texttt{fn\_ctx\_bind()} binds the composite name \texttt{name} to a reference \texttt{ref} resolved relative to \texttt{ctx}.

\texttt{fn\_ctx\_unbind()} unbinds \texttt{name} resolved relative to \texttt{ctx}. \texttt{fn\_ctx\_rename()} binds \texttt{newname} to the reference bound to \texttt{oldname} and unbinds \texttt{oldname}. \texttt{oldname} is resolved relative to \texttt{ctx}; \texttt{newname} is resolved relative to the target context.

\texttt{fn\_ctx\_create\_subcontext()} creates a new context with the given composite name \texttt{name} resolved relative to \texttt{ctx}. \texttt{fn\_ctx\_destroy\_subcontext()} destroys the context named by \texttt{name} resolved relative to \texttt{ctx}.

Normal resolution always follow links. \texttt{fn\_ctx\_lookup\_link()} lookups up \texttt{name} relative to \texttt{ctx}, following links except for the last atomic part of \texttt{name}, which must be bound to an XFN link.

\texttt{fn\_ctx\_get\_syntax\_attrs()} returns an attribute set containing attributes that describes a context's syntax. \texttt{name} must name a context.

\textbf{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).

\textbf{APPLICATION 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.

\textbf{SEE ALSO}

FN\_attrset\_t(3N), FN\_composite\_name\_t(3N), FN\_ref\_t(3N), FN\_status\_t(3N), fn\_ctx\_create\_subcontext(3N), fn\_ctx\_bind(3N), fn\_ctx\_destroy\_subcontext(3N), fn\_ctx\_handle\_destroy(3N), fn\_ctx\_handle\_from\_initial(3N),
fn_ctx_handle_from_ref(3N), fn_ctx_get_ref(3N), fn_ctx_get_syntax_attrs(3N),
fn_ctx_list_names(3N), fn_ctx_list_bindings(3N), fn_ctx_lookup(3N),
fn_ctx_lookup_link(3N), fn_ctx_rename(3N), fn_ctx_unbind(3N), xfn_links(3N),
xfn_status_codes(3N), xfn(3N)
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:

```c
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_ref_t(3N), FN_ref_addr_t(3N), FN_attribute_t(3N), xfn(3N)
### 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

```c
#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);
```

### MT-LEVEL

Safe.

### 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 zero, 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`.

### APPLICATION USAGE

The address type of an `FN_ref_addr_t` object is intended to identify the mechanism that should be used to reach the object using that address. The client must interpret the contents of the opaque data buffer of the address based on the type of the address, and on the type of the reference that the address is in. However, this interpretation is intended
to occur below the application layer. Most applications developers should not have to manipulate the contents of either address or reference objects themselves. These interfaces would generally be used within service libraries.

Multiple addresses in a single reference are intended to identify multiple communication endpoints for the same conceptual object. Multiple addresses may arise for various reasons, such as the object offering interfaces over more than one communication mechanism.

Manipulation of addresses using the operations described in this manual page does not affect their representation in the underlying naming system. Changes to addresses in the underlying naming system can only be effected through the use of the interfaces described in FN_ctx_t(3N).

SEE ALSO  FN_ctx_t(3N), FN_identifier_t(3N), FN_ref_t(3N), FN_string_t(3N), xfn(3N)
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);

MT-LEVEL

Safe.

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 pre-
cise 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 later added to the reference using the functions described below.
fn_ref_destroy() releases the storage associated with ref. fn_ref_copy() creates a copy of
ref and returns it. fn_ref_assign() creates a copy of src and assigns it to dst, releasing any
old contents of dst. A pointer to the same object as dst is returned.

`fn_ref_addrcount()` returns the number of addresses in the reference ref.

`fn_ref_first()` returns the first address in ref and sets iter_pos to be after the address. It returns 0 if there is no address in the list. `fn_ref_next()` returns the address following iter_pos in ref and sets iter_pos to be after the address. If the iteration marker iter_pos is at the end of the sequence, `fn_ref_next()` returns 0.

`fn_ref-prepend_addr()` adds addr to the front of the list of addresses in ref.

`fn_ref-append_addr()` adds addr to the end of the list of addresses in ref.

`fn_ref-insert_addr()` adds addr to ref before iter_pos and sets iter_pos to be immediately after the new reference added. `fn_ref-delete_addr()` deletes the address located before iter_pos in the list of addresses in ref and sets iter_pos back one address.

`fn_ref-delete-all()` deletes all addresses in ref.

`fn_ref-create_link()` creates a reference using the given composite name link_name as an address. The reference type of this reference is defined in `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 zero, 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 VALUE**

The operations `fn_ref-prepend_addr()`, `fn_ref-append_addr()`, `fn_ref-insert_addr()`, `fn_ref-delete_addr()` and `fn_ref-delete-all()` return 1 if the operation succeeds, 0 if the operation fails.

**APPLICATION 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)`.

modified 4 Nov 1994 3N-27
SEE ALSO

FN_composite_name_t(3N), FN_ctx_t(3N), FN_identifier_t(3N), FN_string_t(3N),
FN_ref_addr_t(3N), FN_string_t(3N), fn_ctx_lookup(3N), fn_ctx_lookup_link(3N),
xfn_links(3N), xfn(3N)
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);

MT-LEVEL Safe.

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.

MT-LEVEL Safe.

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.

3N-30 modified 4 Nov 1994
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`. **fn_status_set_resolved_name()** sets the resolved name part of the status object `stat` to `name`. **fn_status_set_resolved_ref()** sets the resolved reference part of the status object `stat` to `ref`. **fn_status_set_diagnostic_message()** sets the diagnostic message part of the status object to `msg`.

**fn_status_set_link_code()** sets the link status code field of the status object `stat` to indicate why resolution of the link failed. **fn_status_set_link_remaining_name()** sets the remaining link name part of the status object `stat` to `name`. **fn_status_set_link_resolved_name()** sets the resolved link name part of the status object `stat` to `name`. **fn_status_set_link_resolved_ref()** sets the resolved link reference part of the status object `stat` to `ref`. **fn_status_set_link_diagnostic_message()** sets the link diagnostic message part of the status object to `msg`.

**fn_status_append_resolved_name()** appends as additional components `name` to the resolved name part of the status object `stat`. **fn_status_append_remaining_name()** appends as additional components `name` to the remaining name part of the status object `stat`. **fn_status_advance_by_name()** removes `prefix` from the remaining name, and appends it to the resolved name. The resolved reference part is set to `resolved_ref`. This operation returns 1 on success, 0 if the `prefix` is not a prefix of the remaining name.

**RETURN VALUE**

The **fn_status_set_***(*) operations return 1 if the operation succeeds, 0 if the operation fails.

**SEE ALSO**

FN_composite_name_t(3N), FN_ref_t(3N), FN_string_t(3N), xfn_status_codes(3N), xfn(3N)
NAME
FN_string_t, fn_string_create, fn_string_destroy, fn_string_from_str,
fn_string_from_str_n, fn_string_str, fn_string_from_contents, fn_string_code_set,
fn_string_charcount, fn_string_bytecount, fn_string_contents, fn_string_copy,
fn_string_assign, fn_string_from_strings, fn_string_from_substring, fn_string_is_empty,
fn_string_compare, fn_string_compare_substring, fn_string_next_substring,
fn_string_prev_substring – a character string

SYNOPSIS
cc [ flag ... ] file ... -lxfn [ library ... ]
#include <xfn/xfn.h>
FN_string_t *fn_string_create(void);
void fn_string_destroy(FN_string_t *str);
FN_string_t *fn_string_from_str(const unsigned char *cstr);
FN_string_t *fn_string_from_str_n(const unsigned char *cstr, size_t n);
const unsigned char *fn_string_str(const FN_string_t *str, unsigned int *status);
FN_string_t *fn_string_from_contents(unsigned long code_set, const void *locale_info,
   size_t locale_info_len, size_t charcount, size_t bytecount, const void *contents,
   unsigned int *status);
unsigned long fn_string_code_set(const FN_string_t *str, const void **locale_info,
   size_t *locale_info_len);
size_t fn_string_charcount(const FN_string_t *str);
size_t fn_string_bytecount(const FN_string_t *str);
const void *fn_string_contents(const FN_string_t *str);
FN_string_t *fn_string_copy(const FN_string_t *str);
FN_string_t *fn_string_assign(FN_string_t *dst, const FN_string_t *src);
FN_string_t *fn_string_from_strings(unsigned int *status, const FN_string_t *s1,
   const FN_string_t *s2, ...)
FN_string_t *fn_string_from_substring(const FN_string_t *str, int first, int last);
int fn_string_is_empty(const FN_string_t *str);
int fn_string_compare(const FN_string_t *str1, const FN_string_t *str2,
   unsigned int string_case, unsigned int *status);
int fn_string_compare_substring(const FN_string_t *str1, int first, int last,
   const FN_string_t *str2, unsigned int string_case, unsigned int *status);
int fn_string_next_substring(const FN_string_t *str, const FN_string_t *sub, int index,
   unsigned int string_case, unsigned int *status);
int fn_string_prev_substring(const FN_string_t *str, const FN_string_t *sub, int index,
   unsigned int string_case, unsigned int *status);
DESCRIPTION

The FN_string_t type is used to represent character strings in the XFN interface. It provides insulation from specific string representations.

The FN_string_t supports multiple code sets. It provides creation functions for character strings of the code set of the current locale setting and a generic creation function for arbitrary code sets. The degree of support for the functions that manipulate FN_string_t for arbitrary code sets is implementation-dependent. An XFN implementation is required to support the ISO 646 code set; all other code sets are optional.

fn_string_destroy() releases the storage associated with the given string.

fn_string_create() creates an empty string.

fn_string_from_str() creates an FN_string_t object from the given null terminated string based on the code set of the current locale setting. The number of characters in the string is determined by the code set of the current locale setting. fn_string_from_str_n() is like fn_string_from_str() except only n characters from the given string are used.

fn_string_str() returns the contents of the given string str in the form of a null terminated string in the code set and current locale setting.

fn_string_from_contents() creates an FN_string_t object using the specified code set code_set, locale information locale_info and data in the given buffer contents. bytecount specifies the number of bytes in contents and charcount specifies the number of characters represented by contents.

fn_string_code_set() returns the code set associated with the given string object, and if present, the locale information in locale_info. fn_string_charcount() returns the number of characters in the given string object. fn_string_bytecount() returns the number of bytes used to represent the given string object. fn_string_contents() returns a pointer to the contents of the given string object.

fn_string_copy() returns a copy of the given string object. fn_string_assign() makes a copy of the string object src and assigns it to dst, releasing any old contents of dst. A pointer to the same object as dst is returned. fn_string_from_strings() is a function that takes a variable number of arguments (minimum of 2), the last of which must be NULL (0); it returns a new string object composed of the left to right concatenation of the given strings, in the given order. The support for strings with different code sets and/or locales as arguments to a single invocation of fn_string_from_strings() is implementation-dependent. fn_string_from_substring() returns a new string object consisting of the characters located between first and last inclusive from str. Indexing begins with 0. If last is FN_STRING_INDEX_LAST or exceeds the length of the string, the index of the last character of the string is used.

fn_string_is_empty() returns whether str is an empty string.

Comparison of two strings must take into account code set and locale information. If strings are in the same code set and same locale, case sensitivity is applied according to the case sensitivity rules applicable for the code set and locale; case sensitivity may not necessarily be relevant for all string encodings. If string_case is non-zero, case is significant and equality for strings of the same code set is defined as equality between
byte-wise encoded values of the strings. If `string_case` is zero, case is ignored and equality for strings of the same code set is defined using the definition of case-insensitive equality for the specific code set. Support for comparison between strings of different code sets, or lack thereof, is implementation-dependent.

`fn_string_compare()` compares strings `str1` and `str2` and returns 0 if they are equal, non-zero if they are not equal. If two strings are not equal, `fn_string_compare()` returns a positive value if the difference of `str2` precedes that of `str1` in terms of byte-wise encoded value (with case-sensitivity taken into account when `string_case` is non-zero), and a negative value if the difference of `str1` precedes that of `str2`, in terms of byte-wise encoded value (with case-sensitivity taken into account when `string_case` is non-zero). Such information (positive versus negative return value) may be used by applications that use strings of code sets in which ordering is meaningful; this information is not of general use in internationalized environments. `fn_string_compare_substring()` is similar to `fn_string_compare()` except `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.

**SEE ALSO**

`xfn(3N)`
NAME | Intro, intro – introduction to functions and libraries

DESCRIPTION | 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 \texttt{#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, \texttt{libucb.so}, and as an archive, \texttt{libucb.a}, but is not automatically linked by the C compilation system. Specify \texttt{−lucb} on the \texttt{cc} command line to link with this library, which is located in the \texttt{/usr/ucb} subdirectory. Header files for this library are located within \texttt{/usr/ucbinclude}.

(3C) These functions, together with those of Section 2 and those marked (3S), constitute the standard C library, \texttt{libc}, which is automatically linked by the C compilation system. The standard C library is implemented as a shared object, \texttt{libc.so}, and as an archive, \texttt{libc.a}. C programs are linked with the shared object version of the standard C library by default. Specify \texttt{−dn} on the \texttt{cc} command line to link with the archive version. (See \texttt{cc(1B)} for other overrides, and the “C Compilation System” chapter of the \texttt{ANSI C Programmer’s Guide} for a discussion.) Some functions behave differently in the XPG4 environment. This behavior is noted on the individual manual pages. See \texttt{xp4g(5)}.

(3E) These functions constitute the ELF access library, \texttt{libelf}, (Extensible Linking Formats). This library provides the interface for the creation and analyses of “elf” files; executables, objects, and shared objects. \texttt{libelf} is implemented as a shared object, \texttt{libelf.so}, and as an archive, \texttt{libelf.a}, but is not automatically linked by the C compilation system. Specify \texttt{−lelf} on the \texttt{cc} command line to link with this library.

(3G) These functions constitute the string pattern-matching \& pathname manipulation library, \texttt{libgen}. This library is implemented as an archive, \texttt{libgen.a}, but not as a shared object, and is not automatically linked by the C compilation system. Specify \texttt{−lgen} on the \texttt{cc} command line to link with this library.

(3I) These functions constitute the wide character libraries for multi-byte character support, and the international library for messaging. These libraries, \texttt{libintl}, and \texttt{libw}, are both implemented as shared objects, \texttt{libintl.so} and \texttt{libw.so}, and as archives, \texttt{libintl.a} and \texttt{libw.a}. However, they are not automatically linked by the C compilation system; specify \texttt{−lintl} or \texttt{−lw} on the \texttt{cc} command line, as needed to link the appropriate library.

(3K) These functions allow access to the kernel’s virtual memory library, which is implemented as a shared object, \texttt{libkvm.so}, and as an archive, \texttt{libkvm.a}, but is not automatically linked by the C compilation system. Specify \texttt{−lkvm} on the \texttt{cc} command line to link with this library.

modified 30 Jun 1995
(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.

(3N) These functions constitute the Network Service Library, **libnsl**. It is implemented as a shared object, **libnsl.so**, and as an archive, **libnsl.a**, but is not automatically linked by the C compilation system. Specify **−linsl** on the **cc** command line to link with this library.

Some of the functions documented in man3n incorporate other network libraries, including:
- **libsocket**
- **libresolv**
- **librpcsrv**
- **libnisdb**
- **librac**
- **libxfn**
- **libkrb**

(3R) These functions constitute the POSIX.4 Realtime library, **libposix4**. It is implemented only as a shared object, **libposix4.so**, and is not automatically linked by the C compilation system. Specify **−lposix4** on the **cc** command line to link with this library.

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

(3T) These functions constitute the threads libraries, **libpthread** and **libthread**. These libraries are used for building multithreaded applications. **libpthread** implements the POSIX 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-compliant 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).

**libpthread** and **libthread** are implemented as shared objects, **libpthread.so** and **libthread.so**, 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.

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 prioceiling);
int pthread_mutexattr_getprioceiling(const pthread_mutexattr_t *attr,
                                    int *prioceiling);
```

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

**DEFINITIONS**

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

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

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

Types of the form **size_t** are defined in the appropriate headers.

**MT-Level of Libraries**

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

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

**Safe**

Safe is simply an attribute of code that can be called from a multithreaded application. It is a generic term used to differentiate between code that is unsafe.
| **Unsafe** | An unsafe library contains global and static data that is not protected. It is not safe to use unless the application arranges for only one thread at a time to execute within the library. Unsafe libraries may contain routines that are safe; however, most of the library's routines are unsafe to call. |
| **MT-Safe** | An MT-Safe library is fully prepared for multithreaded access. It protects its global and static data with locks, and can provide a reasonable amount of concurrency. Note that a library can be safe to use, but not MT-Safe. For example, surrounding an entire library with a monitor makes the library safe, but it supports no concurrency so it is not considered MT-Safe. An MT-Safe library must permit a reasonable amount of concurrency. (This definition's purpose is to give precision to what is meant when a library is described as safe. The definition of a "safe" library does not specify if the library supports concurrency. The MT-Safe definition makes it clear that the library is safe, and supports some concurrency. This clarifies the safe definition, which can mean anything from being single threaded to being any degree of multithreaded.) |
| **Async-Signal-Safe** | Async-Signal-Safe refers to particular library routines that can be safely called from a signal handler. A thread that is executing an Async-Signal-Safe routine will not deadlock with itself if interrupted by a signal. Signals are only a problem for MT-Safe routines that acquire locks. Signals are disabled when locks are acquired in Async-Signal-Safe routines. This prevents a signal handler that might acquire the same lock from being called. The designation "Async-Safe" also indicates "Async-Signal-Safe." The list of "Async-Signal-Safe" functions includes: |
| _exit | kill | tcflow |
| access | link | tcflush |
| aio_error | lseek | tcgetattr |
| aio_return | mkdir | tcgetpgrp |
| aio_suspend | mkfifo | tcsendbreak |
| alarm | open | tcsetattr |
| cfgetispeed | pathconf | tcsetpgrp |
| cfgetospeed | pause | thr_kill |
| cfsetispeed | pipe | thr_sigsetmask |
| cfsetospeed | read | time |
| chdir | rename | timer_getoverrun |
| chmod | rmdir | timer_gettime |
| chown | sem_post | timer_settime |
| clock_gettime | sema_post | times |
| close | setgid | umask |
| creat | setpgid | uname |
| dup | setsid | unlink |

Modified 30 Jun 1995
<table>
<thead>
<tr>
<th>Unsafe Function</th>
<th>Reentrant counterpart</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctime</td>
<td>ctime_r</td>
</tr>
<tr>
<td>localtime</td>
<td>localtime_r</td>
</tr>
<tr>
<td>asctime</td>
<td>asctime_r</td>
</tr>
<tr>
<td>gmtime</td>
<td>gmtime_r</td>
</tr>
<tr>
<td>ctermid</td>
<td>ctermid_r</td>
</tr>
<tr>
<td>getlogin</td>
<td>getlogin_r</td>
</tr>
<tr>
<td>rand</td>
<td>rand_r</td>
</tr>
<tr>
<td>readdir</td>
<td>readdir_r</td>
</tr>
<tr>
<td>strtok</td>
<td>strtok_r</td>
</tr>
<tr>
<td>tmpnam</td>
<td>tmpnam_r</td>
</tr>
</tbody>
</table>

**Fork1-Safe**
A fork1-safe library releases the locks it had held whenever fork1(2) is called in a Solaris thread program, or fork(2) in a POSIX thread program. Calling fork(2) in a POSIX thread program has the same semantic as calling fork1(2) in a Solaris thread program. All system calls, libpthread, and libthread are Fork1-Safe. Otherwise, you should handle the locking clean-up yourself (see pthread_atfork(3T)).

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

**Reentrant functions for libc**

**Unsafe Function**

<table>
<thead>
<tr>
<th>Reentrant counterpart</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctime_r</td>
</tr>
<tr>
<td>localtime_r</td>
</tr>
<tr>
<td>asctime_r</td>
</tr>
<tr>
<td>gmtime_r</td>
</tr>
<tr>
<td>ctermid_r</td>
</tr>
<tr>
<td>getlogin_r</td>
</tr>
<tr>
<td>rand_r</td>
</tr>
<tr>
<td>readdir_r</td>
</tr>
<tr>
<td>strtok_r</td>
</tr>
<tr>
<td>tmpnam_r</td>
</tr>
</tbody>
</table>

**Cancel-Safety**
If a multi-threaded application uses pthread_cancel(3T) to cancel (i.e., kill a
thread), it is possible that the target thread is killed while holding a resource, such as a lock or allocated memory. If the thread has not installed the appropriate cancellation cleanup handlers to release the resources appropriately (see

`pthread_cancel(3T)`), the application is "cancel-unsafe", i.e., it is not safe with respect to cancellation. This unsafety could result in deadlocks due to locks not released by a thread that gets cancelled, or resource leaks; for example, memory not being freed on thread cancellation. All applications that use

`pthread_cancel(3T)` should ensure that they operate cancel-safe environment.

Libraries that have cancellation points and which acquire resources such as locks or allocate memory dynamically, also contribute to the cancel-unsafety of applications that are linked with these libraries. This introduces another level of safety for libraries in a multi-threaded program: cancel-safety.

There are two sub-categories of cancel-safety:

Deferred-cancel-safety and Asynchronous-cancel-safety.

An application is considered to be Deferred-cancel-safe when it is cancel-safe for threads whose cancellation type is PTHREAD_CANCEL_DEFERRED.

An application is considered to be Asynchronous-cancel-safe when it is cancel-safe for threads whose cancellation type is

`PTHREAD_CANCEL_ASYNCHRONOUS`.

Deferred-cancel-safety is easier to achieve than Asynchronous-cancel-safety, since a thread with the deferred cancellation type can be cancelled only at well-defined "cancellation points", whereas a thread with the asynchronous cancellation type can be cancelled anywhere. Since all threads are created by default to have the deferred cancellation type, it may never be necessary to worry about asynchronous cancel safety. Indeed, most applications and libraries are expected to always be Asynchronous-cancel-unsafe.

An application which is Asynchronous-cancel-safe is also, by definition, Deferred-cancel-safe.

**FILES**

<table>
<thead>
<tr>
<th>INCDIR</th>
<th>usually /usr/include</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIBDIR</td>
<td>usually /usr/ccs/lib</td>
</tr>
<tr>
<td>LIBDIR/libc.so</td>
<td></td>
</tr>
<tr>
<td>LIBDIR/libc.a</td>
<td></td>
</tr>
<tr>
<td>LIBDIR/libgen.a</td>
<td></td>
</tr>
<tr>
<td>LIBDIR/libm.a</td>
<td></td>
</tr>
<tr>
<td>LIBDIR/libsfm.sa</td>
<td></td>
</tr>
<tr>
<td>/usr/lib/libc.so.1</td>
<td></td>
</tr>
</tbody>
</table>

**SEE ALSO**

ar(1), cc(1B), ld(1), nm(1), fork(2), intro(2), stdio(3S), pthread_atfork(3T), xpg4(5)

modified 30 Jun 1995
**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 instead 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, the `_REENTRANT` flag must be defined on the compile line (`−D_REENTRANT`). This enables new definitions for functions only applicable to multithreaded applications.

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

When linking, it is a requirement that a multithreaded application be constructed to ensure that libthread physically interposes upon the C library. For example, the command line for linking the application using `ld` should be ordered as follows:

```
example% ld [options] .o's ... −lthread
```

Note that the behavior of the C library is undefined if `−lc` precedes `−lthread`. And, when linking with `cc`, `−lthread` must be last on the command line.

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.

**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 `−I` option to `lint`. (For example, `−Im` includes definitions for `libm`.) Use of `lint` is highly recommended. See the `lint` chapter in `Profiling Tools`.
Users should carefully note the difference between STREAMS and `stream`. STREAMS is a set of kernel mechanisms that support the development of network services and data communication drivers. It is composed of utility routines, kernel facilities, and a set of data structures. A `stream` is a file with its associated buffering. It is declared to be a pointer to a type `FILE` defined in `<stdio.h>`.

In detailed definitions of components, it is sometimes necessary to refer to symbolic names that are implementation-specific, but which are not necessarily expected to be accessible to an application program. Many of these symbolic names describe boundary conditions and system limits.

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

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a64l(3C)</td>
<td>convert between long integer and base-64 ASCII string</td>
</tr>
<tr>
<td>abort(3C)</td>
<td>terminate the process abnormally</td>
</tr>
<tr>
<td>abs(3C)</td>
<td>return absolute value of integer</td>
</tr>
<tr>
<td>accept(3N)</td>
<td>accept a connection on a socket</td>
</tr>
<tr>
<td>aclcheck(3)</td>
<td>check the validity of an ACL</td>
</tr>
<tr>
<td>aclfrommode(3)</td>
<td>See acltomode(3)</td>
</tr>
<tr>
<td>aclfrompbits(3)</td>
<td>See acltopbits(3)</td>
</tr>
<tr>
<td>aclfromtext(3)</td>
<td>See acltotext(3)</td>
</tr>
<tr>
<td>aclsort(3)</td>
<td>sort an ACL</td>
</tr>
<tr>
<td>acltopmode(3)</td>
<td>convert an ACL to/from permission bits</td>
</tr>
<tr>
<td>acltopbits(3)</td>
<td>convert an ACL to/from permission bits</td>
</tr>
<tr>
<td>acltotext(3)</td>
<td>convert an internal representation to/from external representation</td>
</tr>
<tr>
<td>acos(3M)</td>
<td>See trig(3M)</td>
</tr>
<tr>
<td>acosh(3M)</td>
<td>See hyperbolic(3M)</td>
</tr>
<tr>
<td>addch(3X)</td>
<td>See curs_addch(3X)</td>
</tr>
</tbody>
</table>

modified 30 Jun 1995
addchnstr(3X) See curs_addchstr(3X)
addchstr(3X) See curs_addchstr(3X)
addinstr(3X) See curs_addstr(3X)
addnwstr(3X) See curs_addwstr(3X)
adsev(3C) define additional severities
addseverity(3C) build a list of severity levels for an
addstr(3X) See curs_addstr(3X)
addwch(3X) See curs_addwch(3X)
addwchnstr(3X) See curs_addwchstr(3X)
addwchstr(3X) See curs_addwchstr(3X)
addwstr(3X) See curs_addwstr(3X)
adjcurpos(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)
aiofsync(3R) asynchronous file synchronization
aio_read(3) asynchronous I/O operations
aio_read(3R) asynchronous read and write opera-
aio_return(3R) retrieve return or error status of
tions asynchronous I/O operation
aio_suspend(3R) wait for asynchronous I/O request
aio_wait(3) wait for completion of asynchronous
aiowrite(3) See aio_read(3)
aio_write(3R) See aio_read(3)
alloca(3C) See malloc(3C)
alphasort(3B) See scandir(3B)
arc(3) See plot(3)
asctime(3C) See strftime(3C)
asctime(3C) See ctime(3C)
asctime_r(3C) See ctime(3C)
asin(3M) See trig(3M)
asinh(3M) See hyperbolic(3M)
assert(3C) verify program assertion
asysmem(3) See sysmem(3)
atan(3M) See trig(3M)
atan2(3M) See trig(3M)
atanh(3M) See hyperbolic(3M)
atexit(3C) add program termination routine
atof(3C) See strtod(3C)
atoi(3C) See strtol(3C)
atol(3C) See strtol(3C)
atoll(3C) See strtol(3C)
atroff(3X) See curs_attr(3X)
atron(3X) See curs_attr(3X)
atrset(3X) See curs_attr(3X)
au_close(3) See au_open(3)
au_open(3) construct and write audit records
au_preselect(3) preselect an audit event
authdes_create(3N) See rpc_soc(3N)
authdes_getucred(3N) See secure_rpc(3N)
authdes_seccreate(3N) See secure_rpc(3N)
auth_destroy(3N) See rpc_clnt_auth(3N)
authkerb_getucred(3N) See kerberos_rpc(3N)
authkerb_seccreate(3N) See kerberos_rpc(3N)
authnone_create(3N) See rpc_clnt_auth(3N)
authsys_create(3N) See rpc_clnt_auth(3N)
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(3G) return the last element of a path
baudrate(3X) See curs_termattrs(3X)
bcmp(3C) See bstring(3C)
bcopy(3C) See bstring(3C)
beep(3X) See curs_beep(3X)
bessel(3M) Bessel functions
bgets(3G) read stream up to next delimiter
bind(3N) bind a name to a socket
boundtextdomain(3I) See gettext(3I)
bkgd(3X) See curs_bkgd(3X)
bkgdset(3X) See curs_bkgd(3X)
border(3X) See curs_border(3X)
bottom_panel(3X) See panel_top(3X)
box(3) See plot(3)
box(3X) See curs_border(3X)
bsdmalloc(3X) memory allocator
bsearch(3C) binary search a sorted table
bstring(3C) bit and byte string operations
bufsplit(3G) split buffer into fields
byteorder(3N) convert values between host and network byte order
bzero(3C) See bstring(3C)
calloc(3C) See malloc(3C)
calloc(3X) See malloc(3X)
calloc(3X) See mapmalloc(3X)
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>See (Function)</th>
</tr>
</thead>
<tbody>
<tr>
<td>callrpc(3N)</td>
<td></td>
<td>rpc_soc(3N)</td>
</tr>
<tr>
<td>cancellation(3T)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>can_change_color(3X)</td>
<td></td>
<td>curs_color(3X)</td>
</tr>
<tr>
<td>catclose(3C)</td>
<td></td>
<td>catopen(3C)</td>
</tr>
<tr>
<td>catgets(3C)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>catopen(3C)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cbc_crypt(3)</td>
<td></td>
<td>des_crypt(3)</td>
</tr>
<tr>
<td>cbreak(3X)</td>
<td></td>
<td>curs_inopts(3X)</td>
</tr>
<tr>
<td>cbrt(3M)</td>
<td></td>
<td>sqrt(3M)</td>
</tr>
<tr>
<td>ceil(3M)</td>
<td></td>
<td>floor(3M)</td>
</tr>
<tr>
<td>cfgetispeed(3)</td>
<td></td>
<td>termios(3)</td>
</tr>
<tr>
<td>cfgetospeed(3)</td>
<td></td>
<td>termios(3)</td>
</tr>
<tr>
<td>cfsetispeed(3)</td>
<td></td>
<td>mapmalloc(3X)</td>
</tr>
<tr>
<td>cfsetospeed(3)</td>
<td></td>
<td>termios(3)</td>
</tr>
<tr>
<td>cftime(3C)</td>
<td></td>
<td>strftime(3C)</td>
</tr>
<tr>
<td>circle(3)</td>
<td></td>
<td>plot(3)</td>
</tr>
<tr>
<td>clear(3X)</td>
<td></td>
<td>curs_clear(3X)</td>
</tr>
<tr>
<td>clearerr(3S)</td>
<td></td>
<td>ferror(3S)</td>
</tr>
<tr>
<td>clearok(3X)</td>
<td></td>
<td>curs_outopts(3X)</td>
</tr>
<tr>
<td>clnt_broadcast(3N)</td>
<td></td>
<td>rpc_soc(3N)</td>
</tr>
<tr>
<td>clnt_call(3N)</td>
<td></td>
<td>rpc_clnt_calls(3N)</td>
</tr>
<tr>
<td>clnt_control(3N)</td>
<td></td>
<td>rpc_clnt_create(3N)</td>
</tr>
<tr>
<td>clnt_create(3N)</td>
<td></td>
<td>rpc_clnt_create(3N)</td>
</tr>
<tr>
<td>clnt_create_timed(3N)</td>
<td></td>
<td>rpc_clnt_create(3N)</td>
</tr>
<tr>
<td>clnt_create_vers(3N)</td>
<td></td>
<td>rpc_clnt_create(3N)</td>
</tr>
<tr>
<td>clnt_destroy(3N)</td>
<td></td>
<td>rpc_clnt_create(3N)</td>
</tr>
<tr>
<td>clnt dg_create(3N)</td>
<td></td>
<td>rpc_clnt_create(3N)</td>
</tr>
<tr>
<td>clnt_freeres(3N)</td>
<td></td>
<td>rpc_clnt_calls(3N)</td>
</tr>
<tr>
<td>clnt_geterr(3N)</td>
<td></td>
<td>rpc_clnt_calls(3N)</td>
</tr>
<tr>
<td>clnt_pcreateerror(3N)</td>
<td></td>
<td>rpc_clnt_create(3N)</td>
</tr>
<tr>
<td>clnt_errno(3N)</td>
<td></td>
<td>rpc_clnt_create(3N)</td>
</tr>
<tr>
<td>clnt_errno(3N)</td>
<td></td>
<td>rpc_clnt_calls(3N)</td>
</tr>
<tr>
<td>clnt raw_create(3N)</td>
<td></td>
<td>rpc_clnt_create(3N)</td>
</tr>
<tr>
<td>clntraw_create(3N)</td>
<td></td>
<td>rpc_soc(3N)</td>
</tr>
</tbody>
</table>

modified 30 Jun 1995
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>clnt_spcreateerror(3N)</code></td>
<td>See <code>rpc_clnt_create(3N)</code></td>
</tr>
<tr>
<td><code>clnt_sperrno(3N)</code></td>
<td>See <code>rpc_clnt_calls(3N)</code></td>
</tr>
<tr>
<td><code>clnt_serror(3N)</code></td>
<td>See <code>rpc_clnt_create(3N)</code></td>
</tr>
<tr>
<td><code>clnttcp_create(3N)</code></td>
<td>See <code>rpc_soc(3N)</code></td>
</tr>
<tr>
<td><code>clnt_tli_create(3N)</code></td>
<td>See <code>rpc_clnt_create(3N)</code></td>
</tr>
<tr>
<td><code>clnt_tp_create(3N)</code></td>
<td>See <code>rpc_clnt_create(3N)</code></td>
</tr>
<tr>
<td><code>clnt_tp_create_timed(3N)</code></td>
<td>See <code>rpc_clnt_create(3N)</code></td>
</tr>
<tr>
<td><code>clntudp_bufcreate(3N)</code></td>
<td>See <code>rpc_soc(3N)</code></td>
</tr>
<tr>
<td><code>clntudp_create(3N)</code></td>
<td>See <code>rpc_soc(3N)</code></td>
</tr>
<tr>
<td><code>clnt_vc_create(3N)</code></td>
<td>See <code>rpc_clnt_create(3N)</code></td>
</tr>
<tr>
<td><code>clock(3C)</code></td>
<td>report CPU time used</td>
</tr>
<tr>
<td><code>clock_getres(3R)</code></td>
<td>See <code>clock_settime(3R)</code></td>
</tr>
<tr>
<td><code>clock_gettime(3R)</code></td>
<td>See <code>clock_settime(3R)</code></td>
</tr>
<tr>
<td><code>clock_settime(3R)</code></td>
<td>high-resolution clock operations</td>
</tr>
<tr>
<td><code>closedir(3C)</code></td>
<td>See <code>directory(3C)</code></td>
</tr>
<tr>
<td><code>closelog(3)</code></td>
<td>See <code>syslog(3)</code></td>
</tr>
<tr>
<td><code>closepl(3)</code></td>
<td>See <code>plot(3)</code></td>
</tr>
<tr>
<td><code>closevt(3)</code></td>
<td>See <code>plot(3)</code></td>
</tr>
<tr>
<td><code>clrtobot(3X)</code></td>
<td>See <code>curs_clear(3X)</code></td>
</tr>
<tr>
<td><code>clrtoeol(3X)</code></td>
<td>See <code>curs_clear(3X)</code></td>
</tr>
<tr>
<td><code>color_content(3X)</code></td>
<td>See <code>curs_color(3X)</code></td>
</tr>
<tr>
<td><code>compile(3G)</code></td>
<td>See <code>reexpr(3G)</code></td>
</tr>
<tr>
<td><code>cond_broadcast(3T)</code></td>
<td>See <code>condition(3T)</code></td>
</tr>
<tr>
<td><code>cond_destroy(3T)</code></td>
<td>See <code>condition(3T)</code></td>
</tr>
<tr>
<td><code>cond_init(3T)</code></td>
<td>See <code>condition(3T)</code></td>
</tr>
<tr>
<td><code>condition(3T)</code></td>
<td>condition variables</td>
</tr>
<tr>
<td><code>cond_signal(3T)</code></td>
<td>See <code>condition(3T)</code></td>
</tr>
<tr>
<td><code>cond_timedwait(3T)</code></td>
<td>See <code>condition(3T)</code></td>
</tr>
<tr>
<td><code>cond_wait(3T)</code></td>
<td>See <code>condition(3T)</code></td>
</tr>
<tr>
<td><code>confstr(3C)</code></td>
<td>get configurable variables</td>
</tr>
<tr>
<td><code>connect(3N)</code></td>
<td>initiate a connection on a socket</td>
</tr>
<tr>
<td><code>cont(3)</code></td>
<td>See <code>plot(3)</code></td>
</tr>
<tr>
<td><code>conv(3C)</code></td>
<td>translate characters</td>
</tr>
<tr>
<td><code>copylist(3G)</code></td>
<td>copy a file into memory</td>
</tr>
<tr>
<td><code>copsign(3M)</code></td>
<td>See <code>ieee_functions(3M)</code></td>
</tr>
</tbody>
</table>
copywin(3X) See curs_overlay(3X)
cos(3M) See trig(3M)
cosh(3M) See hyperbolic(3M)
crypt(3C) generate encryption
cset(3I) get information on EUC codesets
csetcol(3I) See cset(3I)
csetlen(3I) See cset(3I)
csetno(3I) See cset(3I)
ctermid(3S) generate path name for controlling
terminal
ctermid_r(3S) See ctermid(3S)
ctime(3C) convert date and time to string
ctime_r(3C) See ctime(3C)
cctype(3C) character handling
current_field(3X) See menu_item_current(3X)
current_item(3X) add a character (with attributes) to a
curses window and advance cursor
curs_addch(3X) add string of characters (and attributes) to a curses window
curs_addchstr(3X) add a string of characters to a curses window and advance cursor
curs_addstr(3X)
curs_addwch(3X) add a wchar_t character (with attributes) to a curses window and
advance cursor
curs_addwchstr(3X) add string of wchar_t characters (and attributes) to a curses window
curs_addwstr(3X) add a string of wchar_t characters to a curses window and advance cursor
curs_aletcompat(3X) these functions are added to ALE
curses library for moving the cursor by character.
curs_attr(3X) curses character and window attribute control routines
curs_beep(3X) curses bell and screen flash routines
curs_bkgd(3X) curses window background manipulation routines
curs_border(3X) create curses borders, horizontal and
curs_clear(3X) clear all or part of a curses window

curs_color(3X) curses color manipulation routines

curs_delch(3X) delete character under cursor in a curses window

curs_deletein(3X) delete and insert lines in a curses window

curses(3X) CRT screen handling and optimization package

curs_getch(3X) get (or push back) characters from curses terminal keyboard

curs_getstr(3X) get character strings from curses terminal keyboard

curs_getwch(3X) get (or push back) wchar_t characters from curses terminal keyboard

curs_getwstr(3X) get wchar_t character strings from curses terminal keyboard

curs_getyx(3X) get curses cursor and window coordinates

curs_inch(3X) get a character and its attributes from a curses window

curs_inchstr(3X) get a string of characters (and attributes) from a curses window

curs_initscr(3X) curses screen initialization and manipulation routines

curs_inopts(3X) curses terminal input option control routines

curs_insch(3X) insert a character before the character under the cursor in a curses window

curs_inssstr(3X) insert string before character under the cursor in a curses window

curs_instr(3X) get a string of characters from a curses window

curs_inswhch(3X) insert a wchar_t character before the character under the cursor in a curses window

curs_inswwstr(3X) insert wchar_t string before character under the cursor in a curses window

curs_inwch(3X) get a wchar_t character and its
attributes from a curses window
get a string of wchar_t characters
(and attributes) from a curses window
get a string of wchar_t characters
from a curses window
low-level curses routines
move curses window cursor
curses terminal output option control
routines
overlap and manipulate overlapped
curses windows
create and display curses pads
print formatted output in curses windows
refresh curses windows and lines
convert formatted input from a
curses widow
read (write) a curses screen from (to)
a file
scroll a curses window
See curs_KERNEL(3X)
curses soft label routines
curses environment query routines
curses interfaces (emulated) to the
termcap library
curses interfaces to terminfo data-
base
curses refresh control routines
curses miscellaneous utility routines
create curses windows
get character login name of the user
See form_data(3X)
See form_data(3X)
See nis_db(3N)
See nis_db(3N)
See nis_db(3N)
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>db_destroy_table(3N)</td>
<td>See nis_db(3N)</td>
</tr>
<tr>
<td>db_first_entry(3N)</td>
<td>See nis_db(3N)</td>
</tr>
<tr>
<td>db_free_result(3N)</td>
<td>See nis_db(3N)</td>
</tr>
<tr>
<td>db_initialize(3N)</td>
<td>See nis_db(3N)</td>
</tr>
<tr>
<td>db_list_entries(3N)</td>
<td>See nis_db(3N)</td>
</tr>
<tr>
<td>dbm(3B)</td>
<td>Data base subroutines</td>
</tr>
<tr>
<td>dbm_clearerr(3)</td>
<td>See ndbm(3)</td>
</tr>
<tr>
<td>dbm_close(3)</td>
<td>See ndbm(3)</td>
</tr>
<tr>
<td>dbmclose(3B)</td>
<td>See dbm(3B)</td>
</tr>
<tr>
<td>dbm_delete(3)</td>
<td>See ndbm(3)</td>
</tr>
<tr>
<td>dbm_error(3)</td>
<td>See ndbm(3)</td>
</tr>
<tr>
<td>dbm_fetch(3)</td>
<td>See ndbm(3)</td>
</tr>
<tr>
<td>dbm_firstkey(3)</td>
<td>See ndbm(3)</td>
</tr>
<tr>
<td>dbminit(3B)</td>
<td>See dbm(3B)</td>
</tr>
<tr>
<td>dbm_nextkey(3)</td>
<td>See ndbm(3)</td>
</tr>
<tr>
<td>dbm_open(3)</td>
<td>See ndbm(3)</td>
</tr>
<tr>
<td>dbm_store(3)</td>
<td>See ndbm(3)</td>
</tr>
<tr>
<td>db_next_entry(3N)</td>
<td>See nis_db(3N)</td>
</tr>
<tr>
<td>db_remove_entry(3N)</td>
<td>See nis_db(3N)</td>
</tr>
<tr>
<td>db_reset_next_entry(3N)</td>
<td>See nis_db(3N)</td>
</tr>
<tr>
<td>db_standby(3N)</td>
<td>See nis_db(3N)</td>
</tr>
<tr>
<td>db_table_exists(3N)</td>
<td>See nis_db(3N)</td>
</tr>
<tr>
<td>db_unload_table(3N)</td>
<td>See nis_db(3N)</td>
</tr>
<tr>
<td>dcgettext(3I)</td>
<td>See gettext(3I)</td>
</tr>
<tr>
<td>decimal_to_double(3)</td>
<td>See decimal_to_float(3)</td>
</tr>
<tr>
<td>decimal_to_extended(3)</td>
<td>See decimal_to_float(3)</td>
</tr>
<tr>
<td>decimal_to_float(3)</td>
<td>Convert decimal record to floating-point value</td>
</tr>
<tr>
<td>decimal_to_quadruple(3)</td>
<td>See decimal_to_float(3)</td>
</tr>
<tr>
<td>decimal_to_single(3)</td>
<td>See decimal_to_float(3)</td>
</tr>
<tr>
<td>def_prog_mode(3X)</td>
<td>See curs_kernel(3X)</td>
</tr>
<tr>
<td>def_shell_mode(3X)</td>
<td>See curs_kernel(3X)</td>
</tr>
<tr>
<td>delay_output(3X)</td>
<td>See curs_util(3X)</td>
</tr>
<tr>
<td>delch(3X)</td>
<td>See curs_delch(3X)</td>
</tr>
<tr>
<td>del_curterm(3X)</td>
<td>See curs_terminfo(3X)</td>
</tr>
</tbody>
</table>

Modified 30 Jun 1995
delete(3B)
deleteln(3X)
del_panel(3X)
delscreen(3X)
delwin(3X)
demangle(3)
derwin(3X)
des_crypt(3)
DES_FAILED(3)
des_failed(3)
des_setparity(3)
dgettext(3I)
dial(3N)
difftime(3C)
directory(3C)
dirname(3G)
div(3C)
dladdr(3X)
dlclose(3X)
dlerror(3X)
dlopen(3X)
dlsym(3X)
dn_comp(3N)
dn_expand(3N)
doconfig(3N)
double_to_decimal(3)
doupdate(3X)
drand48(3C)
dup2(3C)
dup_field(3X)

described 30 Jun 1995 3-53
dupwin(3X)  
dynamic_field_info(3X)  
ecb_crypt(3)  
echo(3X)  
echochar(3X)  
echowchar(3X)  
econvert(3)  
ecvt(3)  
ecvt(3C)  
_edata(3C)  
edata(3C)  
elf(3E)  
elf32_fsize(3E)  
elf32_getehdr(3E)  
elf32_getphdr(3E)  
elf32_getshdr(3E)  
elf32_newehdr(3E)  
elf32_newphdr(3E)  
elf32_xlatetof(3E)  
elf32_xlatetom(3E)  
elf_begin(3E)  
elf_cntl(3E)  
elf_end(3E)  
elf_errmsg(3E)  
elf_errno(3E)  
elf_fill(3E)  
elf_flagdata(3E)  
elf_flagehdr(3E)  
elf_flagelf(3E)  
elf_flagphdr(3E)  
elf_flagscn(3E)  
elf_flagshdr(3E)  

See curs_window(3X)  
See form_field_info(3X)  
See des_crypt(3)  
See curs_inopts(3X)  
See curs_addch(3X)  
See curs_addwch(3X)  
See econvert(3)  
convert floating-point number to string  
See end(3C)  
See end(3C)  
oobject file access library  
return the size of an object file type  
retrieve class-dependent object file header  
retrieve class-dependent program header table  
retrieve class-dependent section header  
See elf32_getehdr(3E)  
See elf32_getphdr(3E)  
class-dependent data translation  
See elf32_xlatetof(3E)  
process ELF object files  
control an elf file descriptor  
See elf_begin(3E)  
error handling  
See elf_errno(3E)  
set fill byte  
manipulate flags  
See elf_flagdata(3E)  
See elf_flagdata(3E)  
See elf_flagdata(3E)  
See elf_flagdata(3E)  
See elf_flagdata(3E)  
See elf_flagdata(3E)  
modified 30 Jun 1995
elf_getarhdr(3E) retrieve archive member header
elf_getarsym(3E) retrieve archive symbol table
elf_getbase(3E) get the base offset for an object file
elf_getdata(3E) get section data
elf_getident(3E) retrieve file identification data
elf_getscn(3E) get section information
elf_hash(3E) compute hash value
elf_kind(3E) determine file type
elf_memory(3E) See elf_begin(3E)
elf_ndxscn(3E) See elf_getscn(3E)
elf_newdata(3E) See elf_getdata(3E)
elf_newscn(3E) See elf_getscn(3E)
elf_next(3E) See elf_begin(3E)
elf_nextscn(3E) See elf_getscn(3E)
elf_rand(3E) See elf_begin(3E)
elf_rawdata(3E) retrieve uninterpreted file contents
elf_rawfile(3E) make a string pointer
elf_strptr(3E) update an ELF descriptor
elf_update(3E) coordinate ELF library and application versions
elf_version(3E) See crypt(3C)
encrypt(3C) last locations in program
end(3C) See end(3C)
_end(3C) See getacinfo(3)
endac(3) See getaclassent(3)
endauclass(3) See getaevent(3)
endauevent(3) See getauclassent(3)
endauevent(3) See getauusernam(3)
endauuser(3) See getgrnam(3C)
endgrent(3C) See gethostbyname(3N)
endhostent(3N) See getnetconfig(3N)
endhostent(3N) See gethostbyname(3N)
endnetconfig(3N) See getnetconfig(3N)
endnetent(3N) See getnetbyname(3N)
endnetgrent(3N) See getnetgrent(3N)
endnetpath(3N) See getnetpath(3N)
endprotoent(3N) See getprotobynam(3N)
endpwent(3C) See getpwnam(3C)
endrpcent(3N) See getrpcbyname(3N)
endservent(3N) See getservbyname(3N)
endspent(3C) See getspnam(3C)
endusershell(3C) See getusershell(3C)
endutent(3C) See getutent(3C)
endutxent(3C) See getutxent(3C)
endwin(3X) See curs_initscr(3X)
erand48(3C) See drand48(3C)
erase(3) See plot(3)
erase(3X) See curs_clear(3X)
erasechar(3X) See curs_termattrs(3X)
erfc(3M) error functions
erf(3M) See erf(3M)
erro(3C) See perror(3C)
_etext(3C) See end(3C)
etext(3C) See end(3C)
ether_aton(3N) See ethers(3N)
ether_hostton(3N) See ethers(3N)
ether_line(3N) See ethers(3N)
ether_ntoa(3N) See ethers(3N)
ether_ntohost(3N) See ethers(3N)
eters(3N) Ethernet address mapping operations
euccol(3I) get byte length and display width of EUC characters
euclen(3I) See euclen(3I)
eucscol(3I) terminate process
exit(3C) See exp(3M) exponential, logarithm, power
exp(3M) See exp(3M)
expm1(3M) See expm1(3M)
extended_to_decimal(3) See floating_to_decimal(3)
fabs(3M) See ieee_functions(3M)
fattach(3C) attach a STREAMS-based file descriptor to an object in the file system name space

3-56 modified 30 Jun 1995
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fclose(3S)</td>
<td>Close or flush a stream</td>
</tr>
<tr>
<td>fconvert(3)</td>
<td>See econvert(3)</td>
</tr>
<tr>
<td>fcvt(3)</td>
<td>See econvert(3)</td>
</tr>
<tr>
<td>fcvt(3C)</td>
<td>See ecvt(3C)</td>
</tr>
<tr>
<td>fdatasync(3R)</td>
<td>Synchronize a file's data</td>
</tr>
<tr>
<td>fdetach(3C)</td>
<td>Detach a name from a STREAMS-based file descriptor</td>
</tr>
<tr>
<td>fdopen(3S)</td>
<td>See fopen(3S)</td>
</tr>
<tr>
<td>feof(3S)</td>
<td>See ferror(3S)</td>
</tr>
<tr>
<td>ferror(3S)</td>
<td>Stream status inquiries</td>
</tr>
<tr>
<td>fetch(3B)</td>
<td>See dbm(3B)</td>
</tr>
<tr>
<td>fflush(3S)</td>
<td>See fclose(3S)</td>
</tr>
<tr>
<td>ffs(3C)</td>
<td>Find first set bit</td>
</tr>
<tr>
<td>fgetc(3S)</td>
<td>See getc(3S)</td>
</tr>
<tr>
<td>fgetgrent(3C)</td>
<td>See getgrent(3C)</td>
</tr>
<tr>
<td>fgetgrent_r(3C)</td>
<td>See getgrent(3C)</td>
</tr>
<tr>
<td>fgetpos(3C)</td>
<td>See fsetpos(3C)</td>
</tr>
<tr>
<td>fgetpwent(3C)</td>
<td>See getpwent(3C)</td>
</tr>
<tr>
<td>fgetpwent_r(3C)</td>
<td>See getpwent(3C)</td>
</tr>
<tr>
<td>fgets(3S)</td>
<td>See gets(3S)</td>
</tr>
<tr>
<td>fgetsnext(3C)</td>
<td>See getspnam(3C)</td>
</tr>
<tr>
<td>fgetsnext_r(3C)</td>
<td>See getspnam(3C)</td>
</tr>
<tr>
<td>fgetwc(3I)</td>
<td>See getwc(3I)</td>
</tr>
<tr>
<td>fgetws(3I)</td>
<td>See getws(3I)</td>
</tr>
<tr>
<td>field_arg(3X)</td>
<td>See form_field_validation(3X)</td>
</tr>
<tr>
<td>field_back(3X)</td>
<td>See form_field_attributes(3X)</td>
</tr>
<tr>
<td>field_buffer(3X)</td>
<td>See form_field_buffer(3X)</td>
</tr>
<tr>
<td>field_count(3X)</td>
<td>See form_field(3X)</td>
</tr>
<tr>
<td>field_fore(3X)</td>
<td>See form_field_attributes(3X)</td>
</tr>
<tr>
<td>field_index(3X)</td>
<td>See form_page(3X)</td>
</tr>
<tr>
<td>field_info(3X)</td>
<td>See form_field_info(3X)</td>
</tr>
<tr>
<td>field_init(3X)</td>
<td>See form_hook(3X)</td>
</tr>
<tr>
<td>field_just(3X)</td>
<td>See form_field_just(3X)</td>
</tr>
<tr>
<td>field_opts(3X)</td>
<td>See form_field_opts(3X)</td>
</tr>
<tr>
<td>field_opts_off(3X)</td>
<td>See form_field_opts(3X)</td>
</tr>
</tbody>
</table>
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)
flockfile(3S) acquire and release stream lock
floor(3M) round to integral value in floating-point format
flushinp(3X) See curs_util(3X)
fmod(3M) See ieee_functions(3M)
fmtmsg(3C) display a message on stderr or system console
fn_attr_get(3N) return specified attribute associated with name
fn_attr_get_ids(3N) get a list of the identifiers of all attributes associated with named object
fn_attr_get_values(3N) return values of an attribute
fn_attribute_add(3N) See FN_attribute_t(3N)
fn_attribute_assign(3N) See FN_attribute_t(3N)
fn_attribute_copy(3N) See FN_attribute_t(3N)
fn_attribute_create(3N) See FN_attribute_t(3N)
fn_attribute_destroy(3N) See FN_attribute_t(3N)
fn_attribute_first(3N) See FN_attribute_t(3N)
fn_attribute_identifier(3N) See FN_attribute_t(3N)
fn_attribute_next(3N) See FN_attribute_t(3N)
fn_attribute_remove(3N)
fn_attribute_syntax(3N)
FN_attribute_t(3N)
fn_attribute_valuecount(3N)
fn_attr_modify(3N)

fn_attribute_remove(3N)
fn_attribute_syntax(3N)
FN_attribute_t(3N)
fn_attribute_valuecount(3N)
fn_attr_modify(3N)

fn_attrmodlist_add(3N)
fn_attrmodlist_assign(3N)
fn_attrmodlist_copy(3N)
fn_attrmodlist_count(3N)
fn_attrmodlist_create(3N)
fn_attrmodlist_destroy(3N)
fn_attrmodlist_first(3N)
fn_attrmodlist_next(3N)
FN_attrmodlist_t(3N)
fn_attr_multi_get(3N)

fn_attrmodlist_add(3N)
fn_attrmodlist_assign(3N)
fn_attrmodlist_copy(3N)
fn_attrmodlist_count(3N)
fn_attrmodlist_create(3N)
fn_attrmodlist_destroy(3N)
fn_attrmodlist_first(3N)
fn_attrmodlist_next(3N)
FN_attrmodlist_t(3N)
fn_attr_multi_get(3N)

fn_attr_multi_modify(3N)
fn_attrset_add(3N)
fn_attrset_assign(3N)
fn_attrset_copy(3N)
fn_attrset_count(3N)
fn_attrset_create(3N)
fn_attrset_destroy(3N)
fn_attrset_first(3N)
fn_attrset_get(3N)
fn_attrset_next(3N)
fn_attrset_remove(3N)
FN_attrset_t(3N)
FN_attrvalue_t(3N)
FN_bindinglist_destroy(3N)
fn_bindinglist_destroy(3N)
fn_bindinglist_next(3N)
FN_bindinglist_t(3N)
fn_composite_name_append_comp(3N)
fn_composite_name_append_name(3N)

fn_composite_name_append_comp(3N)
fn_composite_name_append_name(3N)

See FN_attribute_t(3N)
See FN_attribute_t(3N)
an XFN attribute
See FN_attribute_t(3N)
modify specified attribute associated with name
See FN_attrmodlist_t(3N)
See FN_attrmodlist_t(3N)
See FN_attrmodlist_t(3N)
See FN_attrmodlist_t(3N)
See FN_attrmodlist_t(3N)
See FN_attrmodlist_t(3N)
See FN_attrmodlist_t(3N)
See FN_attrmodlist_t(3N)
See FN_attrmodlist_t(3N)
See FN_attrmodlist_t(3N)
See FN_attrmodlist_t(3N)
See FN_attrmodlist_t(3N)
See FN_attrmodlist_t(3N)
See FN_attrmodlist_t(3N)
See FN_attrmodlist_t(3N)
See FN_attrmodlist_t(3N)
See FN_attrmodlist_t(3N)
See FN_attrmodlist_t(3N)
See FN_attrmodlist_t(3N)
See FN_attrmodlist_t(3N)
See FN_attrmodlist_t(3N)
See FN_attrset_t(3N)
See FN_attrset_t(3N)
See FN_attrset_t(3N)
See FN_attrset_t(3N)
See FN_attrset_t(3N)
See FN_attrset_t(3N)
See FN_attrset_t(3N)
See FN_attrset_t(3N)
See FN_attrset_t(3N)
a set of XFN attributes
an XFN attribute value
See fn_ctx_list_bindings(3N)
See fn_ctx_list_bindings(3N)
See fn_ctx_list_bindings(3N)
See FN_composite_name_t(3N)
See FN_composite_name_t(3N)

modified 30 Jun 1995
fn_composite_name_assign
fn_composite_name_copy
fn_composite_name_count
fn_composite_name_create
fn_composite_name_destroy
fn_composite_name_first
fn_composite_name_from_string
fn_composite_name_get_prefix
fn_composite_name_insert_comp
fn_composite_name_insert_name
fn_composite_name_is_empty
fn_composite_name_is_equal
fn_composite_name_is_prefix
fn_composite_name_is_sufﬁx
fn_composite_name_last
fn_composite_name_next
fn_composite_name_prepend_comp
fn_composite_name_prepend_name
fn_composite_name_prev
fn_composite_name_sufﬁx
FN_composite_name_t

fn_compound_name_append_comp
fn_compound_name_assign
fn_compound_name_copy
fn_compound_name_count
fn_compound_name_delete_all
fn_compound_name_delete_comp
fn_compound_name_destroy
fn_compound_name_first
fn_compound_name_from_syntax_atts
fn_compound_name_get_syntax_atts
fn_compound_name_insert_comp
fn_compound_name_is_empty

a sequence of component names spanning multiple naming systems

See FN_composite_name_t
See FN_composite_name_t
See FN_composite_name_t
See FN_composite_name_t
See FN_composite_name_t
See FN_composite_name_t
See FN_composite_name_t
See FN_composite_name_t
See FN_composite_name_t
See FN_composite_name_t
See FN_composite_name_t
See FN_composite_name_t
See FN_composite_name_t
See FN_composite_name_t
See FN_composite_name_t
See FN_composite_name_t
See FN_composite_name_t
See FN_composite_name_t
See FN_composite_name_t
See FN_composite_name_t
See FN_composite_name_t
See FN_composite_name_t
See FN_composite_name_t
See FN_composite_name_t
See FN_composite_name_t
See FN_composite_name_t
See FN_composite_name_t
See FN_composite_name_t
See FN_composite_name_t
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_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_t(3N)
fn_ctx_unbind(3N)
FN_identifier_t(3N)
fnmatch(3C)
fn_multigetlist_destroy(3N)
fn_multigetlist_next(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)
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

return a context’s reference
return syntax attributes associated
with named context
release storage associated with con-
text handle
return a handle to the Initial Context
construct a handle to a context object
using the given reference
list the atomic names and references
bound in a context
list the atomic names bound in a con-
text
look up name in context
look up the link reference bound to a
name
rename the name of a binding
an XFN context
unbind a name from a context
an XFN identifier
match filename or path name
See fn_attr_multi_get(3N)
See fn_attr_multi_get(3N)

modified 30 Jun 1995 3-61
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FN_multigetlist_t(3N)</td>
<td>See fn_attr_multi_get(3N)</td>
</tr>
<tr>
<td>fn_namelist_destroy(3N)</td>
<td>See fn_ctx_list_names(3N)</td>
</tr>
<tr>
<td>fn_namelist_next(3N)</td>
<td>See fn_ctx_list_names(3N)</td>
</tr>
<tr>
<td>FN_namelist_t(3N)</td>
<td>See fn_ctx_list_names(3N)</td>
</tr>
<tr>
<td>fn_ref_addr_assign(3N)</td>
<td>See FN_ref_addr_t(3N)</td>
</tr>
<tr>
<td>fn_ref_addr_copy(3N)</td>
<td>See FN_ref_addr_t(3N)</td>
</tr>
<tr>
<td>fn_ref_addrcount(3N)</td>
<td>See FN_ref_addr_t(3N)</td>
</tr>
<tr>
<td>fn_ref_addr_create(3N)</td>
<td>See FN_ref_addr_t(3N)</td>
</tr>
<tr>
<td>fn_ref_addr_data(3N)</td>
<td>See FN_ref_addr_t(3N)</td>
</tr>
<tr>
<td>fn_ref_addr_description(3N)</td>
<td>See FN_ref_addr_t(3N)</td>
</tr>
<tr>
<td>fn_ref_addr_destroy(3N)</td>
<td>See FN_ref_addr_t(3N)</td>
</tr>
<tr>
<td>fn_ref_addr_length(3N)</td>
<td>See FN_ref_addr_t(3N)</td>
</tr>
<tr>
<td>FN_ref_addr_t(3N)</td>
<td>an address in an XFN reference</td>
</tr>
<tr>
<td>fn_ref_addr_type(3N)</td>
<td>See FN_ref_addr_t(3N)</td>
</tr>
<tr>
<td>fn_ref_append_addr(3N)</td>
<td>See FN_ref_t(3N)</td>
</tr>
<tr>
<td>fn_ref_assign(3N)</td>
<td>See FN_ref_t(3N)</td>
</tr>
<tr>
<td>fn_ref_copy(3N)</td>
<td>See FN_ref_t(3N)</td>
</tr>
<tr>
<td>fn_ref_create(3N)</td>
<td>See FN_ref_t(3N)</td>
</tr>
<tr>
<td>fn_ref_create_link(3N)</td>
<td>See FN_ref_t(3N)</td>
</tr>
<tr>
<td>fn_ref_delete_addr(3N)</td>
<td>See FN_ref_t(3N)</td>
</tr>
<tr>
<td>fn_ref_delete_all(3N)</td>
<td>See FN_ref_t(3N)</td>
</tr>
<tr>
<td>fn_ref_description(3N)</td>
<td>See FN_ref_t(3N)</td>
</tr>
<tr>
<td>fn_ref_destroy(3N)</td>
<td>See FN_ref_t(3N)</td>
</tr>
<tr>
<td>fn_ref_first(3N)</td>
<td>See FN_ref_t(3N)</td>
</tr>
<tr>
<td>fn_ref_insert_addr(3N)</td>
<td>See FN_ref_t(3N)</td>
</tr>
<tr>
<td>fn_ref_is_link(3N)</td>
<td>See FN_ref_t(3N)</td>
</tr>
<tr>
<td>fn_ref_link_name(3N)</td>
<td>See FN_ref_t(3N)</td>
</tr>
<tr>
<td>fn_ref_next(3N)</td>
<td>See FN_ref_t(3N)</td>
</tr>
<tr>
<td>fn_ref_prepend_addr(3N)</td>
<td>See FN_ref_t(3N)</td>
</tr>
<tr>
<td>FN_ref_t(3N)</td>
<td>an XFN reference</td>
</tr>
<tr>
<td>fn_ref_type(3N)</td>
<td>See FN_ref_t(3N)</td>
</tr>
<tr>
<td>fn_status_advance_by_name(3N)</td>
<td>See FN_status_t(3N)</td>
</tr>
<tr>
<td>fn_status_append_remaining_name(3N)</td>
<td>See FN_status_t(3N)</td>
</tr>
<tr>
<td>fn_status_append_resolved_name(3N)</td>
<td>See FN_status_t(3N)</td>
</tr>
<tr>
<td>fn_status_assign(3N)</td>
<td>See FN_status_t(3N)</td>
</tr>
</tbody>
</table>

3-62 modified 30 Jun 1995
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)
fn_string_create(3N)
fn_string_destroy(3N)
fn_string_from_composite_name(3N)
fn_string_from_compound_name(3N)
fn_string_from_contents(3N)
fn_string_from_str(3N)
fn_string_from_strings(3N)
fn_string_from_substring(3N)
fn_string_is_empty(3N)
fn_string_next_substring(3N)
fn_string_prev_substring(3N)
fn_string_str(3N)
FN_string_t(3N)
fn_valuelist_destroy(3N)
fn_valuelist_next(3N)
FN_valuelist_t(3N)
fopen(3B)
fopen(3S)
form_cursor(3X)
form_data(3X)
form_driver(3X)
form_field(3X)
form_field_attributes(3X)
form_field_buffer(3X)
form_field_info(3X)
form_field_just(3X)
form_field_new(3X)
form_field_opts(3X)
form_fields(3X)
form_fieldtype(3X)

See FN_string_t(3N)
See FN_string_t(3N)
See FN_string_t(3N)
See FN_string_t(3N)
See FN_string_t(3N)
See FN_string_t(3N)
See FN_string_t(3N)
See FN_string_t(3N)
See FN_string_t(3N)
See FN_string_t(3N)
See FN_string_t(3N)
See FN_string_t(3N)
See fn_attr_get_values(3N)
See fn_attr_get_values(3N)
See fn_attr_get_values(3N)
open a stream
open a stream
position forms window cursor
tell if forms field has off-screen data ahead or behind
command processor for the forms subsystem
connect fields to forms
format the general display attributes of forms
set and get forms field attributes
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

3-64 modified 30 Jun 1995
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)

fputwc (3I)

fputws (3I)

fread (3S)
free (3C)
free (3X)

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 fpgetround (3C)
See fpgetround (3C)
See fpgetround (3C)
See fputc (3S)
See puts (3S)
See putwc (3I)
See putws (3I)
buffered binary input/output
See malloc (3C)
See bsdmalloc (3X)
free(3X) See malloc(3X)
free(3X) See mapmalloc(3X)
free_field(3X) See form_field_new(3X)
free_fieldtype(3X) See form_fieldtype(3X)
free_form(3X) See form_new(3X)
free_item(3X) See menu_item_new(3X)
free_menu(3X) See menu_new(3X)
freenetconfig(3N) See getnetconfig(3N)
freopen(3B) See fopen(3B)
freopen(3S) See fopen(3S)
frexp(3C) manipulate parts of floating-point numbers See scanf(3S)
fscanf(3S) reposition a file pointer in a stream
fseek(3S) reposition a file pointer in a stream
fsetpos(3C) synchronize a file’s in-memory state with that on the physical medium
fsync(3C) get date and time See stdipc(3C)
ftell(3S) See flockfile(3S)
ftime(3C) See truncate(3C)
ftok(3C) walk a file tree
ftrylockfile(3S) See string_to_decimal(3)
fsync(3C) See flockfile(3S)
func_to_decimal(3) See fread(3S)
funlockfile(3S) See lgamma(3M)
fwrite(3S) See lgamma(3M)
gamma(3M) See econvert(3)
gamma_r(3M) See econvert(3)
gconvert(3) See ecvt(3C)
gcvt(3) get audit control file information See getacinfo(3)
gcvt(3C) See getacinfo(3)
gconvert(3) See getacinfo(3)
getacdir(3) getacinfo(3)
getacfg(3) getacinfo(3)
getacinfo(3) getacinfo(3)
getacmin(3) See getacinfo(3)
getacna(3) See getacinfo(3)
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getauclassent(3)</td>
<td>get audit_class entry</td>
</tr>
<tr>
<td>getauclassent_r(3)</td>
<td>See getauclassent(3)</td>
</tr>
<tr>
<td>getauclassnam(3)</td>
<td>See getauclassent(3)</td>
</tr>
<tr>
<td>getauclassnam_r(3)</td>
<td>See getauclassent(3)</td>
</tr>
<tr>
<td>getauditflags(3)</td>
<td>convert audit flag specifications</td>
</tr>
<tr>
<td>getauditflagsbin(3)</td>
<td>See getauditflags(3)</td>
</tr>
<tr>
<td>getauditflagschar(3)</td>
<td>See getauditflags(3)</td>
</tr>
<tr>
<td>getauevent(3)</td>
<td>get audit_user entry</td>
</tr>
<tr>
<td>getauevent_r(3)</td>
<td>See getauevent(3)</td>
</tr>
<tr>
<td>getauevnam(3)</td>
<td>See getauevent(3)</td>
</tr>
<tr>
<td>getauevnam_r(3)</td>
<td>See getauevent(3)</td>
</tr>
<tr>
<td>getauevnonam(3)</td>
<td>See getauevent(3)</td>
</tr>
<tr>
<td>getauevnum(3)</td>
<td>See getauevent(3)</td>
</tr>
<tr>
<td>getauevnum_r(3)</td>
<td>See getauevent(3)</td>
</tr>
<tr>
<td>getauuserent(3)</td>
<td>See getauusernam(3)</td>
</tr>
<tr>
<td>getauusernam(3)</td>
<td>get audit_user entry</td>
</tr>
<tr>
<td>getbegyx(3X)</td>
<td>See curs_getyx(3X)</td>
</tr>
<tr>
<td>getc(3S)</td>
<td>get character or word from a stream</td>
</tr>
<tr>
<td>getch(3X)</td>
<td>See curs_getch(3X)</td>
</tr>
<tr>
<td>getchar(3S)</td>
<td>See getc(3S)</td>
</tr>
<tr>
<td>getchar_unlocked(3S)</td>
<td>See getc(3S)</td>
</tr>
<tr>
<td>getc_unlocked(3S)</td>
<td>get pathname of current working directory</td>
</tr>
<tr>
<td>getcwd(3C)</td>
<td>convert user format date and time</td>
</tr>
<tr>
<td>getdate(3C)</td>
<td>get file descriptor table size</td>
</tr>
<tr>
<td>getdtablesize(3C)</td>
<td>return value for environment name</td>
</tr>
<tr>
<td>getenv(3C)</td>
<td>generates the process audit state</td>
</tr>
<tr>
<td>getfauditflags(3)</td>
<td>See getgrnam(3C)</td>
</tr>
<tr>
<td>getgrent(3C)</td>
<td>See getgrnam(3C)</td>
</tr>
<tr>
<td>getgrent_r(3C)</td>
<td>See getgrnam(3C)</td>
</tr>
<tr>
<td>getgrgid(3C)</td>
<td>See getgrnam(3C)</td>
</tr>
<tr>
<td>getgrgid_r(3C)</td>
<td>See getgrnam(3C)</td>
</tr>
<tr>
<td>getgrnam(3C)</td>
<td>get group entry</td>
</tr>
<tr>
<td>getgrnam_r(3C)</td>
<td>See getgrnam(3C)</td>
</tr>
<tr>
<td>gethostbyaddr(3N)</td>
<td>See gethostbyname(3N)</td>
</tr>
<tr>
<td>modified 30 Jun 1995</td>
<td></td>
</tr>
</tbody>
</table>
gethostbyaddr_r(3N) See gethostbyname(3N)
gethostbyname(3N) get network host entry
gethostbyname_r(3N) See gethostbyname(3N)
gethostent(3N) See gethostbyname(3N)
gethostent_r(3N) get unique identifier of current host
gethostid(3C) get high resolution time
gethostid(3C) get login name
gethostname(3C) See getlogin(3C)
gethostname(3C) See getlogin(3C)
gethosttime(3C) get /etc/netconfig entry corresponding to NETPATH component
gethosttime(3C) See secure_rpc(3N)
gethrtime(3C) get network entry
gethrtime(3C) See getnetbyname(3N)
gethrvtime(3C) get network configuration database entry
gethrvtime(3C) See getnetconfig(3N)
getlogin(3C) See getnetbyname(3N)
getlogin_r(3C) getnetbyname(3N) getnetbyname(3N) getnetbyname(3N)
getmaxyx(3X) getnetbyname(3N) getnetbyname(3N) getnetbyname(3N) getnetbyname(3N)
getmntany(3C) getnetbyname(3N) getnetbyname(3N) getnetbyname(3N) getnetbyname(3N)
getmntent(3C) getnetbyname(3N) getnetbyname(3N) getnetbyname(3N) getnetbyname(3N)
get_myaddress(3N) getnetbyname(3N) getnetbyname(3N) getnetbyname(3N) getnetbyname(3N)
getnetbyaddr(3N) getnetbyname(3N) getnetbyname(3N) getnetbyname(3N) getnetbyname(3N)
getnetbyaddr_r(3N) getnetbyname(3N) getnetbyname(3N) getnetbyname(3N) getnetbyname(3N)
getnetbyname(3N) getnetbyname(3N) getnetbyname(3N) getnetbyname(3N) getnetbyname(3N)
getnetbyname_r(3N) getnetbyname(3N) getnetbyname(3N) getnetbyname(3N) getnetbyname(3N)
getnetconfig(3N) getnetbyname(3N) getnetbyname(3N) getnetbyname(3N) getnetbyname(3N)
getnetconfigent(3N) getnetbyname(3N) getnetbyname(3N) getnetbyname(3N) getnetbyname(3N)
getnetent(3N) getnetbyname(3N) getnetbyname(3N) getnetbyname(3N) getnetbyname(3N)
getnetent_r(3N) getnetbyname(3N) getnetbyname(3N) getnetbyname(3N) getnetbyname(3N)
getnetgrent(3N) getnetbyname(3N) getnetbyname(3N) getnetbyname(3N) getnetbyname(3N)
getnetgrent_r(3N) getnetbyname(3N) getnetbyname(3N) getnetbyname(3N) getnetbyname(3N)
getnetname(3N) getnetbyname(3N) getnetbyname(3N) getnetbyname(3N) getnetbyname(3N)
getnetpath(3N) getnetbyname(3N) getnetbyname(3N) getnetbyname(3N) getnetbyname(3N)
getnwstr(3X) getnetbyname(3N) getnetbyname(3N) getnetbyname(3N) getnetbyname(3N)
getopt(3C) getnetbyname(3N) getnetbyname(3N) getnetbyname(3N) getnetbyname(3N)
getpagesize(3C) getnetbyname(3N) getnetbyname(3N) getnetbyname(3N) getnetbyname(3N)
getparyx(3X) getnetbyname(3N) getnetbyname(3N) getnetbyname(3N) getnetbyname(3N)
getpass(3C) getnetbyname(3N) getnetbyname(3N) getnetbyname(3N) getnetbyname(3N)
getpeername(3N) getnetbyname(3N) getnetbyname(3N) getnetbyname(3N) getnetbyname(3N)
getpass(3C) getnetbyname(3N) getnetbyname(3N) getnetbyname(3N) getnetbyname(3N)
getparyx(3X) getnetbyname(3N) getnetbyname(3N) getnetbyname(3N) getnetbyname(3N)
getpass(3C) getnetbyname(3N) getnetbyname(3N) getnetbyname(3N) getnetbyname(3N)
getpeername(3N) getnetbyname(3N) getnetbyname(3N) getnetbyname(3N) getnetbyname(3N)
getpriority(3C)
getprotobyname(3N)
getprotobyname_r(3N)
getprotobynumber(3N)
getprotobynumber_r(3N)
getprotoent(3N)
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)
getrpcbynumber(3N)
getrpcbynumber_r(3N)
getrpcpent(3N)
getrpcpent_r(3N)
getrpcport(3N)
getrpcport_r(3N)
getrusage(3C)
gets(3S)
getsecretkey(3N)
getservbyname(3N)
getservbyname_r(3N)
getservbyport(3N)
getservbyport_r(3N)
getservent(3N)
getservent_r(3N)
getsockname(3N)
getsockopt(3N)
getspent(3C)

get/set scheduling priority for process, process group or user
get protocol entry
See getprotobyname(3N)
See getprotobyname(3N)
See getprotobyname(3N)
See getprotobyname(3N)
See getprotobyname(3N)
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)
See getpwnam(3C)
get RPC entry
See getrpcbyname(3N)
See getrpcbyname(3N)
See getrpcbyname(3N)
See getrpcbyname(3N)
See getrpcbyname(3N)
See rpc_soc(3N)
get information about resource utilization
get a string from a stream
See getpublickey(3N)
get service entry
See getservbyname(3N)
See getservbyname(3N)
See getservbyname(3N)
See getservbyname(3N)
See getservbyname(3N)
get socket name
get and set options on sockets
See getservname(3N)
getspent_r(3C)  See getspnam(3C)  get password entry
getspnam(3C)  See getspnam(3C)
getspnam_r(3C)  See curs_getstr(3X)  parse suboptions from a string
getstr(3X)
getsubopt(3C)  See curs_kernel(3X)  message handling functions
getsyx(3X)
getspnam(3C)  get password entry
gettext(3I)  access utmp file entry
gettimeofday(3B)  get or set the date and time
gettimeofday(3C)  get or set the date and time
gettxt(3C)  retrieve a text string
getusershell(3C)  get legal user shells
getutent(3C)  access utmpx file entry
getutid(3C)  See getutent(3C)
getutline(3C)  See getutent(3C)
getutmp(3C)  See getutent(3C)
getutmpx(3C)  See getutent(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)
getvfspec(3C)  See getvfsent(3C)
getw(3S)  See getc(3S)  convert EUC character from the stream to Process Code
getwc(3I)  See getc(3I)
getwch(3X)  See curs_getwch(3X)
getwchar(3I)  See getc(3I)
getwd(3C)  get current working directory pathname
getwidth(3I)  get codeset information
getwin(3X)  See curs_util(3X)  convert a string of EUC characters from the stream to Process Code
getws(3I)  See curs_getwstr(3X)
getwstr(3X)  See curs_getwstr(3X)
getyx(3X)  See curs_getyx(3X)
glob(3C)
globfree(3C)
gmatch(3G)
gmtime(3C)
gmtime_r(3C)
grantpt(3C)
gsignal(3C)
halfdelay(3X)
has_colors(3X)
has_ic(3X)
has_il(3X)
hasmntopt(3C)
havedisk(3N)
hcreate(3C)
hdestroy(3C)
hide_panel(3X)
host2netname(3N)
hsearch(3C)
htonl(3N)
htons(3N)
hyperbolic(3M)
hypot(3M)
iconv(3)
iconv_close(3)
iconv_open(3)
idcok(3X)
idlok(3X)
ieee_functions(3M)
ieee_test(3M)
ilogb(3M)
immedok(3X)

generate path names matching a pattern
See glob(3C)
shell global pattern matching
See ctime(3C)
See ctime(3C)
grant access to the slave pseudo-terminal device
See ssignal(3C)
See curs_inopts(3X)
See curs_color(3X)
See curs_termattrs(3X)
See curs_termattrs(3X)
See getmntent(3C)
See rstat(3N)
See hsearch(3C)
See hsearch(3C)
See panel_show(3X)
See secure_rpc(3N)
manage hash search tables
See byteorder(3N)
See byteorder(3N)
hyperbolic functions
Euclidean distance
code conversion function
code conversion deallocation function
code conversion allocation function
See curs_outopts(3X)
See curs_outopts(3X)
appendix and related miscellaneous functions for IEEE arithmetic
IEEE test functions for verifying standard compliance
See ieee_functions(3M)
See curs_outopts(3X)
intro (3) C Library Functions

inch(3X) See curs_inch(3X)
inchnstr(3X) See curs_inchstr(3X)
inchsstr(3X) See curs_inchstr(3X)
index(3C) string operations
inet(3N) Internet address manipulation
inet_addr(3N) See inet(3N)
inet_inaof(3N) See inet(3N)
inet_makeaddr(3N) See inet(3N)
inet_netof(3N) See inet(3N)
inet_network(3N) See inet(3N)
inet_ntoa(3N) See inet(3N)
init_color(3X) See curs_color(3X)
initgroups(3C) initialize the supplementary group access list
init_pair(3X) See curs_color(3X)
initscr(3X) See curs_initscr(3X)
initstate(3C) See random(3C)
innetgr(3N) See getnetgrent(3N)
instr(3X) See curs_instr(3X)
inwch(3X) See curs_inwch(3X)
inwchnstr(3X) See curs_inwchstr(3X)
inwchstr(3X) See curs_inwchstr(3X)
inwstr(3X) See curs_inwstr(3X)
inwch(3X) See curs_inwch(3X)
inwchnstr(3X) See curs_inwchstr(3X)
inwchstr(3X) See curs_inwchstr(3X)
inwstr(3X) See curs_inwstr(3X)

3-72 modified 30 Jun 1995
isalnum(3C)
isalpha(3C)
isascii(3C)
isastream(3C)
isatty(3C)
iscntrl(3C)
isdigit(3C)
isencrypt(3G)
isendwin(3X)
isenglish(3I)
isgraph(3C)
isideogram(3I)
is_linetouched(3X)
islower(3C)
isnan(3C)
isnan(3M)
isnand(3C)
isnanf(3C)
isnumber(3I)
isphonogram(3I)
isprint(3C)
ispunct(3C)
ispace(3C)
ispecial(3I)
isupper(3C)
iswalnum(3I)
iswalpha(3I)
iswascii(3I)
iswcntrl(3I)
iswctype(3I)
iswdigit(3I)
iswgraph(3I)
is_wintouched(3X)

See ctype(3C)
See ctype(3C)
See ctype(3C)
test a file descriptor
See ttyname(3C)
See ctype(3C)
See ctype(3C)
determine whether a buffer of characters is encrypted
See curs_initscr(3X)
See iswalpha(3I)
See ctype(3C)
See iswalpha(3I)
See curs_touch(3X)
See ctype(3C)
determine type of floating-point number
See ieee_functions(3M)
See isnan(3C)
See isnan(3C)
See isnan(3C)
See iswalpha(3I)
See iswalpha(3I)
See iswalpha(3I)
See ctype(3C)
See ctype(3C)
See ctype(3C)
See iswalpha(3I)
See iswalpha(3I)
Process Code character classification macros and functions
See iswalpha(3I)
See iswalpha(3I)
test character for specified class
See iswalpha(3I)
See iswalpha(3I)
See curs_touch(3X)
iswlower(3I) See iswalpha(3I)
iswprint(3I) See iswalpha(3I)
iswpunct(3I) See iswalpha(3I)
iswspace(3I) See iswalpha(3I)
iswupper(3I) See iswalpha(3I)
iswxdigit(3I) See iswalpha(3I)
isxdigit(3C) See ctype(3C)
item_count(3X) See menu_items(3X)
item_description(3X) See menu_item_name(3X)
item_index(3X) See menu_item_current(3X)
item_init(3X) See menu_item_name(3X)
item_name(3X) See menu_item_name(3X)
item_opts(3X) See menu_item_opts(3X)
item_opts_off(3X) See menu_item_opts(3X)
item_opts_on(3X) See menu_item_opts(3X)
item_term(3X) See menu_item_name(3X)
item_userptr(3X) See menu_item_userptr(3X)
item_value(3X) See menu_item_value(3X)
item_visible(3X) See menu_item_visible(3X)
j0(3M) See bessel(3M)
j1(3M) See bessel(3M)
jn(3M) See bessel(3M)
&rand48(3C) See drand48(3C)
kerberos(3N) Kerberos authentication library
kerberos_rpc(3N) library routines for remote procedure calls using Kerberos authentication
key_decryptsession(3N) See secure_rpc(3N)
key_encryptsession(3N) See secure_rpc(3N)
key_gendes(3N) See secure_rpc(3N)
keyname(3X) See curs_util(3X)
keypad(3X) See curs_inopts(3X)
key_secretkey_is_set(3N) See secure_rpc(3N)
key_setsecret(3N) See secure_rpc(3N)
killchar(3X) See curs_termattr(3X)
killpg(3C) send signal to a process group
<table>
<thead>
<tr>
<th>Function</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>krb_get_admhst(3N)</td>
<td>See krb.realmofhost(3N)</td>
</tr>
<tr>
<td>krb_get_crend(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></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_recvauth(3N)</td>
<td>Kerberos routines for sending authentication via network stream</td>
</tr>
<tr>
<td>krb_sendauth(3N)</td>
<td>sockets</td>
</tr>
<tr>
<td>krb_set_key(3N)</td>
<td>See kerberos(3N)</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>See kvm.nextproc(3K)</td>
</tr>
<tr>
<td>kvm_getu(3K)</td>
<td>get the u-area or invocation arguments for a process</td>
</tr>
<tr>
<td>kvm_kread(3K)</td>
<td>See kvm.read(3K)</td>
</tr>
<tr>
<td>kvm_kwrite(3K)</td>
<td>See kvm.read(3K)</td>
</tr>
</tbody>
</table>

modified 30 Jun 1995
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Related Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>kvm_nextproc(3K)</td>
<td>read system process structures</td>
<td></td>
</tr>
<tr>
<td>kvm_nlist(3K)</td>
<td>get entries from kernel symbol table</td>
<td></td>
</tr>
<tr>
<td>kvm_open(3K)</td>
<td>specify a kernel to examine</td>
<td></td>
</tr>
<tr>
<td>kvm_read(3K)</td>
<td>copy data to or from a kernel image or running system</td>
<td></td>
</tr>
<tr>
<td>kvm_setproc(3K)</td>
<td>See kvm_nextproc(3K)</td>
<td></td>
</tr>
<tr>
<td>kvm_uread(3K)</td>
<td>See kvm_read(3K)</td>
<td></td>
</tr>
<tr>
<td>kvm_uwrite(3K)</td>
<td>See kvm_read(3K)</td>
<td></td>
</tr>
<tr>
<td>kvm_write(3K)</td>
<td>See kvm_read(3K)</td>
<td></td>
</tr>
<tr>
<td>l64a(3C)</td>
<td>manipulate shadow password database lock file</td>
<td></td>
</tr>
<tr>
<td>label(3)</td>
<td>See plot(3)</td>
<td></td>
</tr>
<tr>
<td>labs(3C)</td>
<td>See abs(3C)</td>
<td></td>
</tr>
<tr>
<td>lckpwdf(3C)</td>
<td>display error message in standard format and pass to logging and monitoring services</td>
<td></td>
</tr>
<tr>
<td>lcong48(3C)</td>
<td>log gamma function</td>
<td></td>
</tr>
<tr>
<td>ldexp(3C)</td>
<td>See frexp(3C)</td>
<td></td>
</tr>
<tr>
<td>ldiv(3C)</td>
<td>See div(3C)</td>
<td></td>
</tr>
<tr>
<td>leaveok(3X)</td>
<td>See curs_outopts(3X)</td>
<td></td>
</tr>
<tr>
<td>lfind(3C)</td>
<td>See lsearch(3C)</td>
<td></td>
</tr>
<tr>
<td>lfmt(3C)</td>
<td>display error message in standard format and pass to logging and monitoring services</td>
<td></td>
</tr>
<tr>
<td>lgamma(3M)</td>
<td>log gamma function</td>
<td></td>
</tr>
<tr>
<td>lgamma_r(3M)</td>
<td>See lgamma(3M)</td>
<td></td>
</tr>
<tr>
<td>libpthread(3T)</td>
<td>See threads(3T)</td>
<td></td>
</tr>
<tr>
<td>libthread(3T)</td>
<td>See threads(3T)</td>
<td></td>
</tr>
<tr>
<td>libthread_db(3T)</td>
<td>interface to libthread threads information</td>
<td></td>
</tr>
<tr>
<td>line(3)</td>
<td>See plot(3)</td>
<td></td>
</tr>
<tr>
<td>link_field(3X)</td>
<td>See form_field_new(3X)</td>
<td></td>
</tr>
<tr>
<td>link_fieldtype(3X)</td>
<td>See form_fieldtype(3X)</td>
<td></td>
</tr>
<tr>
<td>linmod(3)</td>
<td>See plot(3)</td>
<td></td>
</tr>
<tr>
<td>lio_listio(3R)</td>
<td>list directed I/O</td>
<td></td>
</tr>
<tr>
<td>listen(3N)</td>
<td>listen for connections on a socket</td>
<td></td>
</tr>
<tr>
<td>llabs(3C)</td>
<td>See abs(3C)</td>
<td></td>
</tr>
<tr>
<td>llldiv(3C)</td>
<td>See div(3C)</td>
<td></td>
</tr>
</tbody>
</table>

modified 30 Jun 1995
lltostr(3C) get numeric formatting information
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) See exp(3M)
log10(3M) See exp(3M)
log1p(3M) See exp(3M)
logb(3C) See frexp(3C)
logb(3M) See ieee_test(3M)
_longjmp(3B) See setjmp(3B)
longjmp(3B) See setjmp(3B)
longjmp(3C) See setjmp(3C)
longname(3X) See curs_termattrs(3X)
lrand48(3C) See drand48(3C)
lsearch(3C) linear search and update
madvise(3) provide advice to VM system
maillock(3X) manage lockfile for user's mailbox
major(3C) See makedev(3C)
makecontext(3C) manipulate user contexts
makedev(3C) manage a device number
mallinfo(3X) See malloc(3X)
malloc(3C) memory allocator
malloc(3X) memory allocator
malloc(3X) See bsdmalloc(3X)
mallocp(3X) See malloc(3X)
mapmalloc(3X) memory allocator
matherr(3M) math library exception-handling function
mbchar(3C) multibyte character handling
mblen(3C) See mbchar(3C)
mbstowcs(3C) See mbstring(3C)
mbstring(3C) multibyte string functions
mbtowc(3C) See mbchar(3C)
mct1(3B) memory management control

modified 30 Jun 1995
media_findname(3X) convert a supplied name into an absolute pathname that can be used to access removable media
media_getattr(3X)
media_setattr(3X)
memalign(3C)
memccpy(3C)
memchr(3C)
memcmp(3C)
memcpy(3C)
memmove(3C)
memory(3C)
memset(3C)
menu_attributes(3X) control menus display attributes
menu_back(3X)
menu_cursor(3X)
menu_driver(3X)
menu_fore(3X)
menu_format(3X)
menu_grey(3X)
menu_hook(3X)
menu_init(3X)
menu_item_current(3X) set and get current menus items
menu_item_name(3X) get menus item name and description
menu_item_new(3X) create and destroy menus items
menu_item_opts(3X) menus item option routines
menu_items(3X) connect and disconnect items to and from menus
menu_item_userptr(3X) associate application data with menus items
menu_item_value(3X) set and get menus item values
menu_item_visible(3X) tell if menus item is visible
menu_mark(3X) menus mark string routines

See media_getattr(3X)
See malloc(3C)
See memory(3C)
See memory(3C)
See memory(3C)
See memory(3C)
See memory(3C)
See memory(3C)
See menu_attributes(3X)
See menu_attributes(3X)
See menu_attributes(3X)
See menu_attributes(3X)
See menu_attributes(3X)
See menu_attributes(3X)
See menu_attributes(3X)
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>menu_new(3X)</td>
<td>create and destroy menus</td>
</tr>
<tr>
<td>menu_opts(3X)</td>
<td>menus option routines</td>
</tr>
<tr>
<td>menu_opts_off(3X)</td>
<td>See menu_opts(3X)</td>
</tr>
<tr>
<td>menu_opts_on(3X)</td>
<td>See menu_opts(3X)</td>
</tr>
<tr>
<td>menu_pad(3X)</td>
<td>set and get menus pattern match buffer</td>
</tr>
<tr>
<td>menu_pattern(3X)</td>
<td>write or erase menus from associated subwindows</td>
</tr>
<tr>
<td>menu_post(3X)</td>
<td>character based menus package</td>
</tr>
<tr>
<td>menus(3X)</td>
<td>See menu_win(3X)</td>
</tr>
<tr>
<td>menu_sub(3X)</td>
<td>See menu_hook(3X)</td>
</tr>
<tr>
<td>menu_term(3X)</td>
<td>associate application data with menus</td>
</tr>
<tr>
<td>menu_userptr(3X)</td>
<td>menus window and subwindow association routines</td>
</tr>
<tr>
<td>menu_win(3X)</td>
<td>See curs_inopts(3X)</td>
</tr>
<tr>
<td>meta(3X)</td>
<td>See makedev(3C)</td>
</tr>
<tr>
<td>minor(3C)</td>
<td>create directories in a path</td>
</tr>
<tr>
<td>mkdirp(3G)</td>
<td>create a new FIFO</td>
</tr>
<tr>
<td>mkfifo(3C)</td>
<td>make a unique file name</td>
</tr>
<tr>
<td>mktemp(3C)</td>
<td>converts a tm structure to a calendar time</td>
</tr>
<tr>
<td>mktime(3C)</td>
<td>lock (or unlock) pages in memory</td>
</tr>
<tr>
<td>mlock(3C)</td>
<td>lock or unlock address space</td>
</tr>
<tr>
<td>mlockall(3C)</td>
<td>See frexp(3C)</td>
</tr>
<tr>
<td>modf(3C)</td>
<td>See frexp(3C)</td>
</tr>
<tr>
<td>modff(3C)</td>
<td>prepare process execution profile</td>
</tr>
<tr>
<td>monitor(3C)</td>
<td>See plot(3)</td>
</tr>
<tr>
<td>move(3)</td>
<td>See curs_move(3X)</td>
</tr>
<tr>
<td>move(3X)</td>
<td>See form_field(3X)</td>
</tr>
<tr>
<td>move_field(3X)</td>
<td>See curs_alecompat(3X)</td>
</tr>
<tr>
<td>movenextch(3X)</td>
<td>See panel_move(3X)</td>
</tr>
<tr>
<td>move_panel(3X)</td>
<td>close a message queue</td>
</tr>
<tr>
<td>moveprevch(3X)</td>
<td>See curs_alecompat(3X)</td>
</tr>
<tr>
<td>mq_close(3R)</td>
<td>See mq_setattr(3R)</td>
</tr>
<tr>
<td>mq_getattr(3R)</td>
<td></td>
</tr>
</tbody>
</table>

modified 30 Jun 1995
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mq_notify(3R)</td>
<td>notify process (or thread) that a message is available on a queue</td>
</tr>
<tr>
<td>mq_open(3R)</td>
<td>open a message queue</td>
</tr>
<tr>
<td>mq_receive(3R)</td>
<td>receive a message from a message queue</td>
</tr>
<tr>
<td>mq_send(3R)</td>
<td>send a message to a message queue</td>
</tr>
<tr>
<td>mq_setattr(3R)</td>
<td>set/get message queue attributes</td>
</tr>
<tr>
<td>mq_unlink(3R)</td>
<td>remove a message queue</td>
</tr>
<tr>
<td>mrand48(3C)</td>
<td>See drand48(3C)</td>
</tr>
<tr>
<td>msync(3C)</td>
<td>synchronize memory with physical storage</td>
</tr>
<tr>
<td>munlock(3C)</td>
<td>See mlock(3C)</td>
</tr>
<tr>
<td>munlockall(3C)</td>
<td>See mlockall(3C)</td>
</tr>
<tr>
<td>mutex(3T)</td>
<td>mutual exclusion locks</td>
</tr>
<tr>
<td>mutex_destroy(3T)</td>
<td>See mutex(3T)</td>
</tr>
<tr>
<td>mutex_init(3T)</td>
<td>See mutex(3T)</td>
</tr>
<tr>
<td>mutex_lock(3T)</td>
<td>See mutex(3T)</td>
</tr>
<tr>
<td>mutex_trylock(3T)</td>
<td>See mutex(3T)</td>
</tr>
<tr>
<td>mutex_unlock(3T)</td>
<td>See mutex(3T)</td>
</tr>
<tr>
<td>mvaddch(3X)</td>
<td>See curs_addch(3X)</td>
</tr>
<tr>
<td>mvaddchnstr(3X)</td>
<td>See curs_addchstr(3X)</td>
</tr>
<tr>
<td>mvaddchstr(3X)</td>
<td>See curs_addchstr(3X)</td>
</tr>
<tr>
<td>mvaddnstr(3X)</td>
<td>See curs_addnstr(3X)</td>
</tr>
<tr>
<td>mvaddnwstr(3X)</td>
<td>See curs_addwstr(3X)</td>
</tr>
<tr>
<td>mvaddstr(3X)</td>
<td>See curs_addstr(3X)</td>
</tr>
<tr>
<td>mvaddwch(3X)</td>
<td>See curs_addwch(3X)</td>
</tr>
<tr>
<td>mvaddwchnstr(3X)</td>
<td>See curs_addwchstr(3X)</td>
</tr>
<tr>
<td>mvaddwchstr(3X)</td>
<td>See curs_addwchstr(3X)</td>
</tr>
<tr>
<td>mvaddwstr(3X)</td>
<td>See curs_addwstr(3X)</td>
</tr>
<tr>
<td>mvcur(3X)</td>
<td>See curs_terminfo(3X)</td>
</tr>
<tr>
<td>mvdelch(3X)</td>
<td>See curs_delch(3X)</td>
</tr>
<tr>
<td>mvderwin(3X)</td>
<td>See curs_window(3X)</td>
</tr>
<tr>
<td>mvgetch(3X)</td>
<td>See curs_getch(3X)</td>
</tr>
<tr>
<td>mvgetnstr(3X)</td>
<td>See curs_getwstr(3X)</td>
</tr>
<tr>
<td>mvgetstr(3X)</td>
<td>See curs_getstr(3X)</td>
</tr>
<tr>
<td>mvgetwch(3X)</td>
<td>See curs_getwch(3X)</td>
</tr>
</tbody>
</table>

modified 30 Jun 1995
<table>
<thead>
<tr>
<th>Function</th>
<th>See Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>mvgetwstr(3X)</td>
<td>See curs_getwstr(3X)</td>
</tr>
<tr>
<td>mvinch(3X)</td>
<td>See curs_inch(3X)</td>
</tr>
<tr>
<td>mvinchnstr(3X)</td>
<td>See curs_inchstr(3X)</td>
</tr>
<tr>
<td>mvinchstr(3X)</td>
<td>See curs_inchstr(3X)</td>
</tr>
<tr>
<td>mvinnstr(3X)</td>
<td>See curs_instr(3X)</td>
</tr>
<tr>
<td>mvinnwstr(3X)</td>
<td>See curs_inwstr(3X)</td>
</tr>
<tr>
<td>mvinsch(3X)</td>
<td>See curs_insch(3X)</td>
</tr>
<tr>
<td>mvinsnstr(3X)</td>
<td>See curs_insstr(3X)</td>
</tr>
<tr>
<td>mvinsnstrw(3X)</td>
<td>See curs_insstrw(3X)</td>
</tr>
<tr>
<td>mvinsstr(3X)</td>
<td>See curs_insstr(3X)</td>
</tr>
<tr>
<td>mvinstr(3X)</td>
<td>See curs_instr(3X)</td>
</tr>
<tr>
<td>mvinswch(3X)</td>
<td>See curs_inswch(3X)</td>
</tr>
<tr>
<td>mvinswstr(3X)</td>
<td>See curs_inswstr(3X)</td>
</tr>
<tr>
<td>mvinwch(3X)</td>
<td>See curs_inwch(3X)</td>
</tr>
<tr>
<td>mvinwchnstr(3X)</td>
<td>See curs_inwchstr(3X)</td>
</tr>
<tr>
<td>mvinwchstr(3X)</td>
<td>See curs_inwchstr(3X)</td>
</tr>
<tr>
<td>mvinwstr(3X)</td>
<td>See curs_inwstr(3X)</td>
</tr>
<tr>
<td>mvpprintw(3X)</td>
<td>See curs_printw(3X)</td>
</tr>
<tr>
<td>mvscanw(3X)</td>
<td>See curs_scanw(3X)</td>
</tr>
<tr>
<td>mvwaddch(3X)</td>
<td>See curs_addch(3X)</td>
</tr>
<tr>
<td>mvwaddchnstr(3X)</td>
<td>See curs_addchstr(3X)</td>
</tr>
<tr>
<td>mvwaddchstr(3X)</td>
<td>See curs_addchstr(3X)</td>
</tr>
<tr>
<td>mvwaddnstr(3X)</td>
<td>See curs_addnstr(3X)</td>
</tr>
<tr>
<td>mvwaddnstrw(3X)</td>
<td>See curs_addnstrw(3X)</td>
</tr>
<tr>
<td>mvwaddstr(3X)</td>
<td>See curs_addstr(3X)</td>
</tr>
<tr>
<td>mvwaddwch(3X)</td>
<td>See curs_addwch(3X)</td>
</tr>
<tr>
<td>mvwaddwchnstr(3X)</td>
<td>See curs_addwchstr(3X)</td>
</tr>
<tr>
<td>mvwaddwchstr(3X)</td>
<td>See curs_addwchstr(3X)</td>
</tr>
<tr>
<td>mvwaddwstr(3X)</td>
<td>See curs_addwstr(3X)</td>
</tr>
<tr>
<td>mvwdelch(3X)</td>
<td>See curs_delch(3X)</td>
</tr>
<tr>
<td>mvwgetch(3X)</td>
<td>See curs_getch(3X)</td>
</tr>
<tr>
<td>mvwgetnstr(3X)</td>
<td>See curs_getnstr(3X)</td>
</tr>
<tr>
<td>mvwgetstr(3X)</td>
<td>See curs_getstr(3X)</td>
</tr>
<tr>
<td>mvwgetwch(3X)</td>
<td>See curs_getwch(3X)</td>
</tr>
<tr>
<td>mvwgetwstr(3X)</td>
<td>See curs_getwstr(3X)</td>
</tr>
</tbody>
</table>

modified 30 Jun 1995
mvwin(3X) See curs_window(3X)
mvwinch(3X) See curs_inch(3X)
mvwinchnstr(3X) See curs_inchstr(3X)
mvwinchstr(3X) See curs_inchstr(3X)
mwinnstr(3X) See curs_instr(3X)
mwinnwstr(3X) See curs_inwstr(3X)
mwinsch(3X) See curs_insch(3X)
mwinsnstr(3X) See curs_insstr(3X)
mwinsnwstr(3X) See curs_inswstr(3X)
mwinsstr(3X) See curs_insstr(3X)
mwinstr(3X) See curs_instr(3X)
mwinswch(3X) See curs_inschwch(3X)
mwinswstr(3X) See curs_insstr(3X)
mwinwch(3X) See curs_inwch(3X)
mwinwchnstr(3X) See curs_inwchstr(3X)
mwinwchstr(3X) See curs_inwchstr(3X)
mwinwstr(3X) See curs_insstr(3X)
mwprintw(3X) See curs_printw(3X)
mwscanw(3X) See curs_scanw(3X)
nanosleep(3R) See curs_kernel(3X)
napms(3X) See getnetconfig(3N)
nc_perror(3N) See getnetconfig(3N)
nc_sperror(3N) data base subroutines
db(3)
generic transport name-to-address translation
netdir_free(3N) See netdir(3N)
etdir_getbyaddr(3N) See netdir(3N)
etdir_getbyname(3N) See netdir(3N)
etdir_mergeaddr(3N) See netdir(3N)
etdir_options(3N) See netdir(3N)
etdir_perror(3N) See netdir(3N)
etdir_sperror(3N) See netdir(3N)
etname2host(3N) See secure_rpc(3N)
etname2user(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)
new_page(3X)  See form_new_page(3X)
new_panel(3X)  See panel_new(3X)
newterm(3X)  See curs_initscr(3X)
newwin(3X)  See curs_window(3X)
nexafter(3C)  See frexp(3C)
nexafter(3M)  See ieee_functions(3M)
nextkey(3B)  See dbm(3B)
nftw(3C)  See ftw(3C)
nice(3B)  change priority of a process
nis_add(3N)  See nis_names(3N)
nis_add_entry(3N)  See nis_tables(3N)
nis_addmember(3N)  See nis_groups(3N)
nis_checkpoint(3N)  See nis_ping(3N)
nis_clone_object(3N)  See nis_subr(3N)
nis_creategroup(3N)  See nis_groups(3N)
nis_db(3N)  NIS+ Database access functions
nis_destroygroup(3N)  See nis_groups(3N)
nis_destroy_object(3N)  See nis_subr(3N)
nis_dir_cmp(3N)  See nis_subr(3N)
nis_domain_of(3N)  See nis_subr(3N)
nis_error(3N)  display NIS+ error messages
nis_first_entry(3N)  See nis_tables(3N)
nis_freenames(3N)  See nis_subr(3N)
nis_freeservlist(3N)  See nis_names(3N)
nis_freevlist(3N)  See nis_server(3N)
nis_freetags(3N)  See nis_server(3N)
nis_getnames(3N)  See nis_subr(3N)
nis_getservlist(3N)  See nis_server(3N)
nis_groups(3N)  NIS+ group manipulation functions

modified 30 Jun 1995
nis_ismember(3N)
  See nis_groups(3N)
nis_leaf_of(3N)
  See nis_subr(3N)
nis_error(3N)
  See nis_error(3N)
nis_list(3N)
  See nis_subr(3N)
nis_local_directory(3N)
  See nis_local_names(3N)
nis_local_group(3N)
  See nis_local_names(3N)
nis_local_host(3N)
  See nis_local_names(3N)
nis_local_names(3N)
  NIS+ local names
nis_local_principal(3N)
  See nis_local_names(3N)
nis_lookup(3N)
  See nis_names(3N)
__nis_map_group(3N)
  See nisGroups(3N)
nis_map_group(3N)
  See nis_groups(3N)
nis_mkdir(3N)
  See nis_server(3N)
nis_modify(3N)
  See nis_names(3N)
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)
nlist(3B)  get entries from symbol table
nlist(3E)  get entries from name list
nl_langinfo(3C)  language information
nlsgtcall(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)

ocdelay(3X)  See curs_inopts(3X)

noecho(3X)  See curs_inopts(3X)
nlonl(3X)  See curs_outopts(3X)

noqiflush(3X)  See curs_inopts(3X)
noraw(3X)  annotate source code with info for tools

NOTE(3X)  See NOTE(3X)

_NotX(3X)  See curs_inopts(3X)

nrand48(3C)  See drand48(3C)

ntohl(3N)  See byteorder(3N)

ntohs(3N)  offset of structure member

offsetof(3C)  See directory(3C)

opendir(3C)  See syslog(3)

openlog(3)  See plot(3)

openpl(3)  See plot(3)

openvt(3)  See curs_overlay(3X)

overlay(3X)  See curs_overlay(3X)

overwrite(3X)  See p2open(3G)

p2close(3G)  open, close pipes to and from a command

p2open(3G)  See curs_color(3X)

pair_content(3X)  panels deck traversal primitives

panel_above(3X)
panel_below(3X)
panel_hidden(3X)
panel_move(3X)
panel_new(3X)
panels(3X)
panel_show(3X)
panel_top(3X)
panel_update(3X)
panel_userptr(3X)
panel_window(3X)
pathfind(3G)
pclose(3S)
pechochar(3X)
pechowchar(3X)
perror(3C)
pfmt(3C)
plock(3C)
plot(3)
pmap_getmaps(3N)
pmap_getport(3N)
pmap_rmtcall(3N)
pmap_set(3N)
pmap_unset(3N)
pnoutrefresh(3X)
point(3)
popen(3S)
pos_form_cursor(3X)
pos_menu_cursor(3X)
post_form(3X)
post_menu(3X)

See panel_above(3X)
See panel_show(3X)
move a panels window on the virtual screen
create and destroy panels
character based panels package
panels deck manipulation routines
panels deck manipulation routines
panels virtual screen refresh routine
associate application data with a panels panel
get or set the current window of a panels panel
search for named file in named directories
See popen(3S)
See curs_pad(3X)
See curs_pad(3X)
print system error messages
display error message in standard format
lock or unlock into memory process, text, or data
graphics interface
See rpc_soc(3N)
See rpc_soc(3N)
See rpc_soc(3N)
See rpc_soc(3N)
See curs_pad(3X)
See plot(3)
initiate pipe to/from a process
See form_cursor(3X)
See menu_cursor(3X)
See form_post(3X)
See menu_post(3X)
pow(3M) See exp(3M)
prefresh(3X) See curs_pad(3X)
printf(3B) formatted output conversion
printf(3S) print formatted output
printw(3X) See curs_printw(3X)
proc_service(3T) process service interface
psignalinfo(3C) See psignal(3C)
psignal(3B) system signal messages
psignal(3C) system signal messages
ps_lcontinue(3T) See proc_service(3T)
ps_lgetfpregs(3T) See proc_service(3T)
ps_lgetregs(3T) See proc_service(3T)
ps_lgetxregs(3T) See proc_service(3T)
ps_lgetxregs_size(3T) See proc_service(3T)
ps_lsetfpregs(3T) See proc_service(3T)
ps_lsetregs(3T) See proc_service(3T)
ps_lsetxregs(3T) See proc_service(3T)
ps_lstop(3T) See proc_service(3T)
ps_pcontinue(3T) See proc_service(3T)
ps pdread(3T) See proc_service(3T)
ps pdwrite(3T) See proc_service(3T)
ps pglobal_lookup(3T) See proc_service(3T)
ps plog(3T) See proc_service(3T)
ps pstop(3T) See proc_service(3T)
ps pthread(3T) See proc_service(3T)
ps pthread_atfork(3T) register fork handlers
pthread_attr_destroy(3T) See pthread_attr_init(3T)
pthread_attr_getdetachstate(3T) See pthread_attr_init(3T)
pthread_attr_getinheritsched(3T) See pthread_attr_init(3T)
pthread_attr_getschedparam(3T) See pthread_attr_init(3T)
 pthread_attr_getschedpolicy(3T) See pthread_attr_init(3T)
 pthread_attr_getscope(3T) See pthread_attr_init(3T)
 pthread_attr_getstackaddr(3T) See pthread_attr_init(3T)
 pthread_attr_getstacksize(3T) See pthread_attr_init(3T)

modified 30 Jun 1995
Intro (3) C Library Functions SunOS 5.5

pthread_attr_init(3T) thread creation attributes
See pthread_attr_init(3T)

pthread_attr_setdetachstate(3T) See pthread_attr_init(3T)

pthread_attr_setinheritsched(3T) See pthread_attr_init(3T)

pthread_attr_setschedparam(3T) See pthread_attr_init(3T)

pthread_attr_setschedpolicy(3T) See pthread_attr_init(3T)

pthread_attr_setscope(3T) See pthread_attr_init(3T)

pthread_attr_setstackaddr(3T) See pthread_attr_init(3T)

pthread_attr_setstacksize(3T) See pthread_attr_init(3T)

pthread_cancel(3T) See cancellation(3T)

pthread_cleanup_pop(3T) See cancellation(3T)

pthread_cleanup_push(3T) See cancellation(3T)

pthread_condattr_destroy(3T) See pthread_condattr_init(3T)

pthread_condattr_getpshared(3T) See pthread_condattr_init(3T)

pthread_condattr_init(3T) condition variable initialization attributes
See pthread_condattr_init(3T)

pthread_condattr_setpshared(3T) See pthread_condattr_init(3T)

pthread_cond_broadcast(3T) See condition(3T)

pthread_cond_destroy(3T) See condition(3T)

pthread_cond_init(3T) See condition(3T)

pthread_cond_signal(3T) See condition(3T)

pthread_cond_timedwait(3T) See condition(3T)

pthread_cond_wait(3T) See condition(3T)

pthread_create(3T) thread creation
dynamically detaching a thread

c ompare thread IDs

pthread_detach(3T) thread termination

pthread_equal(3T) See pthread_setschedparam(3T)

pthread_exit(3T) See pthread_key_create(3T)

pthread_getschedparam(3T) wait for thread termination

pthread_getspecific(3T) thread-specific data functions

pthread_join(3T) See pthread_key_create(3T)

pthread_key_create(3T) send a signal to a thread

pthread_key_delete(3T) See pthread_key_create(3T)

pthread_kill(3T) See pthread_key_create(3T)

pthread_mutexattr_destroy(3T) See pthread_key_create(3T)

pthread_mutexattr_getprioceiling(3T) See pthread_key_create(3T)

pthread_mutexattr_getprotocol(3T) See pthread_key_create(3T)
pthread_mutexattr_getpshared(3T)
See pthread_mutexattr_init(3T)
mutex initialization attributes

pthread_mutexattr_init(3T)
See pthread_mutexattr_init(3T)

pthread_mutexattr_setprioceiling(3T)
See pthread_mutexattr_init(3T)
See pthread_mutexattr_init(3T)
See pthread_mutexattr_init(3T)
See mutex(3T)
See pthread_mutex_setprioceiling(3T)
See mutex(3T)
See mutex(3T)
change the priority ceiling of a mutex
See mutex(3T)
See mutex(3T)
dynamic package initialization
See threads(3T)
get calling thread's ID
See cancellation(3T)
See cancellation(3T)
dynamic access to thread scheduling
See pthread_key_create(3T)
change and/or examine calling thread's signal mask
See cancellation(3T)
get name of the slave pseudo-terminal device
See getpublickey(3N)
put character or word on a stream
See putc(3S)
See putc(3S)
See putc(3S)
change or add value to environment
See getmntent(3C)
See curs terminfo(3X)
write password file entry
put a string on a stream
write shadow password file entry
pututline(3C)  See getutent(3C)
pututxline(3C)  See getutxent(3C)
putw(3S)  See putc(3S)
putwc(3I)
putwchar(3I)
putwin(3X)
putws(3I)
qeconvert(3)  See econvert(3)
qfconvert(3)  See econvert(3)
qgconvert(3)  See econvert(3)
qiflush(3X)
qsort(3C)
quadruple_to_decimal(3)
rac_drop(3N)  See rpc_rac(3N)
rac_poll(3N)  See rpc_rac(3N)
rac_recv(3N)  See rpc_rac(3N)
rac_send(3N)
raise(3C)  send signal to program
rand(3B)  simple random number generator
rand(3C)  simple random-number generator
random(3C)  better random number generator; routines for changing generators
rand_r(3C)  See rand(3C)
raw(3X)
rcmd(3N)
readdir(3B)  read a directory entry
readdir(3C)  See directory(3C)
readdir_r(3C)  See directory(3C)
read_vtoc(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)  returns the real file name
realpath(3C)  reboot system or halt processor
re_boot(3C)  See regex(3C)  receive a message from a socket
re_comp(3C)  See recv(3N)
recv(3N)  See recv(3N)
recvfrom(3N)  See curs_refresh(3X)
recvmsg(3N)  See regex(3C)
re_drawwin(3X)  See curs_refresh(3X)
re_exec(3C)  compile and execute regular expression
refresh(3X)  regular expression matching
regcmp(3G)  See regcomp(3C)
regcomp(3C)  regular expression handle
regerror(3C)  See regcomp(3G)
regex(3C)  See regcomp(3C)
regex(3G)  regular expression compile and
regex(3C)  match routines
regexec(3C)  See regcomp(3C)
regexpr(3G)  See regcomp(3C)
regfree(3C)  regular expression compile and
registerrpc(3N)  match routines
remque(3C)  See regcomp(3C)
reg free(3C)  See rpc_soc(3N)
regcmp(3C)  See ieee_functions(3M)
regerror(3C)  remove file
regex(3C)  See insque(3C)
regex(3G)  See panel_window(3X)
regexec(3C)  See curs_kernel(3X)
regexpr(3G)  See curs_kernel(3X)
regfree(3C)  See resolver(3N)
registerrpc(3N)  resolver routines
remove(3C)  See resolver(3N)
remainder(3M)  See resolver(3N)
remque(3C)  See resolver(3N)
replace_panel(3X)  See resolver(3N)
reset_prog_mode(3X)  See resolver(3N)
reset_shell_mode(3X)  See resolver(3N)
resetty(3X)  See curs_terminfo(3X)
reset_init(3N)  See fseek(3S)
res_mkquery(3N)  See directory(3C)
resolver(3N)  See fseek(3S)
resolver(3N)  See directory(3C)
res_search(3N)  modifier 30 Jun 1995 3-91
res_send(3N)  See directory(3C)
restartterm(3X)  See directory(3C)
rew ind(3S)  See directory(3C)
rew inddir(3C)  See directory(3C)
rexec(3N) return stream to a remote command
See index(3C)
See floor(3M)

rindex(3C) See index(3C)

rint(3M)

ripoffline(3X) See curs_kernel(3X)

rmdirp(3G) See mkdirp(3G)

rnmusers(3N) See rusers(3N)

rpc(3N)

library routines for remote procedure calls
See rpcbind(3N)

rpcb_getaddr(3N) See rpcbind(3N)

rpcb_getmaps(3N) See rpcbind(3N)

rpcb_gettime(3N) See rpcbind(3N)

rpcbind(3N) library routines for RPC bind service
See rpcbind(3N)

rpc_rmtcall(3N) See rpc_clnt_calls(3N)

rpc_broadcast(3N) See rpc_clnt_calls(3N)

rpc_broadcast_exp(3N) See rpcbind(3N)

rpc_set(3N) See rpcbind(3N)

rpc_unset(3N) See rpcbind(3N)

rpc_call(3N) See rpc_clnt_calls(3N)

rpc_clnt_auth(3N)

library routines for client side remote procedure call authentication
library routines for client side calls
library routines for dealing with creation and manipulation of CLIENT handles
library routine for manipulating global RPC attributes for client and server applications

See rpc_clnt_create(3N)

See rpc_svc_reg(3N)

remote asynchronous calls

See rpc_svc_reg(3N)

obsolete library routines for RPC library routines for RPC servers library routines for the creation of server handles library routines for server side remote procedure call errors library routines for registering
rpc_xdr(3N) XDR library routines for remote procedure calls
See rcmd(3N)

rresvport(3N) get performance data from remote kernel
See rcmd(3N)

rstat(3N) return information about users on remote machines
write to specified remote machines
multiple readers, single writer locks
See rwlock(3T)

ruserok(3N) See rcmd(3N)
rusers(3N)

rwall(3N)
rwlock(3T)
rwlock_destroy(3T)
rwlock_init(3T)
rw_rdlock(3T)
rw_tryrdlock(3T)
rw_trywrlock(3T)
rw_unlock(3T)
rw_wrlock(3T)
savetty(3X)
scalb(3C)
scalb(3M)
scalbn(3M)
scale_form(3X)
scale_menu(3X)
scandir(3B)
scanf(3S)
scanw(3X)
sched_getparam(3R)
sched_get_priority_max(3R)
sched_get_priority_min(3R)
sched_setscheduler(3R)
sched_rr_get_interval(3R)
sched_setparam(3R)
sched_setscheduler(3R)
sched_yield(3R) set/get scheduling parameters
set/get scheduling policy and scheduling parameters
yield processor

modified 30 Jun 1995
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>scr_dump(3X)</td>
<td>See curs_scr_dump(3X)</td>
</tr>
<tr>
<td>scr_init(3X)</td>
<td>See curs_scr_dump(3X)</td>
</tr>
<tr>
<td>scr(3X)</td>
<td>See curs_scroll(3X)</td>
</tr>
<tr>
<td>scroll(3X)</td>
<td>See curs_scroll(3X)</td>
</tr>
<tr>
<td>scrollok(3X)</td>
<td>See curs_outopts(3X)</td>
</tr>
<tr>
<td>scr_restore(3X)</td>
<td>See curs_scr_dump(3X)</td>
</tr>
<tr>
<td>scr_set(3X)</td>
<td>See curs_scr_dump(3X)</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 econvert(3)</td>
</tr>
<tr>
<td>seed48(3C)</td>
<td></td>
</tr>
<tr>
<td>seekdir(3C)</td>
<td>See drand48(3C)</td>
</tr>
<tr>
<td>select(3C)</td>
<td>See directory(3C)</td>
</tr>
<tr>
<td>sema_destroy(3T)</td>
<td>synchronous I/O multiplexing</td>
</tr>
<tr>
<td>sema_init(3T)</td>
<td>See semaphore(3T)</td>
</tr>
<tr>
<td>semaphore(3T)</td>
<td>See semaphore(3T)</td>
</tr>
<tr>
<td>sema_post(3T)</td>
<td>semaphores</td>
</tr>
<tr>
<td>sema_trywait(3T)</td>
<td>See semaphore(3T)</td>
</tr>
<tr>
<td>sema_wait(3T)</td>
<td>See semaphore(3T)</td>
</tr>
<tr>
<td>sem_close(3R)</td>
<td>close a named semaphore</td>
</tr>
<tr>
<td>sem_destroy(3R)</td>
<td>destroy an unnamed semaphore</td>
</tr>
<tr>
<td>sem_getvalue(3R)</td>
<td>get the value of a semaphore</td>
</tr>
<tr>
<td>sem_init(3R)</td>
<td>initialize an unnamed semaphore</td>
</tr>
<tr>
<td>sem_open(3R)</td>
<td>initialize/open a named semaphore</td>
</tr>
<tr>
<td>sem_post(3R)</td>
<td>increment the count of a semaphore</td>
</tr>
<tr>
<td>sem_trywait(3R)</td>
<td>See sem_wait(3R)</td>
</tr>
<tr>
<td>sem_unlink(3R)</td>
<td>remove a named semaphore</td>
</tr>
<tr>
<td>sem_wait(3R)</td>
<td>acquire or wait for a semaphore</td>
</tr>
<tr>
<td>send(3N)</td>
<td>send a message from a socket</td>
</tr>
<tr>
<td>sendmsg(3N)</td>
<td>See send(3N)</td>
</tr>
<tr>
<td>sendto(3N)</td>
<td>See send(3N)</td>
</tr>
<tr>
<td>setac(3)</td>
<td>See getacie(3)</td>
</tr>
<tr>
<td>setauctlass(3)</td>
<td>See getauclassent(3)</td>
</tr>
<tr>
<td>setauevent(3)</td>
<td>See getauevent(3)</td>
</tr>
<tr>
<td>setauuser(3)</td>
<td>See getauusernam(3)</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>setbuf(3S)</td>
<td>assign buffering to a stream</td>
</tr>
<tr>
<td>setbuffer(3C)</td>
<td>assign buffering to a stream</td>
</tr>
<tr>
<td>setcat(3C)</td>
<td>define default catalog</td>
</tr>
<tr>
<td>set_current_field(3X)</td>
<td>set current field for a form.</td>
</tr>
<tr>
<td>set_current_item(3X)</td>
<td>set current item for a menu.</td>
</tr>
<tr>
<td>set_curterm(3X)</td>
<td>set current term for a form.</td>
</tr>
<tr>
<td>set_field_back(3X)</td>
<td>set field back</td>
</tr>
<tr>
<td>set_field_buffer(3X)</td>
<td>set field buffer</td>
</tr>
<tr>
<td>set_field_fore(3X)</td>
<td>set field fore</td>
</tr>
<tr>
<td>set_field_init(3X)</td>
<td>set field init</td>
</tr>
<tr>
<td>set_field_just(3X)</td>
<td>set field just</td>
</tr>
<tr>
<td>set_field_opts(3X)</td>
<td>set field options</td>
</tr>
<tr>
<td>set_field_pad(3X)</td>
<td>set field pad</td>
</tr>
<tr>
<td>set_field_status(3X)</td>
<td>set field status</td>
</tr>
<tr>
<td>set_field_term(3X)</td>
<td>set field term</td>
</tr>
<tr>
<td>set_field_type(3X)</td>
<td>set field type</td>
</tr>
<tr>
<td>set_fieldtype_arg(3X)</td>
<td>set field type argument.</td>
</tr>
<tr>
<td>set_fieldtype_choice(3X)</td>
<td>set field type choice.</td>
</tr>
<tr>
<td>set_field_userptr(3X)</td>
<td>set field userptr</td>
</tr>
<tr>
<td>set_form_fields(3X)</td>
<td>set form fields</td>
</tr>
<tr>
<td>set_form_init(3X)</td>
<td>set form init</td>
</tr>
<tr>
<td>set_form_opts(3X)</td>
<td>set form options</td>
</tr>
<tr>
<td>set_form_page(3X)</td>
<td>set form page</td>
</tr>
<tr>
<td>set_form_sub(3X)</td>
<td>set form sub</td>
</tr>
<tr>
<td>set_form_term(3X)</td>
<td>set form term</td>
</tr>
<tr>
<td>set_form_userptr(3X)</td>
<td>set form userptr</td>
</tr>
<tr>
<td>set_form_win(3X)</td>
<td>set form win</td>
</tr>
<tr>
<td>setgrent(3C)</td>
<td>set grent</td>
</tr>
<tr>
<td>sethostent(3N)</td>
<td>set hostent</td>
</tr>
<tr>
<td>sethostname(3C)</td>
<td>set hostname</td>
</tr>
<tr>
<td>set_item_init(3X)</td>
<td>set item init</td>
</tr>
<tr>
<td>set_item_opts(3X)</td>
<td>set item options</td>
</tr>
<tr>
<td>set_item_term(3X)</td>
<td>set item term</td>
</tr>
<tr>
<td>set_item_userptr(3X)</td>
<td>set item userptr</td>
</tr>
<tr>
<td>set_item_value(3X)</td>
<td>set item value</td>
</tr>
</tbody>
</table>

modified 30 Jun 1995
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>setjmp(3B)</td>
<td>non-local goto</td>
</tr>
<tr>
<td>_setjmp(3B)</td>
<td>See <code>setjmp(3B)</code></td>
</tr>
<tr>
<td>setjmp(3C)</td>
<td>non-local goto</td>
</tr>
<tr>
<td>setkey(3C)</td>
<td>See <code>crypt(3C)</code></td>
</tr>
<tr>
<td>setlabel(3C)</td>
<td>define the label for pfmt() and fmt().</td>
</tr>
<tr>
<td>getlinebuf(3C)</td>
<td>See <code>setbuffer(3C)</code></td>
</tr>
<tr>
<td>setlocale(3C)</td>
<td>modify and query a program's locale</td>
</tr>
<tr>
<td>setlogmask(3)</td>
<td>See <code>syslog(3)</code></td>
</tr>
<tr>
<td>set_max_field(3X)</td>
<td></td>
</tr>
<tr>
<td>set_menu_back(3X)</td>
<td></td>
</tr>
<tr>
<td>set_menu_fore(3X)</td>
<td></td>
</tr>
<tr>
<td>set_menu_format(3X)</td>
<td></td>
</tr>
<tr>
<td>set_menu_grey(3X)</td>
<td></td>
</tr>
<tr>
<td>set_menu_init(3X)</td>
<td></td>
</tr>
<tr>
<td>set_menu_items(3X)</td>
<td></td>
</tr>
<tr>
<td>set_menu_mark(3X)</td>
<td></td>
</tr>
<tr>
<td>set_menu_opts(3X)</td>
<td></td>
</tr>
<tr>
<td>set_menu_padi(3X)</td>
<td></td>
</tr>
<tr>
<td>set_menu_pattern(3X)</td>
<td></td>
</tr>
<tr>
<td>set_menu_sub(3X)</td>
<td></td>
</tr>
<tr>
<td>set_menu_term(3X)</td>
<td></td>
</tr>
<tr>
<td>set_menu_userptr(3X)</td>
<td></td>
</tr>
<tr>
<td>set_menu_win(3X)</td>
<td></td>
</tr>
<tr>
<td>setnetconfig(3N)</td>
<td>See <code>getnetconfig(3N)</code></td>
</tr>
<tr>
<td>setnetent(3N)</td>
<td>See <code>getnetbyname(3N)</code></td>
</tr>
<tr>
<td>setnetgrent(3N)</td>
<td>See <code>getnetgrent(3N)</code></td>
</tr>
<tr>
<td>setnetpath(3N)</td>
<td>See <code>getnetpath(3N)</code></td>
</tr>
<tr>
<td>set_new_page(3X)</td>
<td>See <code>form_new_page(3X)</code></td>
</tr>
<tr>
<td>set_panel_userptr(3X)</td>
<td>See <code>panel_userptr(3X)</code></td>
</tr>
<tr>
<td>setpriority(3C)</td>
<td>See <code>getpriority(3C)</code></td>
</tr>
<tr>
<td>setprotoent(3N)</td>
<td>See <code>getprotobynname(3N)</code></td>
</tr>
<tr>
<td>setpwent(3C)</td>
<td>See <code>getpwnam(3C)</code></td>
</tr>
<tr>
<td>setrpcent(3N)</td>
<td>See <code>getrpcbyname(3N)</code></td>
</tr>
<tr>
<td>setscreg(3X)</td>
<td>See <code>curs_outopts(3X)</code></td>
</tr>
<tr>
<td>setservent(3N)</td>
<td>See <code>getservbyname(3N)</code></td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>setsockopt(3N)</code></td>
<td>See <code>getsockopt(3N)</code></td>
</tr>
<tr>
<td><code>setspent(3C)</code></td>
<td>See <code>getspnam(3C)</code></td>
</tr>
<tr>
<td><code>setstate(3C)</code></td>
<td>See <code>random(3C)</code></td>
</tr>
<tr>
<td><code>setsyx(3X)</code></td>
<td>See <code>curs_kernel(3X)</code></td>
</tr>
<tr>
<td><code>set Term(3X)</code></td>
<td>See <code>curs_initscr(3X)</code></td>
</tr>
<tr>
<td><code>setterm(3X)</code></td>
<td>See <code>curs_terminfo(3X)</code></td>
</tr>
<tr>
<td><code>settimeofday(3B)</code></td>
<td>See <code>gettimeofday(3B)</code></td>
</tr>
<tr>
<td><code>settimeofday(3C)</code></td>
<td>See <code>gettimeofday(3C)</code></td>
</tr>
<tr>
<td><code>set_top_row(3X)</code></td>
<td>See <code>menu_item_current(3X)</code></td>
</tr>
<tr>
<td><code>setupterm(3X)</code></td>
<td>See <code>curs_terminfo(3X)</code></td>
</tr>
<tr>
<td><code>setusershell(3C)</code></td>
<td>See <code>getusershell(3C)</code></td>
</tr>
<tr>
<td><code>setutent(3C)</code></td>
<td>See <code>getutent(3C)</code></td>
</tr>
<tr>
<td><code>setutxent(3C)</code></td>
<td>See <code>setbuf(3S)</code></td>
</tr>
<tr>
<td><code>setvbuf(3S)</code></td>
<td>See <code>econvert(3)</code></td>
</tr>
<tr>
<td><code>sfconvert(3)</code></td>
<td>See <code>econvert(3)</code></td>
</tr>
<tr>
<td><code>shm_open(3R)</code></td>
<td>open a shared memory object</td>
</tr>
<tr>
<td><code>shm_unlink(3R)</code></td>
<td>remove a shared memory object</td>
</tr>
<tr>
<td><code>show_panel(3X)</code></td>
<td>See <code>panel_show(3X)</code></td>
</tr>
<tr>
<td><code>shutdown(3N)</code></td>
<td>shut down part of a full-duplex connection</td>
</tr>
<tr>
<td><code>sig2str(3C)</code></td>
<td>See <code>str2sig(3C)</code></td>
</tr>
<tr>
<td><code>sigaddset(3C)</code></td>
<td>See <code>sigsetops(3C)</code></td>
</tr>
<tr>
<td><code>sigblock(3B)</code></td>
<td>block signals</td>
</tr>
<tr>
<td><code>sigdelset(3C)</code></td>
<td>See <code>sigsetops(3C)</code></td>
</tr>
<tr>
<td><code>sigemptyset(3C)</code></td>
<td>See <code>sigsetops(3C)</code></td>
</tr>
<tr>
<td><code>sigfillset(3C)</code></td>
<td>See <code>sigsetops(3C)</code></td>
</tr>
<tr>
<td><code>sigfpe(3)</code></td>
<td>signal handling for specific SIGFPE codes</td>
</tr>
<tr>
<td><code>sighold(3C)</code></td>
<td>See <code>signal(3C)</code></td>
</tr>
<tr>
<td><code>sigignore(3C)</code></td>
<td>See <code>signal(3C)</code></td>
</tr>
<tr>
<td><code>siginterrupt(3B)</code></td>
<td>allow signals to interrupt functions</td>
</tr>
<tr>
<td><code>sigismember(3C)</code></td>
<td>See <code>sigsetops(3C)</code></td>
</tr>
<tr>
<td><code>siglongjmp(3C)</code></td>
<td>See <code>setjmp(3C)</code></td>
</tr>
<tr>
<td><code>sigmask(3B)</code></td>
<td>See <code>sigblock(3B)</code></td>
</tr>
<tr>
<td><code>signal(3B)</code></td>
<td>simplified software signal facilities</td>
</tr>
</tbody>
</table>

modified 30 Jun 1995
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>signal(3C)</td>
<td>simplified signal management for application processes</td>
</tr>
<tr>
<td>significand(3M)</td>
<td>See ieee_test(3M)</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 setjmp(3C)</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>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>See trig(3M)</td>
</tr>
<tr>
<td>single_to_decimal(3)</td>
<td>See floating_to_decimal(3)</td>
</tr>
<tr>
<td>sinh(3M)</td>
<td>See hyperbolic(3M)</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>
<tr>
<td>slk_attroff(3X)</td>
<td>See curs_slk(3X)</td>
</tr>
<tr>
<td>slk_attron(3X)</td>
<td>See curs_slk(3X)</td>
</tr>
<tr>
<td>slkAttrsSet(3X)</td>
<td>See curs_slk(3X)</td>
</tr>
<tr>
<td>slk_clear(3X)</td>
<td>See curs_slk(3X)</td>
</tr>
<tr>
<td>slk_init(3X)</td>
<td>See curs_slk(3X)</td>
</tr>
<tr>
<td>slk_label(3X)</td>
<td>See curs_slk(3X)</td>
</tr>
<tr>
<td>slk_noutrefresh(3X)</td>
<td>See curs_slk(3X)</td>
</tr>
<tr>
<td>slk_refresh(3X)</td>
<td>See curs_slk(3X)</td>
</tr>
<tr>
<td>slk_restore(3X)</td>
<td>See curs_slk(3X)</td>
</tr>
<tr>
<td>slk_set(3X)</td>
<td>See curs_slk(3X)</td>
</tr>
<tr>
<td>slk_touch(3X)</td>
<td>See curs_slk(3X)</td>
</tr>
<tr>
<td>socket(3N)</td>
<td>create an endpoint for communication</td>
</tr>
<tr>
<td>socketpair(3N)</td>
<td>create a pair of connected sockets</td>
</tr>
<tr>
<td>space(3N)</td>
<td>See plot(3)</td>
</tr>
<tr>
<td>spray(3N)</td>
<td>scatter data in order to test the</td>
</tr>
</tbody>
</table>
network

See printf(3B)
sprintf(3S)
See printf(3S)
sqrt(3M)
srand(3B)
See rand(3B)
srand(3C)
See rand(3C)
srand48(3C)
See drand48(3C)
srandom(3C)
See random(3C)
sscanf(3S)
See scanf(3S)
sscanf(3S)
See scanf(3S)
signal(3C)
software signals
standend(3X)
See curs_attr(3X)
standout(3X)
See curs_attr(3X)
start_color(3X)
See curs_color(3X)
stdio(3S)
standard buffered input/output package
stdipc(3C)
standard interprocess communication package
step(3G)
See regexpr(3G)
store(3B)
See dbm(3B)
str(3G)
See string(3C)
str2sig(3C)
translation between signal name and signal number
strcadd(3G)
See strcpy(3G)
strcasecmp(3C)
See string(3C)
strcat(3C)
See string(3C)
strccpy(3G)
copy strings, compressing or expanding escape codes
strchr(3C)
See string(3C)
strcmp(3C)
See string(3C)
strcoll(3C)
string collation
strcpyp(3C)
See string(3C)
strcspn(3C)
See string(3C)
strdup(3C)
See string(3C)
streadd(3G)
See strccpy(3G)
streadd(3G)
See strccpy(3G)
strepy(3G)
See string(3C)
strepy(3G)
get error message string
strerror(3C)
modified 30 Jun 1995
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>strfind(3G)</code></td>
<td>string manipulations</td>
</tr>
<tr>
<td><code>strmon(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>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>strptime(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>strrspn(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><code>strspn(3C)</code></td>
<td>See <code>string(3C)</code></td>
</tr>
<tr>
<td><code>strstr(3C)</code></td>
<td>See <code>string(3C)</code></td>
</tr>
<tr>
<td><code>strtod(3C)</code></td>
<td>convert string to double-precision number</td>
</tr>
<tr>
<td><code>strtok(3C)</code></td>
<td>See <code>string(3C)</code></td>
</tr>
<tr>
<td><code>strtok_r(3C)</code></td>
<td>See <code>string(3C)</code></td>
</tr>
<tr>
<td><code>strtoll(3C)</code></td>
<td>conversion routines</td>
</tr>
<tr>
<td><code>strtoul(3C)</code></td>
<td>See <code>strtol(3C)</code></td>
</tr>
<tr>
<td><code>strtoll(3C)</code></td>
<td>See <code>strtol(3C)</code></td>
</tr>
<tr>
<td><code>strxfrm(3C)</code></td>
<td>string transformation</td>
</tr>
<tr>
<td><code>subpad(3X)</code></td>
<td>See <code>curs_pad(3X)</code></td>
</tr>
<tr>
<td><code>subwin(3X)</code></td>
<td>See <code>curs_window(3X)</code></td>
</tr>
<tr>
<td><code>svc_auth_reg(3N)</code></td>
<td>See <code>rpc_svc_reg(3N)</code></td>
</tr>
<tr>
<td><code>svc_control(3N)</code></td>
<td>See <code>rpc_svc_create(3N)</code></td>
</tr>
<tr>
<td><code>svc_create(3N)</code></td>
<td>See <code>rpc_svc_create(3N)</code></td>
</tr>
<tr>
<td><code>svc_destroy(3N)</code></td>
<td>See <code>rpc_svc_create(3N)</code></td>
</tr>
<tr>
<td><code>svc_dg_create(3N)</code></td>
<td>See <code>rpc_svc_create(3N)</code></td>
</tr>
<tr>
<td><code>svc_dg_enablecache(3N)</code></td>
<td>See <code>rpc_svc_calls(3N)</code></td>
</tr>
</tbody>
</table>
svc_done(3N) See rpc_svc_calls(3N)
svcerr_auth(3N) See rpc_svc_err(3N)
svcerr_decode(3N) See rpc_svc_err(3N)
svcerr_noprocp(3N) See rpc_svc_err(3N)
svcerr_noprog(3N) See rpc_svc_err(3N)
svcerr_progvers(3N) See rpc_svc_err(3N)
svcerr_systemerr(3N) See rpc_svc_err(3N)
svcerr_weakauth(3N) See rpc_svc_err(3N)
svc_exit(3N) See rpc_svc_calls(3N)
svcfd_create(3N) See rpc_soc(3N)
svc_fd_create(3N) See rpc_svc_create(3N)
svc_fds(3N) See rpc_soc(3N)
svc_fdset(3N) See rpc_svc_calls(3N)
svc_freeargs(3N) See rpc_svc_calls(3N)
svc_getargs(3N) See rpc_svc_calls(3N)
svc_getcaller(3N) See rpc_soc(3N)
svc_getreq(3N) See rpc_soc(3N)
svc_getreq_common(3N) See rpc_svc_calls(3N)
svc_getreq_poll(3N) See rpc_svc_calls(3N)
svc_getreqset(3N) See rpc_svc_calls(3N)
svc_getrpccaller(3N) See rpc_svc_calls(3N)
svc_kerb_reg(3N) See kerberos_rpc(3N)
svc_pollset(3N) See rpc_svc_calls(3N)
svcraw_create(3N) See rpc_soc(3N)
svc_raw_create(3N) See rpc_svc_create(3N)
svc_reg(3N) See rpc_svc_reg(3N)
svc_register(3N) See rpc_soc(3N)
svc_run(3N) See rpc_svc_calls(3N)
svc_sendreply(3N) See rpc_svc_calls(3N)
svctcp_create(3N) See rpc_soc(3N)
svc_tli_create(3N) See rpc_svc_create(3N)
svc_tli_create(3N) See rpc_svc_create(3N)
svc_udp_bufcreate(3N) See rpc_soc(3N)
svcudp_create(3N) See rpc_soc(3N)
svcudp_create(3N) See rpc_svc_reg(3N)
**C Library Functions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>See</th>
</tr>
</thead>
<tbody>
<tr>
<td>svc_unregister(3N)</td>
<td>See rpc_soc(3N)</td>
<td></td>
</tr>
<tr>
<td>svc_vc_create(3N)</td>
<td>See rpc_svc_create(3N)</td>
<td></td>
</tr>
<tr>
<td>swab(3C)</td>
<td>swap bytes</td>
<td></td>
</tr>
<tr>
<td>swapcontext(3C)</td>
<td>See makecontext(3C)</td>
<td></td>
</tr>
<tr>
<td>syncok(3X)</td>
<td>See curs_window(3X)</td>
<td>indirect system call</td>
</tr>
<tr>
<td>syscall(3B)</td>
<td>get configurable system variables</td>
<td></td>
</tr>
<tr>
<td>syslog(3)</td>
<td>control system log</td>
<td></td>
</tr>
<tr>
<td>sysmem(3)</td>
<td>return physical memory information</td>
<td></td>
</tr>
<tr>
<td>sys_siglist(3B)</td>
<td>See psignal(3B)</td>
<td></td>
</tr>
<tr>
<td>system(3S)</td>
<td>issue a shell command</td>
<td></td>
</tr>
<tr>
<td>t_accept(3N)</td>
<td>accept a connect request</td>
<td></td>
</tr>
<tr>
<td>taddr2uaddr(3N)</td>
<td>See netdir(3N)</td>
<td>allocate a library structure</td>
</tr>
<tr>
<td>t_alloc(3N)</td>
<td>tan(3M)</td>
<td>See trig(3M)</td>
</tr>
<tr>
<td>tanh(3M)</td>
<td>See hyperbolic(3M)</td>
<td>bind an address to a transport endpoint</td>
</tr>
<tr>
<td>t_bind(3N)</td>
<td>tcdrain(3)</td>
<td>See termios(3)</td>
</tr>
<tr>
<td>tcflow(3)</td>
<td>tcflush(3)</td>
<td>See termios(3)</td>
</tr>
<tr>
<td>tcgetattr(3)</td>
<td>tcgetattr(3)</td>
<td>See termios(3)</td>
</tr>
<tr>
<td>tcgetattr(3)</td>
<td>tcgetattr(3)</td>
<td>See termios(3)</td>
</tr>
<tr>
<td>tcflush(3)</td>
<td>tcflush(3)</td>
<td>See termios(3)</td>
</tr>
<tr>
<td>tcsetattr(3)</td>
<td>tcsetattr(3)</td>
<td>See termios(3)</td>
</tr>
<tr>
<td>tcsetpgrp(3)</td>
<td>tcsetpgrp(3)</td>
<td>set foreground process group id of terminal</td>
</tr>
<tr>
<td>tcsetpgrp(3C)</td>
<td>tcsetpgrp(3C)</td>
<td></td>
</tr>
<tr>
<td>tclose(3N)</td>
<td>t_cline(3N)</td>
<td>close a transport endpoint</td>
</tr>
<tr>
<td>t_connect(3N)</td>
<td>t_connect(3N)</td>
<td></td>
</tr>
<tr>
<td>tcsendbreak(3)</td>
<td>tcsendbreak(3)</td>
<td>See termios(3)</td>
</tr>
<tr>
<td>tcsetattr(3)</td>
<td>tcsetattr(3)</td>
<td>See termios(3)</td>
</tr>
<tr>
<td>tcsetpgrp(3)</td>
<td>tcsetpgrp(3)</td>
<td>See termios(3)</td>
</tr>
<tr>
<td>tcsetpgrp(3C)</td>
<td>tcsetpgrp(3C)</td>
<td>set foreground process group id of terminal</td>
</tr>
<tr>
<td>tdelete(3C)</td>
<td>tdelete(3C)</td>
<td>See tsearch(3C)</td>
</tr>
<tr>
<td>td_init(3T)</td>
<td>td_init(3T)</td>
<td>See libthread_db(3T)</td>
</tr>
<tr>
<td>td_log(3T)</td>
<td>td_log(3T)</td>
<td>See libthread_db(3T)</td>
</tr>
<tr>
<td>td_ta_delete(3T)</td>
<td>td_ta_delete(3T)</td>
<td>See libthread_db(3T)</td>
</tr>
</tbody>
</table>
td_tamonthreads

See libthread_db

td_ta_get_ph

See libthread_db

td_ta_map_id2thr

See libthread_db

td_ta_map_lwp2thr

See libthread_db

td_ta_new

See libthread_db

td_ta_thr_iter

See libthread_db

td_ta_tsd_iter

See libthread_db

td_thr_getfpregs

See libthread_db

td_thr_getgregs

See libthread_db

td_thr_get_info

See libthread_db

td_thr_getxregs

See libthread_db

td_thr_getxregsize

See libthread_db

td_thr_setfpregs

See libthread_db

td_thr_setgregs

See libthread_db

td_thr_setprio

See libthread_db

td_thr_setsigpending

See libthread_db

td_thr_setxregs

See libthread_db

td_thr_sigsetmask

See libthread_db

td_thr_tsd

See libthread_db

td_thr_validate

See libthread_db

telldir

See directory
	
tempnam

See tmpnam

termattrs

See curs_termattrs

termios

general terminal interface

termmame

See curs_termattrs

t_error

produce error message

textdomain

See gettext

tfind

See tsearch

t_free

free a library structure
	
tgetent

See curs_termcap

tgetflag

See curs_termcap

t_getinfo

get protocol-specific service information

tgetnum

See curs_termcap

t_getstate

get the current state
tgetstr(3X)
tgoto(3X)
thr_continue(3T)
thr_create(3T)
threads(3T)

thr_exit(3T)
thr_getconcurrency(3T)
thr_getprio(3T)
thr_getspecific(3T)
thr_join(3T)
thr_keycreate(3T)
thr_kill(3T)
thr_main(3T)
thr_min_stack(3T)

thr_self(3T)
thr_setconcurrency(3T)
thr_setprio(3T)
thr_setspecific(3T)
thr_sigsetmask(3T)
thr_stksegment(3T)

thr_suspend(3T)

thr_yield(3T)
tigetflag(3X)
tigetnum(3X)
tigetstr(3X)
timeout(3X)
timer_create(3R)
timer_delete(3R)
timer_getoverrun(3R)
timer_gettime(3R)
timer_settime(3R)
times(3B)

See curs_termcap(3X)
See curs_termcap(3X)
See thr_suspend(3T)
See pthread_create(3T)
See pthread_exit(3T)
See thr_setconcurrency(3T)
See pthread_setschedparam(3T)
See pthread_key_create(3T)
See pthread_key_create(3T)
See pthread_key_create(3T)
See pthread_self(3T)
See pthread_setschedparam(3T)
See pthread_key_create(3T)
See pthread_sigmask(3T)
get thread stack bottom and stack size
suspend or continue thread execution
thread yield to another thread
See curs_terminfo(3X)
See curs_terminfo(3X)
See curs_terminfo(3X)
See curs_inopts(3X)
create a timer
delete a per-LWP timer
See timer_settime(3R)
See timer_settime(3R)
high-resolution timer operations
get process times
t_listen(3N) listen for a connect request
t_look(3N) look at the current event on a transport endpoint
tmpfile(3S) create a temporary file
tmpnam(3S) create a name for a temporary file
See tmpnam(3S)
tmpnam_r(3S) See tmpnam(3S)
TNF_DECLARE_RECORD(3X) TNF type extension interface for probes
TNF_DEFINE_RECORD_1(3X) See TNF_DECLARE_RECORD(3X)
TNF_DEFINE_RECORD_2(3X) See TNF_DECLARE_RECORD(3X)
TNF_DEFINE_RECORD_3(3X) See TNF_DECLARE_RECORD(3X)
TNF_DEFINE_RECORD_4(3X) See TNF_DECLARE_RECORD(3X)
TNF_DEFINE_RECORD_5(3X) See TNF_DECLARE_RECORD(3X)
TNF_PROBE(3X) probe insertion interface
See TNF_PROBE(3X)
TNF_PROBE_0(3X) See TNF_PROBE(3X)
TNF_PROBE_1(3X) See TNF_PROBE(3X)
TNF_PROBE_2(3X) See TNF_PROBE(3X)
TNF_PROBE_3(3X) See TNF_PROBE(3X)
TNF_PROBE_4(3X) See TNF_PROBE(3X)
TNF_PROBE_5(3X) See TNF_PROBE(3X)
tnf_process_disable(3X) probe control internal interface
tnf_process_enable(3X) See tnf_process_disable(3X)
tnf_thread_disable(3X) See tnf_process_disable(3X)
tnf_thread_enable(3X) See tnf_process_disable(3X)
toascii(3C) See conv(3C)
tolower(3C) See conv(3C)
tolower(3C) See conv(3C)
t_open(3N) establish a transport endpoint
top_panel(3X) See panel_top(3X)
top_row(3X) See menu_item_current(3X)
t_optmgmt(3N) manage options for a transport endpoint
See curs_touch(3X)
touchline(3X) See curs_touch(3X)
touchwin(3X) See conv(3C)
_toupper(3C) See conv(3C)
toupper(3C) See conv(3C)
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>See</th>
</tr>
</thead>
<tbody>
<tr>
<td>towlower</td>
<td>receive data or expedited data sent over a connection</td>
<td>wconv</td>
</tr>
<tr>
<td>towupper</td>
<td>receive the confirmation from a connect request</td>
<td>wconv</td>
</tr>
<tr>
<td>tpar (3X)</td>
<td>retrieve information from disconnect</td>
<td>curs_terminfo</td>
</tr>
<tr>
<td>tputs (3X)</td>
<td>acknowledge receipt of an orderly release indication</td>
<td>curs_termcap</td>
</tr>
<tr>
<td>tputs (3X)</td>
<td>receive a data unit</td>
<td>curs_terminfo</td>
</tr>
<tr>
<td>t puts</td>
<td>receive a unit data error indication</td>
<td></td>
</tr>
<tr>
<td>trigg (3M)</td>
<td>trigonometric functions</td>
<td></td>
</tr>
<tr>
<td>truncate</td>
<td>set a file to a specified length</td>
<td></td>
</tr>
<tr>
<td>tsearch (3C)</td>
<td>manage binary search trees</td>
<td></td>
</tr>
<tr>
<td>t_snd</td>
<td>send data or expedited data over a connection</td>
<td></td>
</tr>
<tr>
<td>t_snddis (3N)</td>
<td>send user-initiated disconnect request</td>
<td></td>
</tr>
<tr>
<td>t SNDrel (3N)</td>
<td>initiate an orderly release</td>
<td></td>
</tr>
<tr>
<td>t_snddata (3N)</td>
<td>send a data unit</td>
<td></td>
</tr>
<tr>
<td>t_s_err (3N)</td>
<td>get error message string</td>
<td></td>
</tr>
<tr>
<td>t_sync (3N)</td>
<td>synchronize transport library</td>
<td></td>
</tr>
<tr>
<td>ttyname (3C)</td>
<td>find name of a terminal</td>
<td>ttyname</td>
</tr>
<tr>
<td>ttyname_r (3C)</td>
<td>find the slot in the utmp file of the current user</td>
<td>ttyname</td>
</tr>
<tr>
<td>ttyslot (3C)</td>
<td>disable a transport endpoint</td>
<td></td>
</tr>
<tr>
<td>uaddr2taddr (3N)</td>
<td>schedule signal after interval in microseconds</td>
<td>netdir</td>
</tr>
<tr>
<td>ualarm (3C)</td>
<td>schedule signal after interval in microseconds</td>
<td></td>
</tr>
</tbody>
</table>

3-106 modified 30 Jun 1995
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>See Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>ulckpwd</td>
<td>See lckpwd(3C)</td>
<td></td>
</tr>
<tr>
<td>ulltostr</td>
<td>See strtol(3C)</td>
<td></td>
</tr>
<tr>
<td>unctrl</td>
<td>See curs_util(3X)</td>
<td></td>
</tr>
<tr>
<td>ungetc</td>
<td>push character back onto input stream</td>
<td></td>
</tr>
<tr>
<td>ungetc</td>
<td>See curs_getch(3X)</td>
<td></td>
</tr>
<tr>
<td>ungetch</td>
<td>push a Process Code character back into input stream</td>
<td></td>
</tr>
<tr>
<td>ungetwch</td>
<td>See curs_getwch(3X)</td>
<td></td>
</tr>
<tr>
<td>unlockp</td>
<td>unlock a pseudo-terminal master/slave pair</td>
<td></td>
</tr>
<tr>
<td>unordered</td>
<td>See isnan(3C)</td>
<td></td>
</tr>
<tr>
<td>unpost_form</td>
<td>See form_post(3X)</td>
<td></td>
</tr>
<tr>
<td>unpost_menu</td>
<td>See menu_post(3X)</td>
<td></td>
</tr>
<tr>
<td>un-touchwin</td>
<td>See curs_touch(3X)</td>
<td></td>
</tr>
<tr>
<td>update_panels</td>
<td>See panel_update(3X)</td>
<td></td>
</tr>
<tr>
<td>updwttmp</td>
<td>See getutxent(3C)</td>
<td></td>
</tr>
<tr>
<td>updwtmpx</td>
<td>See getutxent(3C)</td>
<td></td>
</tr>
<tr>
<td>use_env</td>
<td>See curs_util(3X)</td>
<td></td>
</tr>
<tr>
<td>user2netname</td>
<td>See secure_rpc(3N)</td>
<td></td>
</tr>
<tr>
<td>usleep</td>
<td>suspend execution for interval in microseconds</td>
<td></td>
</tr>
<tr>
<td>utmpname</td>
<td>See getutent(3C)</td>
<td></td>
</tr>
<tr>
<td>utmpxname</td>
<td>See getutxent(3C)</td>
<td></td>
</tr>
<tr>
<td>valloc</td>
<td>See malloc(3C)</td>
<td></td>
</tr>
<tr>
<td>vfprintf</td>
<td>See printf(3B)</td>
<td></td>
</tr>
<tr>
<td>vprintf</td>
<td>See printf(3S)</td>
<td></td>
</tr>
<tr>
<td>vidattr</td>
<td>See curs_terminfo(3X)</td>
<td></td>
</tr>
<tr>
<td>vidputs</td>
<td>See curs_terminfo(3X)</td>
<td></td>
</tr>
<tr>
<td>vlfmt</td>
<td>display error message in standard format and pass to logging and monitoring services</td>
<td></td>
</tr>
<tr>
<td>volmgt_check</td>
<td>have Volume Management check for media</td>
<td></td>
</tr>
<tr>
<td>volmgt_inuse</td>
<td>check whether or not Volume Management is managing a pathname</td>
<td></td>
</tr>
</tbody>
</table>

modified 30 Jun 1995

SunOS 5.5 C Library Functions Intro (3)
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>volmgt_root(3X)</td>
<td>return the Volume Management root directory</td>
</tr>
<tr>
<td>volmgt_running(3X)</td>
<td>return whether or not Volume Management is running</td>
</tr>
<tr>
<td>volmgt_symdev(3X)</td>
<td>See volmgt_symname(3X)</td>
</tr>
<tr>
<td>volmgt_symname(3X)</td>
<td>convert between Volume Management symbolic names, and the devices that correspond to them</td>
</tr>
<tr>
<td>vpfmt(3C)</td>
<td>display error message in standard format and pass to logging and monitoring services</td>
</tr>
<tr>
<td>vprintf(3B)</td>
<td>See printf(3B)</td>
</tr>
<tr>
<td>vprintf(3S)</td>
<td>print formatted output of a variable argument list</td>
</tr>
<tr>
<td>vsprintf(3B)</td>
<td>See printf(3B)</td>
</tr>
<tr>
<td>vsprintf(3S)</td>
<td>See vprintf(3S)</td>
</tr>
<tr>
<td>vsyslog(3)</td>
<td>log message with a varargs argument list</td>
</tr>
<tr>
<td>vwprintw(3X)</td>
<td>See curs_printw(3X)</td>
</tr>
<tr>
<td>vwscanw(3X)</td>
<td>See curs_scanw(3X)</td>
</tr>
<tr>
<td>waddch(3X)</td>
<td>See curs_addch(3X)</td>
</tr>
<tr>
<td>waddchnstr(3X)</td>
<td>See curs_addchstr(3X)</td>
</tr>
<tr>
<td>waddchstr(3X)</td>
<td>See curs_addchstr(3X)</td>
</tr>
<tr>
<td>waddnstr(3X)</td>
<td>See curs_addstr(3X)</td>
</tr>
<tr>
<td>waddnwstr(3X)</td>
<td>See curs_addwstr(3X)</td>
</tr>
<tr>
<td>waddstr(3X)</td>
<td>See curs_addstr(3X)</td>
</tr>
<tr>
<td>waddwch(3X)</td>
<td>See curs_addwch(3X)</td>
</tr>
<tr>
<td>waddwchnstr(3X)</td>
<td>See curs_addwchstr(3X)</td>
</tr>
<tr>
<td>waddwchstr(3X)</td>
<td>See curs_addwchstr(3X)</td>
</tr>
<tr>
<td>waddwstr(3X)</td>
<td>See curs_addwstr(3X)</td>
</tr>
<tr>
<td>wadjcurspos(3X)</td>
<td>See curs_alecompat(3X)</td>
</tr>
<tr>
<td>wait(3B)</td>
<td>wait for process to terminate or stop</td>
</tr>
<tr>
<td>wait3(3B)</td>
<td>See wait(3B)</td>
</tr>
<tr>
<td>wait3(3C)</td>
<td>wait for process to terminate or stop</td>
</tr>
<tr>
<td>wait4(3B)</td>
<td>See wait(3B)</td>
</tr>
<tr>
<td>wait4(3C)</td>
<td>See wait3(3C)</td>
</tr>
<tr>
<td>waitpid(3B)</td>
<td>See wait(3B)</td>
</tr>
</tbody>
</table>

modified 30 Jun 1995
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>watof(3I)</td>
<td>See wcstod(3I)</td>
</tr>
<tr>
<td>watoi(3I)</td>
<td>See wcstol(3I)</td>
</tr>
<tr>
<td>watol(3I)</td>
<td>See wcstol(3I)</td>
</tr>
<tr>
<td>watoll(3I)</td>
<td>See wcstol(3I)</td>
</tr>
<tr>
<td>wattoff(3X)</td>
<td>See curs_attr(3X)</td>
</tr>
<tr>
<td>watton(3X)</td>
<td>See curs_attr(3X)</td>
</tr>
<tr>
<td>watattrset(3X)</td>
<td>See curs_attr(3X)</td>
</tr>
<tr>
<td>wbkgd(3X)</td>
<td>See curs_bkgd(3X)</td>
</tr>
<tr>
<td>wbkgdset(3X)</td>
<td>See curs_bkgd(3X)</td>
</tr>
<tr>
<td>wborder(3X)</td>
<td>See curs_border(3X)</td>
</tr>
<tr>
<td>wcrlrtobot(3X)</td>
<td>See curs_clear(3X)</td>
</tr>
<tr>
<td>wcrlrtoeol(3X)</td>
<td>See curs_clear(3X)</td>
</tr>
<tr>
<td>wconv(3I)</td>
<td>Process Code character conversion macros</td>
</tr>
<tr>
<td>wcscat(3I)</td>
<td>See wcstring(3I)</td>
</tr>
<tr>
<td>wcscrh(3I)</td>
<td>See wcstring(3I)</td>
</tr>
<tr>
<td>wcscmp(3I)</td>
<td>See wcstring(3I)</td>
</tr>
<tr>
<td>wcscoll(3I)</td>
<td>wide character string comparison using collating information</td>
</tr>
<tr>
<td>wcscpy(3I)</td>
<td>See wcstring(3I)</td>
</tr>
<tr>
<td>wcscspn(3I)</td>
<td>See wcstring(3I)</td>
</tr>
<tr>
<td>wcsetno(3I)</td>
<td>See cset(3I)</td>
</tr>
<tr>
<td>wcftime(3I)</td>
<td>convert date and time to wide character string</td>
</tr>
<tr>
<td>wcslen(3I)</td>
<td>See wcstring(3I)</td>
</tr>
<tr>
<td>wcscncat(3I)</td>
<td>See wcstring(3I)</td>
</tr>
<tr>
<td>wcscncmp(3I)</td>
<td>See wcstring(3I)</td>
</tr>
<tr>
<td>wcscncpy(3I)</td>
<td>See wcstring(3I)</td>
</tr>
<tr>
<td>wcspbrk(3I)</td>
<td>See wcstring(3I)</td>
</tr>
<tr>
<td>wcsrcrhr(3I)</td>
<td>See wcstring(3I)</td>
</tr>
<tr>
<td>wcstrspn(3I)</td>
<td>See wcstring(3I)</td>
</tr>
<tr>
<td>wcstod(3I)</td>
<td>convert wide character string to double-precision number</td>
</tr>
<tr>
<td>wcstok(3I)</td>
<td>See wcstring(3I)</td>
</tr>
<tr>
<td>wcstol(3I)</td>
<td>convert wide character string to long</td>
</tr>
</tbody>
</table>
### C Library Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wcstombs(3C)</td>
<td>integer</td>
</tr>
<tr>
<td>wcstoul(3I)</td>
<td></td>
</tr>
<tr>
<td>wcstring(3I)</td>
<td></td>
</tr>
<tr>
<td>wcswcs(3I)</td>
<td></td>
</tr>
<tr>
<td>wcswidth(3I)</td>
<td></td>
</tr>
<tr>
<td>wcsxfrm(3I)</td>
<td></td>
</tr>
<tr>
<td>wctomb(3C)</td>
<td></td>
</tr>
<tr>
<td>wcctype(3I)</td>
<td></td>
</tr>
<tr>
<td>wcursyncup(3X)</td>
<td></td>
</tr>
<tr>
<td>wcwidth(3I)</td>
<td></td>
</tr>
<tr>
<td>wdelch(3X)</td>
<td></td>
</tr>
<tr>
<td>wdeleteln(3X)</td>
<td></td>
</tr>
<tr>
<td>wechochar(3X)</td>
<td></td>
</tr>
<tr>
<td>wechowchar(3X)</td>
<td></td>
</tr>
<tr>
<td>werase(3X)</td>
<td></td>
</tr>
<tr>
<td>wgetch(3X)</td>
<td></td>
</tr>
<tr>
<td>wgetnstr(3X)</td>
<td></td>
</tr>
<tr>
<td>wgetnwstr(3X)</td>
<td></td>
</tr>
<tr>
<td>wgetstr(3X)</td>
<td></td>
</tr>
<tr>
<td>wgetwch(3X)</td>
<td></td>
</tr>
<tr>
<td>wgetwstr(3X)</td>
<td></td>
</tr>
<tr>
<td>whline(3X)</td>
<td></td>
</tr>
<tr>
<td>WIFEXITED(3B)</td>
<td></td>
</tr>
<tr>
<td>WIFSIGNALED(3B)</td>
<td></td>
</tr>
<tr>
<td>WIFSTOPPED(3B)</td>
<td></td>
</tr>
<tr>
<td>winch(3X)</td>
<td></td>
</tr>
<tr>
<td>winchnstr(3X)</td>
<td></td>
</tr>
<tr>
<td>winchstr(3X)</td>
<td></td>
</tr>
<tr>
<td>windex(3I)</td>
<td></td>
</tr>
<tr>
<td>winnstr(3X)</td>
<td></td>
</tr>
<tr>
<td>winnwstr(3X)</td>
<td></td>
</tr>
<tr>
<td>winsch(3X)</td>
<td></td>
</tr>
<tr>
<td>winsdelln(3X)</td>
<td></td>
</tr>
</tbody>
</table>

3-110 modified 30 Jun 1995
<table>
<thead>
<tr>
<th>Function</th>
<th>See</th>
</tr>
</thead>
<tbody>
<tr>
<td>winsertln(3X)</td>
<td>curs_deleteln(3X)</td>
</tr>
<tr>
<td>winsnstr(3X)</td>
<td>curs_insstr(3X)</td>
</tr>
<tr>
<td>winsnwstr(3X)</td>
<td>curs_insstr(3X)</td>
</tr>
<tr>
<td>winsstr(3X)</td>
<td>curs_insstr(3X)</td>
</tr>
<tr>
<td>winswh(3X)</td>
<td>curs_inswh(3X)</td>
</tr>
<tr>
<td>winswhstr(3X)</td>
<td>curs_inswhstr(3X)</td>
</tr>
<tr>
<td>winwch(3X)</td>
<td>curs_inwch(3X)</td>
</tr>
<tr>
<td>winwchar(3X)</td>
<td>curs_in wcharstr(3X)</td>
</tr>
<tr>
<td>winwcharstr(3X)</td>
<td>curs_in wcharstr(3X)</td>
</tr>
<tr>
<td>winwstr(3X)</td>
<td>curs_inwstr(3X)</td>
</tr>
<tr>
<td>wmove(3X)</td>
<td>curs_move(3X)</td>
</tr>
<tr>
<td>wmovenextch(3X)</td>
<td>curs_alecompat(3X)</td>
</tr>
<tr>
<td>wmoveprevch(3X)</td>
<td>curs_alecompat(3X)</td>
</tr>
<tr>
<td>wnoutrefresh(3X)</td>
<td>curs_refresh(3X)</td>
</tr>
<tr>
<td>wordexp(3C)</td>
<td>curs_words(3C)</td>
</tr>
<tr>
<td>wordfree(3C)</td>
<td>curs_words(3C)</td>
</tr>
<tr>
<td>wprintw(3X)</td>
<td>curs_printw(3X)</td>
</tr>
<tr>
<td>wredrawln(3X)</td>
<td>curs_refresh(3X)</td>
</tr>
<tr>
<td>wrefresh(3X)</td>
<td>curs_refresh(3X)</td>
</tr>
<tr>
<td>wrindex(3I)</td>
<td>wcstring(3I)</td>
</tr>
<tr>
<td>write_vtoc(3X)</td>
<td>read_vtoc(3X)</td>
</tr>
<tr>
<td>wscasecmp(3I)</td>
<td>curs_words(3I)</td>
</tr>
<tr>
<td>wcscat(3I)</td>
<td>curs_words(3I)</td>
</tr>
<tr>
<td>wschr(3I)</td>
<td>wcstring(3I)</td>
</tr>
<tr>
<td>wscmp(3I)</td>
<td>wcstring(3I)</td>
</tr>
<tr>
<td>wscoll(3I)</td>
<td>wcstring(3I)</td>
</tr>
<tr>
<td>wscpyp(3I)</td>
<td>wcstring(3I)</td>
</tr>
<tr>
<td>wscr1(3X)</td>
<td>curs_scroll(3X)</td>
</tr>
<tr>
<td>wscspn(3I)</td>
<td>curs_scroll(3X)</td>
</tr>
<tr>
<td>wsdup(3I)</td>
<td>curs_words(3I)</td>
</tr>
<tr>
<td>wsetscreg(3X)</td>
<td>curs_outopts(3X)</td>
</tr>
<tr>
<td>wslen(3I)</td>
<td>wcstring(3I)</td>
</tr>
</tbody>
</table>
wsncasecmp(3I)
wsncat(3I)
wsncmp(3I)
wsncpy(3I)
wspbrk(3I)
wsprintf(3I)
wsrchr(3I)
wsscanf(3I)
wsscanf(3I)
wsspn(3I)
wstandend(3X)
wstandout(3X)
wstod(3I)
wstok(3I)
wstol(3I)
wstring(3I)
wsxfrm(3I)
wsyncdown(3X)
wsyncup(3X)
wttimeout(3X)
wtouchln(3X)
wvline(3X)
xdr(3N)
xdr_accepted_reply(3N)
xdr_admin(3N)
xdr_array(3N)
xdr_authsys_parms(3N)
xdr_authunix_parms(3N)
xdr_bool(3N)
xdr_bytes(3N)
xdr_callhdr(3N)
xdr_callmsg(3N)
xdr_char(3N)
xdr_complex(3N)

See wstring(3I)
See wcstring(3I)
See wcstring(3I)
See wcstring(3I)
See wstring(3I)
See wcstring(3I)
See wcstring(3I)
See wcstod(3I)
See wcstring(3I)
See wcstring(3I)
See wcstol(3I)
Process Code string operations
See wcxfrm(3I)
See curs_window(3X)
See curs_window(3X)
See curs_inopts(3X)
See curs_touch(3X)
See curs_border(3X)
library routines for external data representation
See rpc_xdr(3N)
library routines for external data representation
See xdr_complex(3N)
See rpc_xdr(3N)
See rpc_soc(3N)
See xdr_simple(3N)
See xdr_complex(3N)
See rpc_xdr(3N)
See rpc_xdr(3N)
See xdr_simple(3N)
library routines for external data representation
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>See Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>xdr_control</td>
<td>See xdr_admin</td>
<td>(3N)</td>
</tr>
<tr>
<td>xdr_create</td>
<td>library routines for external data representation stream creation</td>
<td>See xdr_create (3N)</td>
</tr>
<tr>
<td>xdr_destroy</td>
<td>See xdr_create (3N)</td>
<td>(3N)</td>
</tr>
<tr>
<td>xdr_double</td>
<td>See xdr_simple (3N)</td>
<td>(3N)</td>
</tr>
<tr>
<td>xdr_enum</td>
<td>See xdr_simple (3N)</td>
<td>(3N)</td>
</tr>
<tr>
<td>xdr_float</td>
<td>See xdr_simple (3N)</td>
<td>(3N)</td>
</tr>
<tr>
<td>xdr_free</td>
<td>See xdr_simple (3N)</td>
<td>(3N)</td>
</tr>
<tr>
<td>xdr_getpos</td>
<td>See xdr_admin (3N)</td>
<td>(3N)</td>
</tr>
<tr>
<td>xdr_hyper</td>
<td>See xdr_simple (3N)</td>
<td>(3N)</td>
</tr>
<tr>
<td>xdr_inline</td>
<td>See xdr_admin (3N)</td>
<td>(3N)</td>
</tr>
<tr>
<td>xdr_int</td>
<td>See xdr_simple (3N)</td>
<td>(3N)</td>
</tr>
<tr>
<td>xdr_long</td>
<td>See xdr_simple (3N)</td>
<td>(3N)</td>
</tr>
<tr>
<td>xdr_longlong_t</td>
<td>See xdr_simple (3N)</td>
<td>(3N)</td>
</tr>
<tr>
<td>xdrmem_create</td>
<td>See xdr_create (3N)</td>
<td>(3N)</td>
</tr>
<tr>
<td>xdr_opaque</td>
<td>See xdr_complex (3N)</td>
<td>(3N)</td>
</tr>
<tr>
<td>xdr_opaque_auth</td>
<td>See rpc_xdr (3N)</td>
<td>(3N)</td>
</tr>
<tr>
<td>xdr_pointer</td>
<td>See xdr_complex (3N)</td>
<td>(3N)</td>
</tr>
<tr>
<td>xdr_quadruple</td>
<td>See xdr_simple (3N)</td>
<td>(3N)</td>
</tr>
<tr>
<td>xdrrec_create</td>
<td>See xdr_create (3N)</td>
<td>(3N)</td>
</tr>
<tr>
<td>xdrrec_endofrecord</td>
<td>See xdr_admin (3N)</td>
<td>(3N)</td>
</tr>
<tr>
<td>xdrrec_eof</td>
<td>See xdr_admin (3N)</td>
<td>(3N)</td>
</tr>
<tr>
<td>xdrrec_readbytes</td>
<td>See xdr_admin (3N)</td>
<td>(3N)</td>
</tr>
<tr>
<td>xdrrec_skiprecord</td>
<td>See xdr_admin (3N)</td>
<td>(3N)</td>
</tr>
<tr>
<td>xdr_reference</td>
<td>See xdr_complex (3N)</td>
<td>(3N)</td>
</tr>
<tr>
<td>xdr_rejected_reply</td>
<td>See rpc_xdr (3N)</td>
<td>(3N)</td>
</tr>
<tr>
<td>xdr_replmsg</td>
<td>See rpc_xdr (3N)</td>
<td>(3N)</td>
</tr>
<tr>
<td>xdr_setpos</td>
<td>See xdr_admin (3N)</td>
<td>(3N)</td>
</tr>
<tr>
<td>xdr_short</td>
<td>See xdr_simple (3N)</td>
<td>(3N)</td>
</tr>
<tr>
<td>xdr_simple</td>
<td>library routines for external data representation</td>
<td>See xdr_admin (3N)</td>
</tr>
<tr>
<td>xdr_sizeof</td>
<td>See xdr_admin (3N)</td>
<td>(3N)</td>
</tr>
<tr>
<td>xdrstdio_create</td>
<td>See xdr_create (3N)</td>
<td>(3N)</td>
</tr>
<tr>
<td>xdr_string</td>
<td>See xdr_complex (3N)</td>
<td>(3N)</td>
</tr>
<tr>
<td>xdr_u_char</td>
<td>See xdr_simple (3N)</td>
<td>(3N)</td>
</tr>
<tr>
<td>xdr_u_hyper</td>
<td>See xdr_simple (3N)</td>
<td>(3N)</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>xdr_u_int</td>
<td>See xdr_simple(3N)</td>
<td></td>
</tr>
<tr>
<td>xdr_u_long</td>
<td>See xdr_simple(3N)</td>
<td></td>
</tr>
<tr>
<td>xdr_u_longlong_t</td>
<td>See xdr_simple(3N)</td>
<td></td>
</tr>
<tr>
<td>xdr_union</td>
<td>See xdr_complex(3N)</td>
<td></td>
</tr>
<tr>
<td>xdr_u_short</td>
<td>See xdr_simple(3N)</td>
<td></td>
</tr>
<tr>
<td>xdr_vector</td>
<td>See xdr_complex(3N)</td>
<td></td>
</tr>
<tr>
<td>xdr_void</td>
<td>See xdr_simple(3N)</td>
<td></td>
</tr>
<tr>
<td>xdr_wrapstring</td>
<td>See xdr_complex(3N)</td>
<td></td>
</tr>
<tr>
<td>xfn</td>
<td>overview of the XFN interface</td>
<td></td>
</tr>
<tr>
<td>xfn_attributes</td>
<td>an overview of XFN attribute operations</td>
<td></td>
</tr>
<tr>
<td>xfn_composite_names</td>
<td>XFN composite syntax: an overview of the syntax for XFN composite name</td>
<td></td>
</tr>
<tr>
<td>xfn_compound_names</td>
<td>XFN compound syntax: an overview of XFN model for compound name parsing</td>
<td></td>
</tr>
<tr>
<td>xfn_links</td>
<td>XFN links: an overview of XFN links</td>
<td></td>
</tr>
<tr>
<td>xfn_status_codes</td>
<td>descriptions of XFN status codes</td>
<td></td>
</tr>
<tr>
<td>xprt_register</td>
<td>See rpc_svc_reg(3N)</td>
<td></td>
</tr>
<tr>
<td>xprt_unregister</td>
<td>See rpc_svc_reg(3N)</td>
<td></td>
</tr>
<tr>
<td>y0</td>
<td>See bessel(3M)</td>
<td></td>
</tr>
<tr>
<td>y1</td>
<td>See bessel(3M)</td>
<td></td>
</tr>
<tr>
<td>yn</td>
<td>See bessel(3M)</td>
<td></td>
</tr>
<tr>
<td>yp_all</td>
<td>See ypclnt(3N)</td>
<td></td>
</tr>
<tr>
<td>yp_bind</td>
<td>See ypclnt(3N)</td>
<td></td>
</tr>
<tr>
<td>ypclnt</td>
<td>NIS Version 2 client interface</td>
<td></td>
</tr>
<tr>
<td>yperr_string</td>
<td>See ypclnt(3N)</td>
<td></td>
</tr>
<tr>
<td>yp_first</td>
<td>See ypclnt(3N)</td>
<td></td>
</tr>
<tr>
<td>yp_get_default_domain</td>
<td>See ypclnt(3N)</td>
<td></td>
</tr>
<tr>
<td>yp_master</td>
<td>See ypclnt(3N)</td>
<td></td>
</tr>
<tr>
<td>yp_match</td>
<td>See ypclnt(3N)</td>
<td></td>
</tr>
<tr>
<td>yp_next</td>
<td>See ypclnt(3N)</td>
<td></td>
</tr>
<tr>
<td>yp_order</td>
<td>See ypclnt(3N)</td>
<td></td>
</tr>
<tr>
<td>ypprot_err</td>
<td>See ypclnt(3N)</td>
<td></td>
</tr>
<tr>
<td>yp_unbind</td>
<td>See ypclnt(3N)</td>
<td></td>
</tr>
</tbody>
</table>
yp_update(3N) change NIS information

modified 30 Jun 1995
NAME
NOTE, _NOTE – annotate source code with info for tools

SYNOPSIS
#include <note.h>
NOTE(NoteInfo)

or
#include <sys/note.h>
_NOTE(NoteInfo)

MT-LEVEL
Safe

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 protected 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 construct)
- 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 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:

```c
#ifndef _FOO_H
#define _FOO_H
#define FOO _NOTE
#include <sys/note.h>
#endif
```

Public header files which span projects should use _NOTE rather than NOTE, since NOTE may already be used by a program which needs to include such a header file.

NoteInfo Argument

The actual NoteInfo used in an annotation should be specified by a tool that deals with program source (see the documentation for the tool to determine which annotations, if any, it understands).

`NoteInfo` must have one of the following forms:

```
NoteName
NoteName(Args)
```

where `NoteName` is simply an identifier which indicates the type of annotation, and `Args` is something defined by the tool that specifies the particular `NoteName`. The general restrictions on `Args` are that it be compatible with an ANSI C tokenizer and that unquoted parentheses be balanced (so that the end of the annotation can be determined without intimate knowledge of any particular annotation).

SEE ALSO

`note(4)`
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)

AVAILABILITY
SUNWtnfd

MT-LEVEL
MT-Safe.

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.

3X-118 modified 29 Sep 1994
Note that there is no semicolon after the `TNF_DECLARE_RECORD` interface. Having one will generate a compiler warning.

Compiling with the preprocessor option `−DNPROBE` (see `cc(1B)`), or with the preprocessor control statement `#define NPROBE` ahead of the `#include <tnf/probe.h>` statement, will stop the TNF type extension code from being compiled into the program.

**c_type**

`c_type` must be a C struct type. It is the template from which the new `tnf_type` is being created. Not all elements of the C struct need be provided in the TNF type being defined.

**tnf_type**

`tnf_type` is the name being given to the newly created type. Use of this interface uses the name space prefixed by `tnf_type`. So, if a new type called "xxx_type" is defined by a library, then the library should not use "xxx_type" as a prefix in any other symbols it defines. The policy on managing the type name space is the same as managing any other name space in a library i.e., prefix any new TNF types by the unique prefix that the rest of the symbols in the library use. This would prevent name space collisions when linking multiple libraries that define new TNF types. For example, if a library libpalloc.so uses the prefix "pal" for all symbols it defines, then it should also use the prefix "pal" for all new TNF types being defined.

**tnf_member_type_n**

`tnf_member_type_n` is the TNF type of the `n`th provided member of the C structure.

**tnf_member_name_n**

`tnf_member_name_n` is the name of the `n`th provided member of the C structure.

**EXAMPLES**

This example shows how a new TNF type is defined and used in a probe. This code is assumed to be part of a fictitious library called "libpalloc.so" which uses the prefix "pal" for all its symbols.

```c
#include <tnf/probe.h>

typedef struct pal_header {
    long size;
    char * descriptor;
    struct pal_header *next;
} pal_header_t;

TNF_DECLARE_RECORD(pal_header_t, pal_tnf_header);
TNF_DEFINE_RECORD_2(pal_header_t, pal_tnf_header,
    tnf_long, size,
    tnf_string, descriptor)

/*
 * Note: name space prefixed by pal_tnf_header should not be used by this
 * client anymore.
 */

void
```

modified 29 Sep 1994
pal_free(pal_header_t *header_p)
{
    int state;

    TNF_PROBE_2(pal_free_start, "paloc pal_free",
                "sunw%debug entering pal_free",
                tnf_long, state_var, state,
                pal_tnf_header, header_var, header_p);
    ...
}

SEE ALSO prex(1), tnfdump(1), TNF_PROBE(3X), tnf_process_disable(3X)

NOTES It is possible to make a tnf_type definition be recursive or mutually recursive e.g. a structure that uses the "next" field to point to itself (a linked list). If such a structure is sent in to a TNF_PROBE(3X), then the entire linked list will be logged to the trace file (until the "next" field is NULL). But, if the list is circular, it will result in an infinite loop. To break the recursion, either don’t include the "next" field in the tnf_type, or define the type of the "next" member as tnf_opaque.
NAME
TNF_PROBE, TNF_PROBE_0, TNF_PROBE_1, TNF_PROBE_2, TNF_PROBE_3, 
TNF_PROBE_4, TNF_PROBE_5 – probe insertion interface

SYNOPSIS
cc [ flag ... ] file ... [ -ltcfprobe ] [ 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);

AVAILABILITY
SUNWtnfd

MT-LEVEL
MT-Safe.

DESCRIPTION
This macro interface is used to insert probes into C or C++ code. Probes can be placed anywhere in these programs including .init sections, .fini sections, multi-threaded code, shared objects, and shared objects opened by dlopen(3X). Probes can be used to generate trace data for performance analysis or to write debugging output to stderr. Probes are controlled 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 describe in prex(1). The trace file can be thought of as a least recently used circular buffer. Once the file has been filled, newer events will overwrite the older ones.

modified 28 Sep 1994
Compiling with the preprocessor option −DNPROBE (see cc(1B)), or with the preprocessor control statement \#define NPROBE ahead of the \#include <tnf/probe.h> statement, will stop probes from being compiled into the program.

name

name of the probe and 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. A semicolon or single quotation mark is not allowed in this string. If any of the groups are enabled, the probe is enabled. keys cannot be a variable—it has to be an inlined string.

detail

detail is a string that consists of <attribute> <value> pairs that are each separated by a semicolon. The first word (up to a space) is considered to be the attribute and the rest of the string (up to the semicolon) is considered the value. The value is optional. Semicolons or single quotation marks are not allowed in either the attribute or the value. detail is used for two reasons—first, it can be used 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. Secondly, detail can be used 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 has to be an inlined string. For example, the detail string:

"sunw%debug entering function A; comX%exception no file; comY%func_entry; comY%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>sunw%debug</td>
<td>entering function A</td>
<td>entering &lt;or&gt; function &lt;or&gt; A</td>
</tr>
<tr>
<td>comX%exception</td>
<td>no file</td>
<td>no &lt;or&gt; file</td>
</tr>
<tr>
<td>comY%func_entry</td>
<td></td>
<td>/.* (regular expression)</td>
</tr>
<tr>
<td>comY%color</td>
<td>red blue</td>
<td>red &lt;or&gt; blue</td>
</tr>
</tbody>
</table>

Attribute names have to be prefixed by the vendor stock symbol followed by the ‘%’ character. This is to avoid collisions in the attribute name space. All attributes that do not have a ‘%’ character are reserved. These are the predefined attributes:

<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 (i.e., arg_name_n)</td>
</tr>
<tr>
<td>enable</td>
<td>off =&gt; probe is disabled. on =&gt; probe is enabled.</td>
</tr>
<tr>
<td>trace</td>
<td>off =&gt; tracing is off. on =&gt; tracing is on.</td>
</tr>
<tr>
<td>object</td>
<td>the executable or shared object that this probe is in.</td>
</tr>
</tbody>
</table>
funcs: list of probe functions connected to this probe.

arg_type_n: This is the type of the n-th argument. These are the predefined TNF types:

<table>
<thead>
<tr>
<th>tnf type</th>
<th>associated C type (and semantics)</th>
</tr>
</thead>
<tbody>
<tr>
<td>tnf_long</td>
<td>int, long</td>
</tr>
<tr>
<td>tnf_ulong</td>
<td>unsigned int, 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 TNF_DECLARE_RECORD(3X).

arg_name_n: arg_name_n is the name that the user wants associated with the n-th argument. It should not have any quotes around it and should follow the syntax guidelines for identifiers in ANSI C. The string version of arg_name_n is stored for every probe and can be accessed as the attribute 'slots' as mentioned above.

arg_value_n: arg_value_n is evaluated to yield a value to be included in the trace file. A read access is done on any variables that are in mentioned in arg_value_n. In a multi-threaded program, it is the user's responsibility to place locks around the TNF_PROBE macro if arg_value_n contains a variable that needs to be read protected.

EXAMPLES: In this example, probes are placed at the entry and exit of a function to determine how much time is spent in the function. The function entry probe also logs the arguments to the function. When a probe is encountered at run time and if it is enabled for tracing, a record is generated to a trace file. All probes log the time when it was encountered and also have a reference to a tag record which has information like the file name, line number, name, keys, and detail of the probe. The first probe work_args will also log the value of the two arguments of the probe (state and message).

```c
#include <tnf/probe.h>

int
work(int state, char *message)
{
    TNF_PROBE_2(work_start, "work_module work", "sunw%debug in function work", tnf_long, int_input, state, tnf_string, string_input, message);
    ...
    TNF_PROBE_0(work_end, "work_module work", "");
}
```

modified 28 Sep 1994
SEE ALSO  cc(1B), ld(1), prex(1), tnfdump(1), dlopen(3X), TNF_DECLARE_RECORD(3X),
         tnf_process_disable(3X)

NOTES  If attaching to a running program with prex(1) to control the probes, compile the pro-
        gram with –ltntfprobe or start the program with the environment variable LD_PRELOAD
        set to libtnfprobe.so.1 (see ld(1)). If libtnfprobe is explicitly linked in to the program, it
        has to be before libthread on the link line.
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);

MT-LEVEL
MT-Safe

DESCRIPTION
These functions are used to maintain numbers stored in base-64 ASCII characters. These
caracters 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.
NAME     abort – terminate the process abnormally

SYNOPSIS  #include <stdlib.h>
           void abort(void);

MT-LEVEL  Safe

DESCRIPTION abort() causes abnormal process termination to occur, unless the signal
             SIGABRT is being caught and the signal handler does not return. The abnormal termination processing
             includes at least the effect of fclose(3S) on all open streams and message catalogue
             descriptors, and the default actions defined for SIGABRT. The SIGABRT signal is sent to
             the calling process as if by means of the raise(3C) function with the argument SIGABRT.
             The status made available to wait(2) or waitpid(2) by abort will be that of a process ter-
             minated by the SIGABRT signal. abort will override blocking or ignoring the SIGABRT
             signal.

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

NOTES    Catching the signal is intended to provide the application writer with a portable means to
             abort processing, free from possible interference from any implementation-provided
             library functions. If SIGABRT is neither caught nor ignored, and the current directory is
             writable, a core dump may be produced.
NAME  abs, labs, llabs – return absolute value of integer

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

MT-LEVEL  MT-Safe

DESCRIPTION  abs() returns the absolute value of its int operand. labs() returns the absolute value of its long operand. llabs() returns the absolute value of its long long operand.

NOTES  In 2's-complement representation, the absolute value of the largest magnitude negative integral value is undefined.
NAME  accept – accept a connection on a socket

SYNOPSIS  cc [ flag ... ] file ... -lsocket -lssl [ library ... ]
#include <sys/types.h>
#include <sys/socket.h>
int accept(int s, struct sockaddr *addr, int *addrlen);

MT-LEVEL  Safe

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).
accept( ) 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. accept( ) uses the netconfig(4) file to determine the
STREAMS device file name associated with s. This is the device on which the connect indi-
cation 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 origi-
nal 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.

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.

accept( ) is used with connection-based socket types, currently with SOCK_STREAM.
It is possible to select(3C) or poll(2) a socket for the purpose of an accept( ) by selecting
or polling it for a read. However, this will only indicate when a connect indication is
pending; it is still necessary to call accept( ).

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

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

3N-128  modified 18 Nov 1993
<table>
<thead>
<tr>
<th>ENOSR</th>
<th>There were insufficient STREAMS resources available to complete the operation.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENOTSOCK</td>
<td>The descriptor does not reference a socket.</td>
</tr>
<tr>
<td>EOPNOTSUPP</td>
<td>The referenced socket is not of type SOCK_STREAM.</td>
</tr>
<tr>
<td>EPROTO</td>
<td>A protocol error has occurred; for example, the STREAMS protocol stack has not been initialized.</td>
</tr>
<tr>
<td>EWOULDBLOCK</td>
<td>The socket is marked as non-blocking and no connections are present to be accepted.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRP_ERROR</td>
<td>There is more than one (default) group_obj ACL entry.</td>
</tr>
<tr>
<td>USER_ERROR</td>
<td>There is more than one (default) user_obj ACL entry.</td>
</tr>
<tr>
<td>CLASS_ERROR</td>
<td>There is more than one (default) class_obj ACL entry.</td>
</tr>
<tr>
<td>OTHER_ERROR</td>
<td>There is more than one (default) other_obj ACL entry.</td>
</tr>
<tr>
<td>DUPLICATE_ERROR</td>
<td>Duplicate (default) entries of user or group.</td>
</tr>
<tr>
<td>ENTRY_ERROR</td>
<td>The entry type is invalid.</td>
</tr>
</tbody>
</table>

3-130 modified 27 Oct 1994
<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MISS_ERROR</td>
<td>Missing (default) group_obj, user_obj, class_obj, or other_obj 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  
aclsort — sort an ACL

SYNOPSIS  
#include <sys/acl.h>

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

DESCRIPTION  
aclbufp points to a buffer containing ACL entries. nentries specifies the number of ACL entries in the buffer. calclass, if non-zero, indicates that the CLASS_OBJ 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)
NAME acltomode, aclfrommode – convert an ACL to/from permission bits

SYNOPSIS

```c
#include <sys/types.h>
#include <sys/acl.h>

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

DESCRIPTION

`acltomode()` converts an ACL pointed to by `aclbufp` into 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 `OTHER_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`.

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    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 internal
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  addsev – define additional severities

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

MT-LEVEL  MT-safe

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

SEE ALSO  gettxt(3C), lfmt(3C), pfmt(3C).
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);

MT-LEVEL
Safe

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 func-
tion 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 enviroment vari-
able (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

3C-138 modified 22 Jan 1993
RETURN VALUES

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

SEE ALSO

fmtmsg(1), fmtmsg(3C), gettext(3C), printf(3S)
NAME aio_cancel – cancel asynchronous I/O request

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

MT-LEVEL MT-Safe

DESCRIPTION aio_cancel( ) attempts to cancel either one or all outstanding asynchronous I/O requests pending on the file descriptor specified by fildes. If aiodc is NULL, then all outstanding cancelable requests are canceled; otherwise, the individual request referenced by aiodc 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 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: `aio_cancel()` is not supported by this implementation.

See Also

- `aio_return(3R)`, `aio_read(3R)`

Notes

Applications compiled under Solaris 2.3 and 2.4 and using POSIX aio must be recompiled to work correctly when Solaris supports the Asynchronous Input and Output 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 aio_fsync – asynchronous file synchronization

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

MT-LEVEL MT-Safe

DESCRIPTION aio_fsync() 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. aio_return(3R) and aio_error(3R) 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 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 */
};

3R-142 modified 19 Aug 1993
If the I/O operation is successfully queued, `aio_fsync()` returns 0. Otherwise, it returns -1, and sets `errno` to indicate the error condition.

The requested asynchronous operation was not queued due to temporary resource limitations.

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

This implementation does not support synchronized I/O for this file.

A value of `op` other than `O_DSYNC` or `O_SYNC` was specified.

`aio_fsync()` is not supported by this implementation.

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 aio must be recompiled to work correctly when Solaris supports the Asynchronous Input and Output 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. It is our intention to provide support for these interfaces in future releases.
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 */
};

MT-LEVEL MT-Safe

DESCRIPTION aio_read() queues an asynchronous read request, and returns control immediately. Rather than blocking until completion, the read operation continues concurrently with other activity of the process.

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.

aio_write() queues an asynchronous write request, and returns control immediately. Rather than blocking until completion, the write operation continues concurrently with other activity of the process.

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 \texttt{aiocbp->offset} (in bytes).

These asynchronous operations are submitted at a priority equal to the calling process’ scheduling priority minus \texttt{aiocbp->aio_reqprio}.

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

**RETURN VALUES**

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

**ERRORS**

- **EAGAIN** The requested asynchronous I/O operation was not queued due to system resource limitations.
- **ENOSYS** \texttt{aio_read()} or is not supported by this implementation.
- **EBADF** If the calling function is \texttt{aio_read()}, and \texttt{aiocbp->fildes} is not a valid file descriptor open for reading. If the calling function is \texttt{aio_write()}, and \texttt{aiocbp->fildes} is not a valid file descriptor open for writing.
- **EINVAL** The file offset value implied by \texttt{aiocbp->aio_offset} would be invalid, \texttt{aiocbp->aio_reqprio} is not a valid value, or \texttt{aiocbp->aio_nbytes} is an invalid value.
- **ECANCELED** The requested I/O was canceled before the I/O completed due to an explicit \texttt{aio_cancel(3R)} request.
- **EINVAL** The file offset value implied by \texttt{aiocbp->aio_offset} would be invalid.

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

**NOTES**

For portability, the application should set \texttt{aiocb->aio_reqprio} to 0.

Applications compiled under Solaris 2.3 and 2.4 and using POSIX aio must be recompiled to work correctly when Solaris supports the Asynchronous Input and Output option.

**BUGS**

In Solaris 2.5, these functions always return -1 and set \texttt{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.

modified 19 Aug 1993
NAME aio_return, aio_error – retrieve return or error status of asynchronous I/O operation

SYNOPSIS

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_llio_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 */
};

MT-LEVEL Async-Signal-Safe

DESCRIPTION

aio_return() 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.
aio_return() should be called only once to retrieve the valid return status of a given asynchronous operation, after aio_error() has returned a value other than EINPROGRESS.

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.

3R-146 modified 19 Aug 1993
If unsuccessful, `aio_return()` or `aio_error()` return -1, and set `errno` to indicate the error condition.

**ERRORS**

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

**EXAMPLES**

```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 */
      ...
    }
  }
}
```

**SEE ALSO**

- close(2), exec(2), exit(2), fork(2), lseek(2), read(2), write(2), fsync(3C), aio_cancel(3R), aio_fsync(3R), aio_read(3R), lio_listio(3R)
| **NOTES** | Applications compiled under Solaris 2.3 and 2.4 and using POSIX aio must be recompiled to work correctly when Solaris supports the Asynchronous Input and Output 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    aio_suspend – wait for asynchronous I/O request

SYNOPSIS  cc [ flag ... ] file ... -lposix4 [ library ... ]
#include <aio.h>

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

MT-LEVEL    Async-Signal-Safe

DESCRIPTION    aio_suspend() 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 (i.e., the error status for the operation is not equal to EINPROGRESS), at the time of the call, the function returns without suspending the caller.

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.

list is an array of pointers to asynchronous I/O control blocks. nent 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 */
};

modified 19 Mar 1993 3R-149
struct timespec {
    time_t tv_sec; /* seconds */
    long tv_nsec; /* and nanoseconds */
};

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

ERRORS
EAGAIN No asynchronous I/O indicated in the list referenced by `list` completed in the time interval indicated by `timeout`.
EINTR A signal interrupted the `aio_suspend()` function. Note that, since each asynchronous I/O operation may possibly provoke a signal when it completes, this error return may be caused by the completion of one (or more) of the very I/O operations being awaited.
ENOSYS `aio_suspend()` is not supported by this implementation.

SEE ALSO `aio_fsync(3R), aio_read(3R), aio_return(3R), aio_write(3R), lio_listio(3R)`

NOTES
Applications compiled under Solaris 2.3 and 2.4 and using POSIX aio must be recompiled to work correctly when Solaris supports the Asynchronous Input and Output 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       aiocancel – cancel an asynchronous operation

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

MT-LEVEL   Unsafe

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 aiocancel() returns:
0 on success.
−1 on failure 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 below.
EINVAL     There are not any outstanding requests to cancel.

SEE ALSO   aioread(3), aiowait(3)

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

SYNOPSIS

cc [ flag ... ] file ... -laio [ library ... ]
#include <sys/asynch.h>
#include <fcntl.h>
int aioread(int fd, char *bufp, int bufs, off_t offset, int whence, aio_result_t *resultp);
int aiowrite(int fd, const char *bufp, int bufs, off_t offset, int whence, aio_result_t *resultp);

MT-LEVEL Unsafe

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 fd 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 fd.

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.

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

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

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

RETURN VALUES

aioread() and aiowrite() return:

0 on success.

−1 on failure and set errno to indicate the error.

ERRORS

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

EBADF fildes is not a valid file descriptor open for reading.

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

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

ENOMEM Memory resources are unavailable to initiate request.

SEE ALSO

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

NOTES

Passing an illegal address to bufp will result in setting errno to EFAULT only if it is detected by the application process.
NAME
aiowait – wait for completion of asynchronous I/O operation

SYNOPSIS
cc [ flag ... ] file ... -laio [ library ... ]
#include <sys/asynch.h>
#include <sys/time.h>

aio_result_t *aiowait(const struct timeval *timeout);

MT-LEVEL
Unsafe

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

ERRORS
EFAULT timeout points to an address outside the address space of the requesting pro-
cess. See NOTES below.
EINTR A signal was delivered before an asynchronous I/O operation completed.
The time limit expired.
EINVAL There are no outstanding asynchronous I/O requests.

SEE ALSO
aiocancel(3), aioread(3)

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.

3-154 modified 23 Mar 1993
NAME       assert – verify program assertion

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

MT-LEVEL    Safe

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 prepro-
          essor control statement #define NDEBUG ahead of the #include <assert.h> statement, will
          stop assertions from being compiled into the program.

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

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

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

atexit – add program termination routine

**SYNOPSIS**

```c
#include <stdlib.h>

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

**MT-LEVEL**

Safe

**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.

**SEE ALSO**

exit(3C)
NAME        au_open, au_close, au_write — construct and write audit records

SYNOPSIS   cc [ flag ... ] file ... -l bsm -lsocket -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);

MT-LEVEL    Safe.

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

DESCRIPTION au_open() allocates an audit record descriptor to which audit tokens can be written using au_write().

            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 for all calls will return a 0.

            au_open() returns −1 if a descriptor could not be allocated. au_close() and au_write() return −1 if d is not a valid descriptor or if audit(2) experienced an error. In the latter case, errno is set to indicate the error.

SEE ALSO bsmconv(1M), audit(2), auditon(2), au_preselect(3), au_to(3), free(3C)
### NAME
au_preselect – preselect an audit event

### SYNOPSIS
```
cc [flag ...] file ... -lbsm -lssocket -llnsl -lintl [ library ... ]
#include <bsm/libbsm.h>
int au_preselect(au_event_t event, au_mask_t *mask_p, int sorf, int flag);
```

### MT_LEVEL
MT-Safe.

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

### DESCRIPTION
`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.

`sorf` 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 `sorf`:

- **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.

SEE ALSO

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

NOTES

`au_preselect()` is normally called prior to constructing and writing an audit record. If the event is not preselected, the overhead of constructing and writing the record can be saved.
NAME
au_to, au_to_arg, au_to_attr, au_to_data, au_to_groups, au_to_in_addr, au_to_ipc,
au_to_ipc_perm, au_to_iport, au_to_me, au_to_opaque, au_to_path, au_to_process,
au_to_return, au_to_socket, au_to_text – create audit record tokens

SYNOPSIS
cc [ flag ... ] file ... –bsm –socket –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,
    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);

MT_LEVEL    MT-Safe.

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

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

au_to_groups( ) formats the array of 16 integers pointed to by groups into a “groups token.”
au_to_in_addr( ) formats the data pointed to by internet_addr into an “internet address token.”
au_to_ipc( ) formats the data in the id parameter into an “interprocess communications id token.”
au_to_ipc_perm( ) formats the data pointed to by perm into an “interprocess communications permission token.”
au_to_iport( ) formats the data pointed to by iport into an “ip port address token.”
au_to_me( ) collects audit information from the current process and creates a “subject token” by calling au_to_subject().
au_to_newgroups( ) formats the array of n integers pointed to by groups into a “new-groups token.”
au_to_subject( ) formats an auid (audit user ID), an euid (effective user ID), an egid (effective group ID), a ruid (real user ID), an rgid (real group ID), a pid (process ID), an sid (audit session ID), an tid (audit terminal ID), into a “subject token.”
au_to_opaque( ) formats the bytes bytes pointed to by data into an “opaque token.” The value of size must be positive.
au_to_path( ) formats the path name pointed to by path into a “path token.”
au_to_process( ) formats an auid (audit user ID), an euid (effective user ID), an egid (effective group ID), a ruid (real user ID), a rgid (real group ID), a pid (process ID), an sid (audit session ID), and a tid (audit terminal ID), into a “process token.” A process token should be used when the process is the object of an action (ie. when the process is the receiver of a signal).
au_to_return( ) formats an error number number and a return value value into a “return value token.”
au_to_socket( ) format the data pointed to by so into a “socket token.”
au_to_text( ) formats the NULL terminated string pointed to by text into a “text token.”

**RETURN VALUES**

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

modified 18 Feb 1994
<table>
<thead>
<tr>
<th>SEE ALSO</th>
<th>bsmconv(1M), au_open(3)</th>
</tr>
</thead>
</table>

3-162

modified 18 Feb 1994
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);

MT-LEVEL
MT-Safe.

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

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

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

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

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

RETURN VALUES
au_user_mask() returns:

0 Success.

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

FILES
/etc/security/audit_control contains default parameters read by the audit daemon, auditd(1M)

/etc/security/audit_user stores per-user audit event mask

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

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

modified 18 Feb 1994
NAME  basename – return the last element of a path name

SYNOPSIS  cc [ flag ... ] file ... -lgen [ library ... ]
#include <libgen.h>
char *basename(char *path);

MT-LEVEL  MT-Safe

DESCRIPTION  Given a pointer to a null-terminated character string that contains a path name, basename() returns a pointer to the last element of path. Trailing “/” characters are deleted.

If path or *path is zero, pointer to a static constant “.” is returned.

EXAMPLES  

<table>
<thead>
<tr>
<th>Input string</th>
<th>Output pointer</th>
</tr>
</thead>
<tbody>
<tr>
<td>/usr/lib</td>
<td>lib</td>
</tr>
<tr>
<td>/usr/</td>
<td>usr</td>
</tr>
<tr>
<td>/</td>
<td>/</td>
</tr>
</tbody>
</table>

SEE ALSO  basename(1), dirname(3G)

NOTES  When compiling multi-thread applications, the _REENTRANT flag must be defined on the compile line. This flag should only be used in multi-thread applications.
NAME      bessel, j0, j1, jn, y0, y1, yn – Bessel functions

SYNOPSIS cc [ flag ... ] file ... -lm [ library ... ]
#include <math.h>
double j0(double x);
double j1(double x);
double jn(int n, double x);
double y0(double x);
double y1(double x);
double yn(int n, double x);

MT-LEVEL     MT-Safe

DESCRIPTION These functions calculate Bessel functions of the first and second kinds for real arguments
and integer orders.

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

SEE ALSO exp(3M), matherr(3M)

DIAGNOSTICS In IEEE754 mode (i.e. the −xlibmee cc compilation option), the functions y0(), y1(), and
yn() have logarithmic singularities at the origin, so they treat zero and negative argu-
ments the way log() does, as described in exp(3M). Such arguments are unexceptional
for j0(), j1(), and jn().
NAME  bgets – read stream up to next delimiter

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

MT-LEVEL  MT-Safe

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 character
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, ":");

SEE ALSO  gets(3S)

NOTES  When compiling multi-thread applications, the _REENTRANT flag must be defined on
the compile line. This flag should only be used in multi-thread applications.
NAME  bind – bind a name to a socket

SYNOPSIS  cc [ flag ...] file ... –lsocket –lnsl [ library ... ]
#include <sys/types.h>
#include <sys/socket.h>
int bind(int s, const struct sockaddr *name, int namelen);

MT-LEVEL  Safe

DESCRIPTION  bind() assigns a name to an unnamed socket. When a socket is created with socket(3N), it exists in a name space (address family) but has no name assigned. bind() requests that the name pointed to by 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.

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

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

NOTES Binding a name in the UNIX domain creates a socket in the file system that must be deleted by the caller when it is no longer needed (using unlink(2)). The rules used in name binding vary between communication domains.
NAME
bsdmalloc, malloc, free, realloc – memory allocator

SYNOPSIS
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
table of free blocks for efficient allocation and coalescing of free storage. When there is
no suitable space already free, the allocation routines call sbrk(2) to get more memory
from the system. Each of the allocation routines returns a pointer to space suitably aligned
for storage of any type of object. Each returns a NULL pointer if the request cannot be completed (see DIAGNOSTICS).

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

free() releases a previously allocated block. Its argument is a pointer to a block previously allocated by malloc() or realloc().

realloc() changes the size of the block referenced 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 unable to honor a reallocation request, realloc() leaves its first argument unaltered. For backwards compatibility, realloc() accepts a pointer to a block freed since the most recent call to malloc() or realloc().

RETURN VALUES
malloc() and realloc() return a NULL pointer if there is not enough available memory.
When realloc() returns NULL, the block pointed to by ptr is left intact.

ERRORS
If malloc() 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.

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

WARNINGS
Use of libbsdmalloc renders an application non-SCD compliant.
libbsdmalloc routines are incompatible with the memory allocation routines in the standard C-library (libc): malloc(3C), alloca(3C), calloc(3C), free(3C), memalign(3C), realloc(3C), and valloc(3C).

modified 11 Feb 1993
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`.
bsearch – binary search a sorted table

#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.

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

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

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

SEE ALSO  hsearch(3C), lsearch(3C), qsort(3C), tsearch(3C)

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

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

If the number of elements in the table is less than the size reserved for the table, nel should be the lower number.
NAME
bstring, bcopy, bcmp, bzero – bit and byte string operations

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

MT-LEVEL
MT-Safe

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 separa-
tors 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.
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);
u_short htons(u_short hostshort);
u_long ntohl(u_long netlong);
u_short ntohs(u_short netshort);

MT-LEVEL Safe

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.)

SEE ALSO gethostbyname(3N), getservbyname(3N)
NAME
cancellation, pthread_cancel, pthread_setcancelstate, pthread_setcanceltype,
pthread_testcancel, pthread_cleanup_push, pthread_cleanup_pop – canceling execution of a thread

SYNOPSIS
#include <pthread.h>

int pthread_cancel(pthread_t target_thread);

int pthread_setcancelstate(int state, int *oldstate);

int pthread_setcanceltype(int type, int *oldtype);

void pthread_testcancel();

void pthread_cleanup_push(void (*handler)(void *void *arg));

void pthread_cleanup_pop(int execute);

MT-LEVEL
MT-Safe

DESCRIPTION
Thread cancellation enables a thread to terminate the execution of any other thread in the process. When the notice of cancellation is acted upon, the target thread (the thread being cancelled) is allowed to hold pending cancellation requests in several ways and to perform application-specific cleanup processing.

As a thread acquires resources around areas where it may get cancelled (i.e., before a cancellation point), it needs to push cancellation cleanup handlers along with the acquisition of these resources. The cleanup handlers release these resources and are invoked only if the thread were to be cancelled. As the thread leaves the last cancellation point before releasing a resource, it needs to pop the cleanup handler it had pushed earlier for this resource.

When a thread is cancelled, all the currently stacked cleanup handlers are executed and thread execution is terminated when the last cancellation cleanup handler returns. Its exit status of PTHREAD_CANCELED is then available to any threads joining with the cancelled target thread.

The thread’s cancellation state and type determine when a thread could get cancelled.

State

PTHREAD_CANCEL_DISABLE
All cancellation requests to the target_thread are held pending.

PTHREAD_CANCEL_ENABLE
Cancellation requests are acted upon, depending upon the thread’s cancellation type:

PTHREAD_CANCELASYNCHRONOUS
If the cancellation state is enabled, new or pending cancellation requests may be acted upon at any time.

PTHREADCANCELDEFERRED
Cancellation requests are held pending until a cancellation point (see below) is reached.

3T-176 modified 30 Jun 1995
Disabling cancellation will cause the setting of the cancellation type to be ineffective because all cancellation requests are held pending; however, when cancellation is enabled again, the new type will be in effect. The cancellation state is set to enabled, by default.

**Type**
When the cancellation state is disabled, a thread’s cancellation type is meaningless. The following cancellation types behave as follows when enabled:

- **PTHREAD_CANCEL ASYNCHRONOUS**
  Receipt of a `pthread_cancel()` call will result in an immediate cancellation.

- **PTHREAD_CANCEL_DEFERRED**
  Cancellation will not occur until the target thread reaches a cancellation point (see below). Receipt of a `pthread_cancel()` call will result in an immediate cancellation at this cancellation point.

The cancellation type is set to **PTHREAD_CANCEL_DEFERRED**, by default.

**Cancellation Points**
Cancellation begins at a point in a thread’s execution when pending cancellation requests are tested and the cancellation state is found to be enabled. This is called the cancellation point.

A cancellation point can be explicitly set by inserting a call to the `pthread_testcancel()` function.

In addition to explicit `pthread_testcancel()` cancellation points, implicit cancellation points occur at a defined list of system entry points. Typically, any call that might require a long term wait should be a cancellation point. Operations need only 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)`.

POSIX has also defined several other functions (in `libc` and `libposix4`), as implicit cancellation points. In general, these are functions in which threads may block:

- `aio_suspend(3R)`, `close(2)`, `creat(2)`, `fcntl(2)`, `fsync(3C)`, `mq_receive(3R)`, `mq_send(3R)`, `msync(3C)`, `nanosleep(3R)`, `open(2)`, `pause(2)`, `pthread_cond_timedwait(3T)`, `pthread_cancel(3T)`, `pthread_testcancel(3T)`, `read(2)`, `sem_wait(3R)`, `sigwaitinfo(3R)`, `sigsuspend(2)`, `sigtimedwait(3R)`, `sigwait(2)`, `sleep(3C)`, `system(3S)`, `tcdrain(3)`, `wait(2)`, `waitpid(2)`, and `write(2)`.

A cancellation point may also occur when a thread is executing the following functions:

- `closedir(3C)`, `ctermid(3S)`, `fclose(3S)`, `fcntl(2)`, `fflush(3S)`, `fgetc(3S)`, `fgetchar(3S)`, `fgetchar_unlocked(3S)`, `fgetwc(3C)`, `fgetwc_unlocked(3S)`, `fopen(3S)`, `fprintf(3S)`, `fputc(3S)`, `fputs(3S)`, `fread(3S)`, `freopen(3S)`, `fscanf(3S)`, `ftime(3S)`, `ftruncate(3S)`, `fwrite(3S)`, `get(3S)`, `getcwds(3S)`, `getdelim(3S)`, `getdir(3S)`, `getdtablesize(3S)`, `getenv(3S)`, `getenv(3S)`, `getenv(3S)`, `geteuid(3S)`, `getegid(3S)`, `getgid(3S)`, `getgrent_r(3C)`, `gethostname(3S)`, `gethostbyname(3S)`, `gethostbyaddr(3S)`, `gethostent(3S)`, `gethostent_r(3C)`, `gethostid(3S)`, `gethostnamex(3S)`, `gethostname(3S)`, `gethostbyname(3S)`, `gethostbyaddr(3S)`, `gethostent(3S)`, `gethostent_r(3C)`, `gethostid(3S)`, `gethostnamex(3S)`, `gethostname(3S)`, `gethostbyname(3S)`, `gethostbyaddr(3S)`, `gethostent(3S)`, `gethostent_r(3C)`, `gethostid(3S)`, `gethostnamex(3S)`, `gethostname(3S)`, `gethostbyname(3S)`, `gethostbyaddr(3S)`, `gethostent(3S)`, `gethostent_r(3C)`, `gethostid(3S)`, `gethostnamex(3S)`.

modified 30 Jun 1995
Cleanup Handling

An application should set up a cancellation cleanup handling function to restore any resources before a thread reaches a cancellation point. Specified cancellation points allow programmers to easily keep track of actions needed in a cancellation cleanup handler. A thread should only be made asynchronously cancelable when it is not in the process of acquiring or releasing resources (or locks), or otherwise, not in a difficult or impossible recover state.

When a cancellation request is acted upon, the routines in the list are invoked one-by-one in LIFO (last-in, first-out) order. When a scope’s cancellation cleanup handler is invoked, the storage for that scope remains valid.

pthread_cancel

pthread_cancel() requests that target_thread be canceled. If the target_thread’s cancellation state is enabled, the pthread_cancel() call will result in an immediate cancellation, if the target thread has the PTHREAD_CANCELASYNCHRONOUS type set. 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_threads. target_thread is terminated when the last destructor function returns.

pthread_setcancelstate

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.

A cancellation point occurs in the calling thread once the state is set if pthread_setcancelstate() is called with PTHREAD_CANCEL_ENABLE, and type is PTHREAD_CANCELASYNCHRONOUS.

pthread_setcanceltype

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.

If pthread_setcanceltype() is called with PTHREADCANCELASYNCHRONOUS, and if state is PTHREADCANCEL_ENABLE, a cancellation point is set in the calling thread after type is specified.

Legal values for state are PTHREADCANCEL_ENABLE and PTHREADCANCEL_DISABLE. Legal values for type are PTHREADCANCEL_DEFERRED and PTHREADCANCELASYNCHRONOUS. The cancellation state and type for newly created threads, including the thread in which main() was first invoked, are PTHREADCANCEL_ENABLE and PTHREADCANCEL_DEFERRED, respectively.

pthread_testcancel

pthread_testcancel() creates a cancellation point in the calling thread; it has no effect if cancellation is disabled.

pthread_cleanup_push

pthread_cleanup_push() pushes the specified cancellation cleanup handler routine, handler, onto the cancellation cleanup stack of the calling thread. When the thread exits, implicitly or explicitly, or is cancelled, its cancellation cleanup handler is popped from the cancellation cleanup stack and invoked with the argument arg. The thread acts upon
a cancellation request, or the thread calls `pthread_cleanup_pop()` with a non-zero `execute` argument.

**pthread_cleanup_pop**

`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.

If there are any calls to `pthread_cleanup_push()` or `pthread_cleanup_pop()` made without the matching call after the jump buffer is full, the effect of calling `longjmp(3C)` or `siglongjmp(3C)` is undefined.

Calls to `longjmp()` or `siglongjmp()` from within a cancellation cleanup handler is also undefined unless the jump buffer was also filled in the cancellation cleanup handler.

**RETURN VALUES**

If successful, `pthread_cancel()` `pthread_setcancelstate()` and `pthread_setcanceltype()` returns 0; otherwise, an error number is returned.

`pthread_testcancel()`, `pthread_cleanup_push()`, and `pthread_cleanup_pop()` are state-ments and do not return anything.

**ERRORS**

For each of the following conditions, `pthread_cancel()` returns the corresponding error number if the condition is detected:

- **ESRCH**
  No thread was found corresponding to that specified by the `target_thread` ID.

For each of the following conditions, `pthread_setcancelstate()` returns the corresponding error if the condition is detected:

- **EINVAL**
  The specified state is not `PTHREAD_CANCEL_ENABLE` or `PTHREAD_CANCEL_DISABLE`.

For each of the following conditions, `pthread_setcanceltype()` returns the corresponding error if the condition is detected:

- **EINVAL**
  The specified type is not `PTHREADCANCEL_DEFERRED` or `PTHREAD_CANCELASYNCHRONOUS`.

**SEE ALSO**

`condition(3T)`, `pthread_exit(3T)`, `pthread_join(3T)`, `setjmp(3C)`

**NOTES**

Please see `Intro(3)` for the notion of cancel-safety, Deferred-cancel-safety, and Asynchronous-cancel-safety. All libraries that have cancellation points but do not push/pop cancellation cleanup handlers are cancel-unsafe. If they push/pop cancellation handlers around cancellation points, they would become Deferred-cancel-safe, but could still be Asynchronous-cancel-unsafe.

In general, on Solaris, unless stated otherwise, all libraries are Asynchronous-cancel-unsafe and they may always remain so, because it may be too expensive for the common case (which is deferred cancellation) to make them Asynchronous-cancel-safe.

Libraries that do not have cancellation points are, by definition, Deferred-cancel-safe. Libraries that do have cancellation points but do not acquire any resources, such as locks or memory around these cancellation points, are also Deferred-cancel-safe. Those libraries which acquire locks and/or other resources before cancellation points are
Deferred-cancel-unsafe. Currently, there does not exist any labeling of libraries on Solaris about their cancel-safety status.

Applications can ensure cancel-safety of libraries by disabling cancellation before entering the library and restoring the old cancellation state on exit from the library.

Solaris threads do not offer this functionality.

Use of asynchronous cancellation while holding resources that need to be released may result in resource loss. Similarly, cancellation scopes may be safely manipulated (pushed and popped) only when the thread is in the deferred or disabled cancellation states.

For every push() there must be the same number of pop()s to compile the application.

EXAMPLES

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

Before f2() starts running, the newly created thread has most likely posted a cancellation on the main thread since the main thread calls thr_yield() right after creating thread2. Since cancellation has been disabled in the main thread initially, via the call to pthread_setcancelstate(), the call to f2() from main() proceeds fine with "X" being constructed at each recursive call, although the main thread has a pending cancellation.

Now, when f2() is called for the fifty-first time (i.e., when "i == 50"), f2() enables cancellation by calling pthread_setcancelstate() and then establishes a cancellation point for itself by calling pthread_testcancel().

Instead of pthread_testcancel(), there could have been a call to a cancellation point such as read(2) or write(2), which would have a similar effect (i.e., cause the caller to get cancelled at this point since there is a pending cancellation). Hence, the main() thread gets cancelled at the fifty-first iteration and then 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, and since the main thread has been cancelled, its return status is PTHREAD_CANCELED, which is obtained from the pthread_join(). This status is printed out and then 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 {

```

modified 30 Jun 1995
int x;
X(int i){x = i; printf("X(%d) constructed.0, i");
}X0{ printf("X(%d) destroyed.0, x");
}

void
free_res(void *i)
{
    printf("Freeing `%d`0, 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.0);
    }
    return "f2";
}

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

    printf("I am new thread :%d0, pthread_self()");
    pthread_cancel((pthread_t)tid);
    pthread_join((pthread_t)tid, &sts);
    printf("main thread cancelled due to %d0, sts");
    return (sts);
}
main()
{
    pthread_setcancelstate(PTHREAD_CANCEL_DISABLE, NULL);
    pthread_create(NULL, NULL, thread2, (void *)pthread_self());
    thr_yield();
    printf("Returned from %s0, f2(0));
}
**NAME**  
catgets – read a program message

**SYNOPSIS**  
```c
#include <nl_types.h>

char *catgets(nl_catd catd, int set_num, int msg_num, char *s);
```

**MT-LEVEL**  
MT-Safe

**DESCRIPTION**  
catgets() attempts to read message `msg_num`, in `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`.

**SEE ALSO**  
gencat(1), catopen(3C), setlocale(3C), gettext(3I)

*Developers Guide to Internationalization*

**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(char *name, int oflag);
    int catclose(nl_catd catd);

MT-LEVEL  MT-Safe

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 C locale, catopen() will always succeed without even looking at the implementation defined 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 catalogs 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

```
LANG=language[_[territory[.codeset]]]
```

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

```
LANG=De_A.88591
```

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

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

If the value of 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.

**RETURN VALUES**

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

`catclose()` returns 0 if successful, otherwise −1.

**SEE ALSO**

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

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

SYNOPSIS    #include <time.h>
             clock_t clock(void);

MT-LEVEL    MT-Safe

DESCRIPTION  clock( ) returns the amount of CPU time (in microseconds) used since the first call to
              clock( ) in the calling process. The time reported is the sum of the user and system times
              of the calling process and its terminated child processes for which it has executed the
              wait( ) function, the pclose( ) function, or the system( ) function.

              Dividing the value returned by clock() by the constant CLOCKS_PER_SEC, defined in the
              <time.h> header, will give the time in seconds.

SEE ALSO     times(2), wait(2), popen(3S), system(3S)

NOTES        The value returned by clock( ) is defined in microseconds for compatibility with systems
              that have CPU clocks with much higher resolution. Because of this, the value returned
              will wrap around after accumulating only 2147 seconds of CPU time (about 36 minutes).
              If the process time used is not available or cannot be represented, clock returns the value
              (clock_t) −1.
NAME  clock_settime, clock_gettime, clock_getres – high-resolution clock operations

SYNOPSIS

```c
#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 */
};
```

MT-LEVEL clock_gettime() is Async-Signal-Safe

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.

SEE ALSO

- `time(2)`, `ctime(3C)`, `timer_gettime(3R)`
Clock resolutions are implementation defined and are not settable by a process. Time values that are between two consecutive non-negative integer multiples of the resolution of the specified clock are truncated down to the smaller multiple of the resolution.
NAME
condition, 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 ... -lpthread [ 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 ... -lthread [ library ...]
#include <thread.h>
#include <synch.h>
int cond_init(cond_t *cvp, int type, int 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);

MT-LEVEL
MT-Safe

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
disposition, it uses a condition variable in conjunction with a mutex. Although a mutex is
exclusive and the code it protects is sharable (at certain moments), condition variables
enable the synchronization of differing events that share a mutex, but not necessarily
data. Several condition variables may be used by threads to signal each other when a
task is complete, which then allows the next waiting thread to take ownership of the
mutex.

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.

modified 30 Jun 1995

3T-189
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.

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.
Initializing condition variables can also be accomplished by allocating-in zeroed 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. The USYNC_PROCESS Solaris condition variable type for system-wide scope is equivalent to the POSIX condition variable

modified 30 Jun 1995

3T-191
attribute setting PTHREAD_PROCESS_SHARED. arg is ignored.

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()` 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()`.

**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 `pgm_start_time + 50,000,000`, where `pgm_start_time` is the start time of the application, or the number of nanoseconds is greater than or equal to 1,000,000,000.

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

  - **EINVAL**
    
    The time specified by `abstime` has passed.

  - **pthread_cond_timedwait()** or **cond_timedwait()** fails and returns the corresponding value if any of the following conditions are detected:

  - **ETIME**

**SEE ALSO**

`mmap(2)`, `fork(2)`, `signal(3C)`, `mutex(3T)`, `pthread_condattr_init(3T)`

**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

3T-194 modified 30 Jun 1995
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).

When cond_wait() returns the value of the condition is indeterminate and must be reevaluated.

cond_timedwait() is similar to cond_wait(), except that the calling thread will not wait for the condition to become true past the absolute time specified by abstime. Note that cond_timedwait() may continue to block as it tries to reacquire the mutex pointed to by mp, which may be locked by another thread. If abstime then cond_timedwait() returns because of a timeout, it returns the error code ETIME.
NAME
confstr – get configurable variables

SYNOPSIS
#include <unistd.h>
size_t confstr(int name, char *buf, size_t len);

MT-LEVEL
Mt-Safe

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. The implementation supports the name value of _CS_PATH, defined in <unistd.h>. It may support others.

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.

SEE ALSO
pathconf(2), sysconf(3C)
NAME    connect – initiate a connection on a socket

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

int connect(int s, struct sockaddr *name, int namelen);

MT-LEVEL    Safe

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 pathname 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.

modified 6 Jul 1995 3N-197
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.

The connection attempt was interrupted before any data arrived by the delivery of a signal.

`name.len` is not the size of a valid address for the specified address family.

An I/O error occurred while reading from or writing to the file system.

The socket is already connected.

Too many symbolic links were encountered in translating the pathname in `name`.

The network is not reachable from this host.

A component of the path prefix of the pathname in `name` does not exist.

The socket referred to by the pathname in `name` does not exist.

There were insufficient STREAMS resources available to complete the operation.

The server exited before the connection was complete.

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.

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

`s` is not a socket.

`name` is not a socket.

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).

SEE ALSO `close(2)`, `accept(3N)`, `getsockname(3N)`, `select(3C)`, `socket(3N)`
NAME
conv, toupper, tolower, _toupper, _tolower, toascii – translate characters

SYNOPSIS
#include <ctype.h>
int toupper(int c);
int tolower(int c);
int _toupper(int c);
int _tolower(int c);
int toascii(int c);

DESCRIPTION
toupper() and tolower() have as their domain the range of the function getc(): all values represented in an unsigned char and the value of the macro EOF as defined in stdio.h. If the argument of toupper() represents a lower-case letter, the result is the corresponding upper-case letter. If the argument of tolower() represents an upper-case letter, the result is the corresponding lower-case letter. All other arguments in the domain are returned unchanged.

The macros _toupper() and _tolower() accomplish the same things as toupper() and tolower(), respectively, but have restricted domains and are faster. _toupper() requires a lower-case letter as its argument; its result is the corresponding upper-case letter. _tolower() requires an upper-case letter as its argument; its result is the corresponding lower-case letter. Arguments outside the domain cause undefined results.

toascii() yields its argument with all bits turned off that are not part of a standard 7-bit ASCII character; it is intended for compatibility with other systems.

toupper(), tolower(), _toupper(), and _tolower() are affected by LC_CTYPE. In the “C” locale, or in a locale where shift information is not defined, these functions determine the case of characters according to the rules of the ASCII-coded character set. Characters outside the ASCII range of characters are returned unchanged.

SEE ALSO
cctype(3C), setlocale(3C), getc(3S), environ(5)

NOTES
toupper, tolower, _toupper, _tolower and toascii can be used safely in a multi-thread application, as long as setlocale(3C) is not being called to change the locale.
NAME        copylist – copy a file into memory

SYNOPSIS   cc [ flag ... ] file ... -lgen [ library ... ]
            #include <libgen.h>
            char *copylist(const char *filenm, off_t *szptr);

MT-LEVEL    MT-Safe

DESCRIPTION copylist() copies a list of items from a file into freshly allocated memory, replacing newlines with null characters. It expects two arguments: a pointer filenm 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.

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);
}

SEE ALSO    malloc(3C)

NOTES       When compiling multi-thread applications, the _REENTRANT flag must be defined on the compile line. This flag should only be used in multi-thread applications.
NAME

crypt, setkey, encrypt – generate encryption

SYNOPSIS

#include <crypt.h>

char *crypt(const char *key, const char *salt);
void setkey(const char *key);
void encrypt(char *block, int edflag);

MT-LEVEL

Safe

DESCRIPTION

crypt( ) is the password encryption function. It is based on a one-way encryption algo-
rithm with variations intended (among other things) to frustrate use of hardware imple-
mentations of a key search.

key is the input string to encrypt, for instance, a user’s typed password. Only the first
eight characters are used; the rest are ignored. 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, after which the input string is used as the key to repeatedly encrypt a con-
stant string. The returned value points to the encrypted input string. The first two char-
acters of the return value are the salt itself.

The setkey( ) and encrypt( ) functions provide (rather primitive) access to the actual hash-
ing algorithm. The argument of setkey( ) is a character array of length 64 containing only
the characters with numerical value 0 and 1. This string is divided into groups of 8, the
low-order bit in each group is ignored; this gives a 56-bit key that is set into the machine.
This is the key that will be used with the hashing algorithm to encrypt the string block
with the encrypt( ) function.

The block argument of encrypt( ) is a character array of length 64 containing only the char-
acters with numerical value 0 and 1. The argument array is modified in place to a similar
array representing the bits of the argument after having been subjected to the hashing
algorithm using the key set by setkey( ). The argument edflag, indicating decryption
rather than encryption, is ignored; use encrypt( ) in libcrypt( ) (see crypt(3X)) for decryp-
tion.

RETURN VALUES

If edflag is set to anything other than zero, errno will be set to ENOSYS.

SEE ALSO

login(1), passwd(1), crypt(3X), getpass(3C), passwd(4)

NOTES

The return value for crypt( ) points to static data that are overwritten by each call. In the
case of multithreaded applications, the return value is a pointer to thread specific data.

modified 22 Jan 1993
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);

MT-LEVEL MT-Safe with exceptions

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,

```c
#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,

```c
#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.

SEE ALSO setlocale(3C), euclen(3I)

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.
NAME
ctermid, ctermid_r – generate path name for controlling terminal

SYNOPSIS
#include <stdio.h>
char *ctermid(char *s);
char *ctermid_r(char *s);

MT-LEVEL
See the NOTES section of this page.

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.

SEE ALSO
ttyname(3C)

NOTES
The ctermid_r() interface is as proposed in the POSIX.4a Draft #6 document, and is sub-
ject 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.

modified 22 Jan 1993

3S-203
NAME  
cctime, 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);
char *ctime_r(const time_t *clock, char *buf, int buflen);
struct tm *localtime(const time_t *clock);
struct tm *localtime_r(const time_t *clock, struct tm *res);
struct tm *gmtime(const time_t *clock);
struct tm *gmtime_r(const time_t *clock, struct tm *res);
char *asctime(const struct tm *tm);
char *asctime_r(const struct tm *tm, char *buf, int buflen);
extern time_t timezone, altzone;
extern int daylight;
extern char *tzname[2];
void tzset(void);
void tzsetwall(void);

POSIX  
cc [ flag ... ] file ... -D_POSIX_PTHREAD_SEMANTICS [ library ... ]
char *ctime_r(const time_t *clock, char *buf,);
char *asctime_r(const struct tm *tm, char *buf,);

MT-LEVEL  
See the NOTES section of this page.

DESCRIPTION  
cctime(), localtime(), and gmtime() accept arguments of type time_t, pointed to by  
clock(), representing the time in seconds since 00:00:00 UTC, January 1, 1970. cctime()  
returns a pointer to a 26-character string as shown below. Time zone and daylight sav-  
ings corrections are made before string generation. The fields are constant width:
Fri Sep 13 00:00:00 1986

time_r() 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  
cctime_r() routine does not take a buflen parameter.

localtime() and gmtime() return pointers to tm structures (see below). localtime()  
corrects for the main time zone and possible alternate (“daylight savings”) time zone;  
gmtime() converts directly to Coordinated Universal Time (UTC), which is what the  
UNIX system uses internally.

localtime_r() and gmtime_r() have the same functionality as localtime() and gmtime()  
respectively, except that the caller must supply a buffer res to store the result.
asctime() converts a tm structure to a 26-character string, as shown in the above example, and returns a pointer to the string.

asctime_r() has the same functionality as asctime() except that the caller must supply a buffer buf with length buflen for the result to be stored. buf must be at least 26 bytes. The POSIX asctime_r() routine does not take a buflen parameter. asctime_r() 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] */
    /* for leap seconds */
int tm_min; /* minutes after the hour — [0, 59] */
int tm_hour; /* hour since midnight — [0, 23] */
int tm_mday; /* day of the month — [1, 31] */
int tm_mon; /* months since January — [0, 11] */
int tm_year; /* years since 1900 */
int tm_wday; /* days since Sunday — [0, 6] */
int tm_yday; /* days since January 1 — [0, 365] */
int tm_isdst; /* flag for alternate daylight savings time */
```

The value of tm_isdst is positive if daylight savings time is in effect, zero if daylight savings time is not in effect, and negative if the information is not available. (Previously, the value of tm_isdst was defined as non-zero if daylight savings was in effect.)

The external time_t variable altzone contains the difference, in seconds, between Coordinated Universal Time and the alternate time zone. The external variable timezone contains the difference, in seconds, between UTC and local standard time. The external variable daylight indicates whether time should reflect daylight savings time. Both timezone and altzone default to 0 (UTC). The external variable daylight is non-zero if an alternate time zone exists. The time zone names are contained in the external variable tzname, which by default is set to:

```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.

tzset() uses the contents of the environment variable TZ to override the value of the different external variables. The function tzset() is called by asctime() and may also be called by the user. See environ(5) for a description of the TZ environment variable.

Starting and ending times are relative to the current local time zone. If the alternate time zone start and end dates and the time are not provided, the days for the United States that year will be used and the time will be 2 AM. If the start and end dates are provided but the time is not provided, the time will be 2 AM. The effects of tzset() change the values of the external variables timezone, altzone, daylight, and tzname.
Note that in most installations, TZ is set to the correct value by default when the user logs on, using the local /etc/default/init file (see TIMEZONE(4)).

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

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

ERRORS ctime_r() and asctime_r() will fail if the following is true:

ERANGE length of the buffer supplied by caller is not large enough to store the result.

EXAMPLES tzset() scans the contents of the environment variable and assigns the different fields to the respective variable. For example, the most complete setting for New Jersey in 1986 could be

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. ctime(), localtime(), mktime(), and strftime() will also update these external variables as if they had called tzset() at the time specified by the time_t or struct tm value that they are converting.

FILES /usr/lib/locale/locale/LC_TIME/time

file containing locale specific date and time information

SEE ALSO time(2), getenv(3C), mktime(3C), printf(3S), putenv(3C), setlocale(3C), strftime(3C), TIMEZONE(4), environ(5)

NOTES 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 return values for ctime(), localtime(), and gmtime() point to static data whose content is overwritten by each call.

3C-206 modified 22 Aug 1995
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.

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

All interfaces on this page with an `_r` suffix are as specified in POSIX 1003.1c Draft #10.
NAME
cctype, 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);

MT-LEVEL
MT-Safe with exceptions

DESCRIPTION
These macros classify character-coded integer values. Each is a predicate returning non-zero for true, zero for false. The behavior of these macros, except isascii(), is affected by the current locale (see setlocale(3C)). To modify the behavior, change the LC_TYPE category in setlocale(), that is, setlocale(LC_CTYPE, newlocale). In the C locale, or in a locale where character type information is not defined, characters are classified according to the rules of the US-ASCII 7-bit coded character set.

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.

isalpha() tests for any character for which isupper() or islower() is true, or any character that is one of an implementation-defined set of characters for which none of iscntrl(), isdigit(), ispunct(), or isspace() is true. In the C locale, isalpha() returns true only for the characters for which isupper() or islower() is true.

isupper() tests for any character that is an upper-case letter or is one of an implementation-defined set of characters for which none of iscntrl(), isdigit(), ispunct(), isspace(), or islower() is true. In the C locale, isupper() returns true only for the characters defined as upper-case ASCII characters.
islower()  tests for any character that is a lower-case letter or is one of an
implementation-defined set of characters for which none of iscntrl(),
isdigit(), ispunct(), isspace(), or isupper() is true. In the C locale,
islower() returns true only for the characters defined as lower-case
ASCII characters.

isdigit()  tests for any decimal-digit character.
isxdigit()  tests for any hexadecimal-digit character ([0–9], [A–F] or [a–f]).
isalnum()  tests for any character for which isalpha() or isdigit() is true (letter or
digit).

isspace()  tests for any space, tab, carriage-return, newline, vertical-tab or form-
feed (standard white-space characters) or for one of an implementation-
defined set of characters for which isalnum() is false. In the C locale,
ispace() returns true only for the standard white-space characters.

ispunct()  tests for any printing character which is neither a space nor a character
for which isalnum() is true.
isprint()  tests for any printing character, including space (" ").
isgraph()  tests for any printing character, except space.
isctrl()  tests for any “control character” as defined by the character set.
isascii()  tests for any ASCII character, code between 0 and 0177 inclusive.

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

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

RETURN VALUES  If the argument to any of the character handling macros is not in the domain of the func-
tion, the result is undefined.

FILES  /usr/lib/locale/locale/LC_CTYPE

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

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

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);

MT-LEVEL
Unsafe

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 mar-
gin, an automatic newline is performed. At the bottom of the scrolling region, if scroll-
lok() 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 win-
dow. 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 attri-
butes, can be copied from one place to another using inch() and addch().) (see stand-
dout(), 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_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 ((\downarrow))</td>
</tr>
<tr>
<td>ACS_TTEE</td>
<td>+</td>
<td>top tee ((\uparrow))</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.

**SEE ALSO**

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

**NOTES**
The header `<curses.h>` automatically includes the headers `<stdio.h>` and `<unctrl.h>.

Note that `addch()`, `mvaddch()`, `mvwaddch()`, and `echochar()` may be macros.

modified 22 Jan 1993

3X-211
NAME
curs_addchstr, addchstr, addchnstr, waddchstr, waddchnstr, mvaddchstr, mvaddchnstr,
mvwaddchstr, mvwaddchnstr – add string of characters (and attributes) to a curses window

SYNOPSIS
cc [ flag ... ] file ... -lcurses [ 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);

MT-LEVEL
Unsafe

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 ele-
ments, 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.

SEE ALSO
curs_addstr(3X), curses(3X)

NOTES
The header <curses.h> automatically includes the headers <stdio.h> and <unctrl.h>.
Note that all routines except waddchnstr() and waddchstr() may be macros.

3X-212
modified 22 Jan 1993
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);

MT-LEVEL
Unsafe

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.

SEE ALSO
curs_addch(3X), curses(3X)

NOTES
The header <curses.h> automatically includes the headers <stdio.h> and <unctrl.h>. Note that all routines except waddstr() and waddnstr() may not be macros.
NAME
curs_addwch, addwch, waddwch, mvaddwch, mvwaddwch, echowchar, wechowchar —
add a 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);

MT-LEVEL
Unsafe

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(3I) 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 win-
dow. 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 winch(3X) after adding a control character does not return the control charac-
ter, 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_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.

3X-214 modified 12 Mar 1992
<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.

**SEE ALSO**

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

**NOTES**

The header file `<curses.h>` automatically includes the header files `<stdio.h>, <unctrl.h>` and `<widec.h>`.

Note that `addwch()`, `mvaddwch()`, `mvwaddwch()`, and `echowchar()` may be macros. None of these routines can use the color attribute in `chtype`.

modified 12 Mar 1992
## NAME

curs_addwchstr, addwchstr, addwchnstr, waddwchstr, waddwchnstr, mvaddwchstr, mvaddwchnstr, mvwaddwchstr, mvwaddwchnstr — add string of wchar_t characters (and attributes) to a curses window

## SYNOPSIS

```c
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);
```

## MT-LEVEL

Unsafe

## DESCRIPTION

All of these routines copy `wchstr`, which points to a string of wchar_t characters, directly into the window image structure starting at the current cursor position. The four routines with `n` as the last argument copy at most `n` elements, but no more than will fit on the line. If `n=−1` then the whole string is copied, to the maximum number that fit on the line.

The position of the window cursor is not advanced. These routines work faster than `waddnwstr(3X)` because they merely copy `wchstr` into the window image structure. On the other hand, care must be taken when using these functions because they don’t perform any kind of checking (such as for the newline character), they do not advance the current cursor position, and they truncate the string, rather than wrapping it around to the new line.

## RETURN VALUE

All routines return the integer `ERR` upon failure and an integer value other than `ERR` upon successful completion, unless otherwise noted in the preceding routine descriptions.

## SEE ALSO

curses(3X), waddnwstr(3X)

## 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 **chtpe**.
NAME
curs_addwstr, addwstr, addnwstr, waddwstr, waddnwstr, mvaddwstr, mvaddnwstr, 
mvwaddwstr, mvwaddnwstr – add a string of wchar_t characters to a curses window 
and advance cursor

SYNOPSIS
cc [ flag ... ] file ... -lcurses [ library ... ]
#include <curses.h>

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

MT-LEVEL
Unsafe

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.

SEE ALSO
curses(3X), waddwch(3X)

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.

3X-218
modified 28 Jun 1993
NAME
curs_alecompat, movenextch, wmovenextch, moveprevch, wmoveprevch, adjcurspos,
wadjcurspos – these functions are added to ALE curses library for moving the cursor by character.

SYNOPSIS
cc [ 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);

MT-LEVEL
Unsafe

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 wmo-
venextch(), 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.

SEE ALSO
curses(3X), getyx(3X)

NOTES
The header file <curses.h> automatically includes the header files <stdio.h>, <unctrl.h>
and <widec.h>.
Note that movenextch(), moveprevch(), and adjcurspos() may be macros.

modified 28 Jun 1993

3X-219
NAME

curs_attr, attroff, wattroff, attron, wattron, attrset, wattrset, standend, wstandend, standout, wstandout – curses character and window attribute control routines

SYNOPSIS

#include <curses.h>

int attroff(int attrs);
int wattroff(WINDOW *win, int attrs);
int attron(int attrs);
int wattron(WINDOW *win, int attrs);
int attrset(int attrs);
int wattrset(WINDOW *win, int attrs);
int standend(void);
int wstandend(WINDOW *win);
int standout (void);
int wstandout(WINDOW *win);

MT-LEVEL

Unsafe

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().

- **A_STANDOUT** - Best highlighting mode of the terminal.
- **A_UNDERLINE** - Underlining
- **A_REVERSE** - Reverse video
- **A_BLINK** - Blinking
- **A_DIM** - Half bright
- **A_BOLD** - Extra bright or bold
- **A_ALTCHARSET** - Alternate character set
- **A_CHARTEXT** - Bit-mask to extract a character
- **COLOR_PAIR(n)** - Color-pair number n

3X-220 modified 22 Jan 1993
The following macro is the reverse of \texttt{COLOR\_PAIR(n)}:

\begin{verbatim}
PAIR\_NUMBER(attrs) Returns the pair number associated with the \texttt{COLOR\_PAIR(n)} attribute.
\end{verbatim}

\textbf{RETURN VALUES}
These routines always return 1.

\textbf{SEE ALSO}
curs\_addch(3X), curs\_addstr(3X), curs\_printw(3X), curses(3X)

\textbf{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);

MT-LEVEL  
Unsafe

DESCRIPTION  
The beep() and flash() routines are used to signal the terminal user. The routine beep() sounds the audible alarm on the terminal, if possible; if that is not possible, it flashes the screen (visible bell), if that is possible. The routine flash() flashes the screen, and if that is not possible, sounds the audible signal. If neither signal is possible, nothing happens. Nearly all terminals have an audible signal (bell or beep), but only some can flash the screen.

RETURN VALUES  
These routines always return OK.

SEE ALSO  
curses(3X)

NOTES  
The header <curses.h> automatically includes the headers <stdio.h> and <unctrl.h>.
NAME
curs_bkgd, bkgd, bkgdset, wbkgdset, wbkgd – curses window background manipulation routines

SYNOPSIS
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);

MT-LEVEL
Unsafe

DESCRIPTION
The bkgdsets() 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).

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

NOTES
The header <curses.h> automatically includes the headers <stdio.h> and <unctrl.h>. Note that bkgdset() and bkgd() may be macros.

modified 22 Jan 1993
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);

MT-LEVEL
Unsafe

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)`.

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

NOTES
The header `<curses.h>` automatically includes the headers `<stdio.h>` and `<unctrl.h>`. Note that `border()` and `box()` may be macros.
NAME
curs_clear, erase, werase, clear, wclear, 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);

MT-LEVEL
Unsafe

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).

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

NOTES
The header <curses.h> automatically includes the headers <stdio.h> and <unctrl.h>.
Note that erase(), werase(), clear(), wclear(), clrtobot(), and clrtoeol() may be macros.

3X-226
modified 22 Jan 1993
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.

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; otherwise, it returns FALSE. This routine facilitates writing terminal-independent programs.

The color_content() routine gives users a way to find the intensity of the red, green, and blue (RGB) components in a color. It requires four arguments: the color number, and three addresses of shorts for storing the information about the amounts of red, green, and blue components in the given color. The value of the first argument must be between 0 and COLORS. The values that are stored at the addresses pointed to by the last three arguments are between 0 (no component) and 1000 (maximum amount of component).

The pair_content() routine allows users to find out what colors a given color-pair consists of. It requires three arguments: the color-pair number, and two addresses of shorts for storing the foreground and the background color numbers. The value of the first argument must be between 1 and COLOR_PAIRS−1. The values that are stored at the addresses pointed to by the second and third arguments are between 0 and COLORS.

Colors In <curses.h> the following macros are defined. These are the default colors. curses also assumes that COLOR_BLACK is the default background color for all terminals.

COLOR_BLACK
COLOR_RED
COLOR_GREEN
COLOR_YELLOW
COLOR_BLUE
COLOR_MAGENTA
COLOR_CYAN
COLOR_WHITE

RETURN VALUES All routines that return an integer return ERR upon failure and OK upon successful completion.

SEE ALSO curs_attr(3X), curs_initscr(3X), curses(3X)
| NOTES | The header `<curses.h>` automatically includes the headers `<stdio.h>` and `<unctrl.h>`. |

modified 22 Jan 1993
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);

MT-LEVEL
Unsafe

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.

SEE ALSO
curses(3X)

NOTES
The header <curses.h> automatically includes the headers <stdio.h> and <unctrl.h>.
Note that delch(), mvdelch(), and mvwdelch() may be macros.
NAME
curs_deleteln, deleteln, wdeleteln, insdelln, winsdelln, insertln, winsertln – delete and
insert lines in a curses window

SYNOPSIS
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);

MT-LEVEL
Unsafe

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.

SEE ALSO
curses(3X)

NOTES
The header <curses.h> automatically includes the headers <stdio.h> and <unctrl.h>.
Note that all but winsdelln() may be macros.
NAME  curs_getch, getch, wgetch, mvgetch, mvwgetch, ungetch – get (or push back) characters from curses terminal keyboard

SYNOPSIS  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);

MT-LEVEL  Unsafe

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_LL</td>
<td>Home down or bottom (lower left)</td>
</tr>
<tr>
<td></td>
<td>Keypad is arranged like this:</td>
</tr>
<tr>
<td></td>
<td><strong>A1</strong> up <strong>A3</strong></td>
</tr>
<tr>
<td></td>
<td><strong>C1</strong> down <strong>C3</strong></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>Begin(nning) 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>
<tr>
<td>Key Code</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>------------------------------</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>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>Suspended key</td>
</tr>
<tr>
<td>KEY_UNDO</td>
<td>Undo key</td>
</tr>
</tbody>
</table>

3X-234  
modified 22 Jan 1993
RETURN VALUES
All routines return the integer ERR upon failure. The ungetch() routine returns an
integer value other than ERR upon successful completion. The other routines return the
next input character or function key code upon successful completion.

SEE ALSO
curs_inopts(3X), curs_move(3X), curs_refresh(3X), curses(3X)

NOTES
The header <curses.h> automatically includes the headers <stdio.h> and <unctrl.h>.
Use of the escape key for a single character function is discouraged.
When using getch(), wgetch(), mvgetch(), or mvwgetch(), nocbreak mode (nocbreak())
and echo mode (echo()) should not be used at the same time. Depending on the state of
the tty driver when each character is typed, the program may produce undesirable
results.
Note that getch(), mvgetch(), and mvwgetch() may be macros.
NAME    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);

MT-LEVEL    Unsafe

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.

SEE ALSO    curs_getch(3X), curses(3X)

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);

MT-LEVEL
Unsafe

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-

modified 12 Mar 1992

SunOS 5.5  Miscellaneous Library Functions  curs_getwch (3X)
<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-238

modified 12 Mar 1992
<table>
<thead>
<tr>
<th>KEY</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_SFIN D</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_SMES SAGE</td>
<td>Shifted message key</td>
</tr>
<tr>
<td>KEY_SMOVE</td>
<td>Shifted move key</td>
</tr>
<tr>
<td>KEY_SNEX T</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>Suspend key</td>
</tr>
<tr>
<td>KEY_UNDO</td>
<td>Undo key</td>
</tr>
</tbody>
</table>

modified 12 Mar 1992
RETURN VALUE

All routines return the integer \texttt{ERR} upon failure and an integer value other than \texttt{ERR} upon successful completion.

SEE ALSO

\texttt{curses(3X)}, \texttt{curs_inopts(3X)}, \texttt{curs_move(3X)}, \texttt{wrefresh(3X)}, \texttt{terminfo(4)}

NOTES

The header file \texttt{<curses.h>} automatically includes the header files \texttt{<stdio.h>}, \texttt{<unctrl.h>}, and \texttt{<widec.h>}.

Use of the escape key by a programmer for a single character function is discouraged.

When using \texttt{getwch()}, \texttt{wgetwch()}, \texttt{mvgetwch()}, or \texttt{mvwgetwch()}, \texttt{nocbreak} mode and \texttt{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 \texttt{getwch()}, \texttt{mvgetwch()}, and \texttt{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);

MT-LEVEL Unsafe

DESCRIPTION
The effect of getwstr() is as though a series of calls to getwch(3X) were made, until a newline and carriage return is received. The resulting value is placed in the area pointed to by the wchar_t pointer 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.

SEE ALSO
curses(3X), getwch(3X)

NOTES
The header file <curses.h> automatically includes the header files <stdio.h>, <unctrl.h>, and <widec.h>.
Note that all routines except wgetnwstr() may be macros.

modified 28 Jun 1993
NAME       curs_getyx, getyx, getparyx, getbegyx, getmaxyx – get curses cursor and window coordinates

SYNOPSIS   cc [ flag ... ] file ... -lcurses [ library ... ]
            #include <curses.h>
            void getyx(WINDOW *win, int y, int x);
            void getparyx(WINDOW *win, int y, int x);
            void getbegyx(WINDOW *win, int y, int x);
            void getmaxyx(WINDOW *win, int y, int x);

MT-LEVEL   Unsafe

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 coor-
dinates 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).

SEE ALSO   curses(3X)

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);

MT-LEVEL  Unsafe

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

SEE ALSO  
curses(3X)

NOTES  
The header <curses.h> automatically includes the headers <stdio.h> and <unctrl.h>.
Note that all of these routines may be macros.
NAME

curs_inchstr, inchstr, inchnstr, winchstr, winchnstr, mvinchstr, mvinchnstr, mvwinchstr,
mvwinchnstr – get a string of characters (and attributes) from a curses window

SYNOPSIS

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 mvchinchnstr(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);

MT-LEVEL
Unsafe

DESCRIPTION

With these routines, a string of type chtype(), starting at the current cursor position in
the named window and ending at the right margin of the window, is returned. The four
functions with n as the last argument, return the string at most n characters long. Con-
stants 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.

SEE ALSO

curs_inch(3X), curses(3X)

NOTES

The header <curses.h> automatically includes the headers <stdio.h> and <unctrl.h>.
Note that all routines except winchnstr() may be macros.
NAME  
curs_initscr, initscr, newterm, endwin, isendwin, set_term, delscreen – curses screen initialization and manipulation routines

SYNOPSIS  
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);

MT-LEVEL  
Unsafe

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 douupdate() after a temporary escape causes the program to resume visual mode.

The isendwin() routine returns TRUE if endwin() has been called without any subsequent calls to wrefresh(), and FALSE otherwise.

The set_term() routine is used to switch between different terminals. The screen reference new becomes the new current terminal. The previous terminal is returned by the routine. This is the only routine which manipulates SCREEN pointers; all other routines
affect only the current terminal.
The `delscreen()` routine frees storage associated with the SCREEN data structure. The `endwin()` routine does not do this, so `delscreen()` should be called after `endwin()` if a particular SCREEN is no longer needed.

**RETURN VALUES**
- `endwin()` returns the integer ERR upon failure and OK upon successful completion. Routines that return pointers always return NULL on error.

**SEE ALSO**
- `curs_kernel(3X)`, `curs_refresh(3X)`, `curs_slk(3X)`, `curs_util(3X)`, `curses(3X)`

**NOTES**
The header `<curses.h>` automatically includes the headers `<stdio.h>` and `<unctrl.h>`. Note that `initscr()` and `newterm()` may be macros.
NAME
curs_inopts, cbreak, nocbreak, echo, noecho, halfdelay, intrflush, keypad, meta, nodelay,
notimeout, raw, noraw, noqiflush, qiflush, timeout, wtimeout, typeahead – curses terminal
input option control routines

SYNOPSIS
cc [ flag ... ] file ... -lcurses [ library ... ]
#include <curses.h>
int cbreak(void);
int nocbreak(void);
int echo(void);
int noecho(void);
int halfdelay(int tenths);
int intrflush(WINDOW *win, bool bf);
int keypad(WINDOW *win, bool bf);
int meta(WINDOW *win, bool bf);
int nodelay(WINDOW *win, bool bf);
int notimeout(WINDOW *win, bool bf);
int raw(void);
int noraw(void);
void noqiflush(void);
void qiflush(void);
void timeout(int delay);
void wtimeout(WINDOW *win, int delay);
int typeahead(int fildes);

MT-LEVEL Unsafe

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 in-
itially getch() is in echo mode, so characters typed are echoed. Authors of most
interactive programs prefer to do their own echoing in a controlled area of the screen, or
not to echo at all, so they disable echoing by calling \texttt{noecho()}. (See \texttt{curs_getch(3X)} for a
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 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 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 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 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 false.

Initially, whether the terminal returns 7 or 8 significant bits on input depends on the con-
trol 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} (meta\_on) and \texttt{rmm}
(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 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 associ-
ated 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 `fildes` is to be used to check for typeahead instead. If `fildes` is −1, then no typeahead checking is done.

### RETURN VALUES

All routines that return an integer return `ERR` upon failure and an integer value other than `ERR` upon successful completion, unless otherwise noted in the preceding routine descriptions.

### SEE ALSO

`curs_getch(3X)`, `curs_initscr(3X)`, `curses(3X)`, `termio(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.

modified 22 Jan 1993
NAME  curs_insch, insch, winsch, mvinsch, mvwinsch – insert a character before the character under the cursor in a curses window

SYNOPSIS  cc [ 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);

MT-LEVEL  Unsafe

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.

SEE ALSO  curses(3X)

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
c [ 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);

MT-LEVEL
Unsafe

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.

SEE ALSO
curs_clear(3X), curs_inch(3X), curses(3X)

NOTES
The header <curses.h> automatically includes the headers <stdio.h> and <unctrl.h>.
Note that all but winsnstr() may be macros.

modified 22 Jan 1993
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);

MT-LEVEL Unsafe

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.

SEE ALSO curses(3X)

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);

MT-LEVEL  
Unsafe

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.

SEE ALSO  
curses(3X)

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.
<table>
<thead>
<tr>
<th>NAME</th>
<th>curs_inswstr, inswstr, insnwstr, winswstr, winsnwstr, mvinswstr, mvinsnwstr, mvwinswstr, mvwinsnwstr – insert wchar_t string before character under the cursor in a curses window</th>
</tr>
</thead>
</table>
| 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); |
| MT-LEVEL | Unsafe |
| 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 character (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. |
| SEE ALSO | clrtoeol(3X), curses(3X), winwch(3X) |
| NOTES | The header file <curses.h> automatically includes the header files <stdio.h>, <unctrl.h> and <widec.h>. |

3X-254 modified 28 Jun 1993
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);

MT-LEVEL  Unsafe

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

SEE ALSO  
curses(3X)

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);

MT-LEVEL
Unsafe

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.

SEE ALSO
curses(3X), curs_inwch(3X)

NOTES
The header file <curses.h> automatically includes the header files <stdio.h>, <unctrl.h>,
and <widec.h>.
Note that all routines except winwchnstr() may be macros.
None of these routines can use the color attribute in chtype.
NAME

curs_inwstr, inwstr, innwstr, winwstr, winnwwstr, mvwinwstr, mvinnwstr, mvwinwwstr,
mvwinnwstr – get a string of wchar_t characters from a curses window

SYNOPSIS

cc [ flag ... ] file ... -lcurses [ library ... ]
#include <curses.h>

int inwstr(wchar_t *wstr);
int innwstr(wchar_t *wstr, int n);
int winwstr(WINDOW *win, wchar_t *wstr);
int 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);

MT-LEVEL

Unsafe

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.

SEE ALSO

curses(3X)

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);

MT-LEVEL
Unsafe

DESCRIPTION
The following routines give low-level access to various curses functionality. These routines 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 “program” (in curses) or “shell” (out of curses) state. These are done automatically by endwin() and, after an endwin(), by dupdate(), 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()`.

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.

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

**NOTES**
The header `<curses.h>` automatically includes the headers `<stdio.h>` and `<unctrl.h>`,

Note that `getsyx()` is a macro, so an ampersand (`&`) is not necessary before the variables `y` and `x`. 

---

modified 22 Jan 1993
NAME
curs_move, move, wmove – move curses window cursor

SYNOPSIS
#include <curses.h>
int move(int y, int x);
int wmove(WINDOW *win, int y, int x);

MT-LEVEL
Unsafe

DESCRIPTION
With these routines, the cursor associated with the window is moved to line 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.

SEE ALSO
curs_refresh(3X), curses(3X)

NOTES
The header <curses.h> automatically includes the headers <stdio.h> and <unctrl.h>.
Note that move() may be a macro.
NAME
curs_outopts, clearok, idlok, idcok, immedok, leaveok, setscrreg, wsetscrreg, scrollok, nl, nonl – curses terminal output option control routines

SYNOPSIS
c [ 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 setsrcreg(int top, int bot);
int wsetsrcreg(WINDOW *win, int top, int bot);
int scrollok(WINDOW *win, bool bf);
int nl(void);
int nonl(void);

MT-LEVEL Unsafe

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 setscrreg() and wsetscrreg() routines allow the application programmer to set a software scrolling region in a window. top and bot are the line numbers of the top and bottom margin of the scrolling region. (Line 0 is the top line of the window.) If this option and scrollok() are enabled, an attempt to move off the bottom margin line causes all lines in the scrolling region to scroll up one line. Only the text of the window is scrolled. (Note that this has nothing to do with the use of a physical scrolling region capability in the terminal, like that in the VT100. If idlok() is enabled and the terminal has either a scrolling region or insert/delete line capability, they will probably be used by the output routines.)

The scrollok() option controls what happens when the cursor of a window is moved off the edge of the window or scrolling region, either as a result of a newline action on the bottom line, or typing the last character of the last line. If disabled, (bf is FALSE), the cursor is left on the bottom line. If enabled, (bf is TRUE), wrefresh() is called on the window, and the physical terminal and window are scrolled up one line. (Note that in order to get the physical scrolling effect on the terminal, it is also necessary to call idlok().)

The nl() and nonl() routines control whether newline is translated into carriage return and linefeed on output, and whether return is translated into newline on input. Initially, the translations do occur. By disabling these translations using nonl(), curses is able to make better use of the linefeed capability, resulting in faster cursor motion.

RETURN VALUES
setscrreg() and wsetscrreg() return OK upon success and ERR upon failure. All other routines that return an integer always return OK.

SEE ALSO
curs_addch(3X), curs_clear(3X), curs_initscr(3X), curs_refresh(3X), curs_scroll(3X), curses(3X)

NOTES
The header <curses.h> automatically includes the headers <stdio.h> and <unctrl.h>.
Note that clearok(), leaveok(), scrollok(), idcok(), nl(), nonl(), and setscrreg() may be macros.
The immedok() routine is useful for windows that are used as terminal emulators.

modified 22 Jan 1993
NAME  | curs_overlay, overlay, overwrite, copywin – overlap and manipulate overlapped curses windows

SYNOPSIS  | cc [ flag ... ] file ... -lcurses [ library ... ]
| #include <curses.h>
| int overlay(WINDOW *srcwin, WINDOW *dstwin);
| int overwrite(WINDOW *srcwin, WINDOW *dstwin);
| int copywin(WINDOW *srcwin, WINDOW *dstwin, int sminrow, int smincol, int dminrow, int dmincol, int dmaxrow, int dmaxcol, int overlay);

MT-LEVEL  | Unsafe

DESCRIPTION  | The overlay() and overwrite() routines overlay srcwin on top of dstwin. srcwin and dstwin are not required to be the same size; only text where the two windows overlap is copied. The difference is that overlay() is non-destructive (blanks are not copied) whereas overwrite() is destructive.
| The copywin() routine provides a finer granularity of control over the overlay() and overwrite() routines. Like in the prefresh() routine, a rectangle is specified in the destination window, (dminrow, dmincol) and (dmaxrow, dmaxcol), and the upper-left-corner coordinates of the source window, (sminrow, smincol). If the argument overlay is true, then copying is non-destructive, as in overlay().

RETURN VALUES  | Routines that return an integer return ERR upon failure and an integer value other than ERR upon successful completion.

SEE ALSO  | curs_pad(3X), curs_refresh(3X), curses(3X)

NOTES  | The header <curses.h> automatically includes the headers <stdio.h> and <unctrl.h>. Note that overlay() and overwrite may be macros.
NAME
curs_pad, newpad, subpad, prefresh, pnoutrefresh, pechochar, pechowchar – create and display curses pads

SYNOPSIS
cc [ flag ...] file ... -lcurses [ library ... ]
#include <curses.h>
WINDOW *newpad(int nlines, int ncols);
WINDOW *subpad(WINDOW *orig, int nlines, int ncols, int begin_y, int begin_x);
it int prefresh(WINDOW *pad, int pminrow, int pmincol, int sminrow, int smincol,
 int smaxrow, int smaxcol);
it int pnoutrefresh(WINDOW *pad, int pminrow, int pmincol, int sminrow, int smincol,
 int smaxrow, int smaxcol);
it int pechochar(WINDOW *pad, chtype ch);
it int pechowchar(WINDOW *pad, chtype wch);

MT-LEVEL
Unsafe

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, 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.

modified 22 Jan 1993
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.

**SEE ALSO**

`addch(3X)`, `curses(3X)`, `refresh(3X)`, `subwin(3X)`, `touchline(3X)`, `touchwin(3X)`, `waddch(3X)`, `wnoutrefresh(3X)`, `wrefresh(3X)`

**NOTES**

The header `<curses.h>` automatically includes the headers `<stdio.h>`, `<unctrl.h>` and `<widec.h>`. Note that `pechochar()` may be a macro.
NAME
curs_printw, printw, wprintw, mvprintw, mvwprintw, vwprintw – print formatted output in curses windows

SYNOPSIS
cc [ flag ...] file ... -lcurses [ library ... ]
#include <curses.h>
int printw(char *fmt, l* arg */ ...);
int wprintw(WINDOW *win, char *fmt, l* arg */ ...);
int mvprintw(int y, int x, char *fmt, l* arg */ ...);
int mvwprintw(WINDOW *win, int y, int x, char *fmt, l* arg */ ...);
#include <varargs.h>
int vwprintw(WINDOW *win, char *fmt, l* varglist */ ...);

MT-LEVEL
Unsafe

DESCRIPTION
The printf(), 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.

SEE ALSO
curses(3X), printf(3S), vprintf(3S)

NOTES
The header <curses.h> automatically includes the headers <stdio.h> and <unctrl.h>.

modified 22 Jan 1993
NAME

curs_refresh, refresh, wrefresh, wnoutrefresh, douupdate, redrawwin, wredrawln –-refresh curses windows and lines

SYNOPSIS

cc [flag ...] file ... –lcurses [library ...]
#include <curses.h>
int refresh(void);
int wrefresh(WINDOW *win);
int wnoutrefresh(WINDOW *win);
int douupdate(void);
int redrawwin(WINDOW *win);
int wredrawln(WINDOW *win, int beg_line, int num_lines);

MT-LEVEL
Unsafe

DESCRIPTION

The refresh() and wrefresh() routines (or wnoutrefresh() and douupdate()) 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 douupdate() 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() works by first calling wnoutrefresh(), which copies the named window to the virtual screen, and then calling douupdate(), 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 douupdate(), causing several bursts of output to the screen. By first calling wnoutrefresh() for each window, it is then possible to call douupdate() 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.

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

NOTES
The header <curses.h> automatically includes the headers <stdio.h> and <unctrl.h>. Note that refresh() and redrawwin() may be macros.
NAME
curs_scanw, scanw, wscanw, mvscanw, mvwscanw, vwscanw – convert formatted input
from a curses widow

SYNOPSIS
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);

MT-LEVEL Unsafe

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
defined 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.

SEE ALSO curs_getstr(3X), curs_printw(3X), curses(3X), scanf(3S)

NOTES The header <curses.h> automatically includes the headers <stdio.h> and <unctrl.h>.
NAME
curs_scr_dump, scr_dump, scr_restore, scr_init, scr_set – read (write) a curses screen from (to) a file

SYNOPSIS
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);

MT-LEVEL
Unsafe

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 douplex() 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 program 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.

SEE ALSO
curs_initscr(3X), curs_refresh(3X), curs_util(3X), curses(3X), system(3S)

NOTES
The header <curses.h> automatically includes the headers <stdio.h> and <unctrl.h>
Note that scr_init(), scr_set(), and scr_restore() may be macros.

modified 22 Jan 1993
NAME    curs_scroll, 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);

MT-LEVEL    Unsafe

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.

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

NOTES    The header <curses.h> automatically includes the headers <stdio.h> and <unctrl.h>.
Note that scrl() and scroll() may be macros.
NAME   curs_slk, slk_init, slk_set, slk_refresh, slk_noutrefresh, slk_label, slk_clear, slk_restore, slk_touch, slk_attron, slk_attrset, slk_attroff — curses soft label routines

SYNOPSIS  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);

MT-LEVEL Unsafe

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, centered, or right-justified, respectively, within the label.

The slk_refresh() and slk_noutrefresh() routines correspond to the wrefresh() and wnoutrefresh() routines.

With the slk_label() routine, the current label for label number labnum is returned with leading and trailing blanks stripped.

With the slk_clear() routine, the soft labels are cleared from the screen.

With the slk_restore() routine, the soft labels are restored to the screen after a slk_clear() is performed.
With the slk_touch() routine, all the soft labels are forced to be output the next time a slk_noutrefresh() is performed.
The slk_attron(), slk_attrset(), and slk_attoff() 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.

SEE ALSO
curs_attr(3X), curs_initscr(3X), curs_refresh(3X), curses(3X)

NOTES
The header <curses.h> automatically includes the headers <stdio.h> and <unctrl.h>. Most applications would use slk_noutrefresh() because a wrefresh() is likely to follow soon.
NAME       curs_termattrs, baudrate, erasechar, has_ic, has_il, killchar, longname, termattrs, termname – curses environment query routines

SYNOPSIS  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);
cttype termattrs(void);
char *termname(void);

MT-LEVEL  Unsafe

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.

modified 22 Jan 1993
SEE ALSO  
curs_initcr(3X), curs_outopts(3X), curses(3X)

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
c [ 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));

MT-LEVEL
Unsafe

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.

SEE ALSO
curs_terminfo(3X), curses(3X), putc(3S)

NOTES
The header <curses.h> automatically includes the headers <stdio.h> and <unctrl.h>.
NAME
curs_terminfo, setupterm, setterm, set_curterm, del_curterm, restartterm, tparm, tputs, putp, vidputs, vidattr, mvcur, tigetflag, tigetnum, tigetstr – curses interfaces to terminfo database

SYNOPSIS
cc [ −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));
int putp(char *str);
int vidputs(chtype attrs, int (*putc)(char));
int vidattr(chtype attrs);
int mvcur(int oldrow, int oldcol, int newrow, int newcol);
int tigetflag(char *capname);
int tigetnum(char *capname);
int tigetstr(char *capname);

MT-LEVEL
Unsafe

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 recommended.

Initially, setupterm() should be called. Note that setupterm() is automatically called by initscr() and newterm(). This defines the set of terminal-dependent variables (listed in terminfo(4)). The terminfo variables lines and columns are initialized by setupterm() as follows: If use_env(FALSE) has been called, values for lines and columns specified in terminfo are used. Otherwise, if the environment variables LINES and COLUMNS exist, their values are used. If these environment variables do not exist and the program is running in a window, the current window size is used. Otherwise, if the environment variables do not exist, the values for lines and columns specified in the terminfo database are used.

The headers <curses.h> and <term.h> should be included (in this order) to get the definitions for these strings, numbers, and flags. Parameterized strings should be passed through tparm() to instantiate them. All terminfo strings (including the output of
tparm() should be printed with puts() or putp(). Call the reset_shell_mode() routine to restore the tty modes before exiting (see curs_kernel(3X)). Programs which use cursor addressing should output enter_ca_mode upon startup and should output exit_ca_mode before exiting. Programs desiring shell escapes should call reset_shell_mode() and output exit_ca_mode before the shell is called and should output enter_ca_mode and call reset_prog_mode after returning from the shell.

The setupterm() routine reads in the terminfo database, initializing the terminfo structures, but does not set up the output virtualization structures used by curses. The terminal type is the character string term; if term is null, the environment variable TERM is used. All output is to file descriptor fildes which is initialized for output. If errret is not null, then setupterm() returns OK or ERR and stores a status value in the integer pointed to by errret. A status of 1 in errret is normal, 0 means that the terminal could not be found, and −1 means that the terminfo database could not be found. If errret is null, setupterm() prints an error message upon finding an error and exits. Thus, the simplest call is:

```
setupterm((char **)0, 1, (int **)0);
```

which uses all the defaults and sends the output to stdout.

The setterm() routine is being replaced by setupterm(). The call:

```
setupterm(term, 1, (int **)0)
```

provides the same functionality as setterm(term). The setterm() routine is included here for compatibility and is supported at Level 2.

The set_curterm() routine sets the variable cur_term to nterm, and makes all of the terminfo boolean, numeric, and string variables use the values from nterm.

The del_curterm() routine frees the space pointed to by oterm and makes it available for further use. If oterm is the same as cur_term, references to any of the terminfo boolean, numeric, and string variables thereafter may refer to invalid memory locations until another setupterm() has been called.

The restartterm() routine is similar to setupterm() and initscr(), except that it is called after restoring memory to a previous state. It assumes that the windows and the input and output options are the same as when memory was saved, but the terminal type and baud rate may be different.

The tparm() routine instantiates the string str with parameters pi. A pointer is returned to the result of str with the parameters applied.

The tputs() routine applies padding information to the string str and outputs it. The str must be a terminfo string variable or the return value from tparm(), tgetstr(), or tgoto(). affcnt is the number of lines affected, or 1 if not applicable. putc is a putchar()-like routine to which the characters are passed, one at a time.

The putp() routine calls tputs(str, 1, putchar). Note that the output of putpA() always goes to stdout, not to the fildes 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 *boolenames, *boolcodes, *boolfnames`  
`char *numnames, *numcodes, *numfnames`  
`char *strnames, *strcodes, *strfnames`  
These null-terminated arrays contain the `capnames`, the `termcap` codes, and the full C names, for each of the `terminfo` variables.  

## RETURN VALUES  
All routines return the integer `ERR` upon failure and an integer value other than `ERR` upon successful completion, unless otherwise noted in the preceding routine descriptions.  
Routines that return pointers always return `NULL` on error.  

## SEE ALSO  
`curs_initscr(3X), curs_kernel(3X), curs_termcap(3X), curses(3X), putc(3S), terminfo(4)`

## NOTES  
The header `<curses.h>` automatically includes the headers `<stdio.h>` and `<unctrl.h>`.  
The `setupterm()` routine should be used in place of `setterm()`.  
Note that `vidattr()` and `vidputs()` may be macros.
### NAME
curs_touch, touchwin, touchline, untouchwin, wtouchln, is_linetouched, is_wintouched – curses refresh control routines

### SYNOPSIS
c
```c
#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);
```

### MT-LEVEL
Unsafe

### 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 descriptions.

### SEE ALSO
curs_refresh(3X), curses(3X)

### NOTES
The header `<curses.h>` automatically includes the headers `<stdio.h>` and `<unctrl.h>.

Note that all routines except `wtouchln()` may be macros.

modified 22 Jan 1993

3X-281
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 *filep);
WINDOW *getwin(FILE *filep);
int delay_output(int ms);
int flushinp(void);

MT-LEVEL
Unsafe

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 ter-
minfo 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 filep points. This information can be later retrieved using the getwin() function.
The getwin() routine reads window related data stored in the file by putwin(). The 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.

3X-282 modified 22 Jan 1993
RETURN VALUES

Except for `flushinp()`, routines that return an integer return `ERR` upon failure and an integer value other than `ERR` upon successful completion. `flushinp()` always returns `OK`.

Routines that return pointers return `NULL` on error.

SEE ALSO

curs_initscr(3X), curs_scr_dump(3X), curses(3X)

NOTES

The header `<curses.h>` automatically includes the headers `<stdio.h>` and `<unctrl.h>`. Note that `unctrl()` is a macro, which is defined in `<unctrl.h>`.
NAME
curs_window, newwin, delwin, mvwin, subwin, derwin, mvderwin, dupwin, wsyncup,
syncok, wcursyncup, wsyncdown – create curses windows

SYNOPSIS
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);

MT-LEVEL Unsafe

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 rela-
tive to the origin of the window orig rather than the screen. There is no difference
between the subwindows and the derived windows.
The `mvderwin()` routine moves a derived window (or subwindow) inside its parent window. The screen-relative parameters of the window are not changed. This routine is used to display different parts of the parent window at the same physical position on the screen.

The `dupwin()` routine creates an exact duplicate of the window `win`.

Each `curses` window maintains two data structures: the character image structure and the status structure. The character image structure is shared among all windows in the window hierarchy (that is, the window with all subwindows). The status structure, which contains information about individual line changes in the window, is private to each window. The routine `wrefresh()` uses the status data structure when performing screen updating. Since status structures are not shared, changes made to one window in the hierarchy may not be properly reflected on the screen.

The routine `wsyncup()` causes the changes in the status structure of a window to be reflected in the status structures of its ancestors. If `syncok()` is called with second argument `TRUE` then `wsyncup()` is called automatically whenever there is a change in the window.

The routine `wcursyncup()` updates the current cursor position of all the ancestors of the window to reflect the current cursor position of the window.

The routine `wsyncdown()` updates the status structure of the window to reflect the changes in the status structures of its ancestors. Applications seldom call this routine because it is called automatically by `wrefresh()`.

**RETURN VALUES**

Routines that return an integer return the integer `ERR` upon failure and an integer value other than `ERR` upon successful completion.

`delwin()` returns the integer `ERR` upon failure and `OK` upon successful completion.

Routines that return pointers return `NULL` on error.

**SEE ALSO**

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

**NOTES**

The header `<curses.h>` automatically includes the headers `<stdio.h>` and `<unctrl.h>.

If many small changes are made to the window, the `wsyncup()` option could degrade performance.

Note that `syncok()` may be a macro.
NAME curses – CRT screen handling and optimization package

SYNOPSIS cc [ flag  ... ] file  ...  -lcurses [ library  ... ]
#include <curses.h>

MT-LEVEL Unsafe

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 affect 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_addchnstr(3X)</td>
</tr>
<tr>
<td>addchstr</td>
<td>curs_addchstr(3X)</td>
</tr>
<tr>
<td>addnstr</td>
<td>curs_addnstr(3X)</td>
</tr>
<tr>
<td>addnwstr</td>
<td>curs_addnwstr(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_addwchnstr(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_adjcurspos(3X)</td>
</tr>
<tr>
<td>attroff</td>
<td>curs_attroff(3X)</td>
</tr>
<tr>
<td>attron</td>
<td>curs_attron(3X)</td>
</tr>
<tr>
<td>attrset</td>
<td>curs_attrset(3X)</td>
</tr>
<tr>
<td>baudrate</td>
<td>curs_baudrate(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_bkgdset(3X)</td>
</tr>
<tr>
<td>border</td>
<td>curs_border(3X)</td>
</tr>
<tr>
<td>box</td>
<td>curs_box(3X)</td>
</tr>
<tr>
<td>can_change_color</td>
<td>curs_can_change_color(3X)</td>
</tr>
<tr>
<td>cbreak</td>
<td>curs_cbreak(3X)</td>
</tr>
<tr>
<td>clear</td>
<td>curs_clear(3X)</td>
</tr>
<tr>
<td>clearok</td>
<td>curs_clearok(3X)</td>
</tr>
<tr>
<td>clrtobot</td>
<td>curs_clrtobot(3X)</td>
</tr>
<tr>
<td>color_content</td>
<td>curs_color_content(3X)</td>
</tr>
<tr>
<td>copywin</td>
<td>curs_copywin(3X)</td>
</tr>
<tr>
<td>curs_set</td>
<td>curs_set(3X)</td>
</tr>
<tr>
<td>def_prog_mode</td>
<td>curs_def_prog_mode(3X)</td>
</tr>
<tr>
<td>def_shell_mode</td>
<td>curs_def_shell_mode(3X)</td>
</tr>
<tr>
<td>del_curterm</td>
<td>curs_del_curterm(3X)</td>
</tr>
<tr>
<td>delay_output</td>
<td>curs_delay_output(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>delscroll</td>
<td>curs_delscroll(3X)</td>
</tr>
<tr>
<td>delwin</td>
<td>curs_delwin(3X)</td>
</tr>
<tr>
<td>derwin</td>
<td>curs_derwin(3X)</td>
</tr>
<tr>
<td>dopupdate</td>
<td>curs_dopupdate(3X)</td>
</tr>
<tr>
<td>dupwin</td>
<td>curs_dupwin(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_bbeep(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_getysy(3X)</td>
</tr>
<tr>
<td>getwch</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>insdelnl</td>
<td>curs_deletein(3X)</td>
</tr>
<tr>
<td>insertln</td>
<td>curs_deletein(3X)</td>
</tr>
<tr>
<td>insnstr</td>
<td>curs_inssnstr(3X)</td>
</tr>
<tr>
<td>insnwstr</td>
<td>curs_inswstr(3X)</td>
</tr>
<tr>
<td>insptr</td>
<td>curs_insstr(3X)</td>
</tr>
<tr>
<td>inst</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>
</tbody>
</table>

3X-290 modified 22 Jan 1993
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inwch</td>
<td>curs_inwch(3X)</td>
</tr>
<tr>
<td>inwchnstr</td>
<td>curs_inwchstr(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_termattrs(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>moveextendch</td>
<td>curs_aletciamp(3X)</td>
</tr>
<tr>
<td>moveprevch</td>
<td>curs_aletciamp(3X)</td>
</tr>
<tr>
<td>mvaddch</td>
<td>curs_addch(3X)</td>
</tr>
<tr>
<td>mvaddchnstr</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_addwstr(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>mvaddwchnstr</td>
<td>curs_addwchstr(3X)</td>
</tr>
<tr>
<td>mvaddwchstr</td>
<td>curs_addwchstr(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>mvgetnwsstr</td>
<td>curs_getwstr(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>mvgetwsr</td>
<td>curs_getwstr(3X)</td>
</tr>
<tr>
<td>mvinch</td>
<td>curs_inch(3X)</td>
</tr>
<tr>
<td>mvinchnstr</td>
<td>curs_inchstr(3X)</td>
</tr>
<tr>
<td>mvinchnstr</td>
<td>curs_inchstr(3X)</td>
</tr>
<tr>
<td>mvinnstr</td>
<td>curs_instr(3X)</td>
</tr>
<tr>
<td>mvinnwstr</td>
<td>curs_inwstr(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>mvinsnswstr</td>
<td>curs_insstr(3X)</td>
</tr>
<tr>
<td>mvinsstr</td>
<td>curs_insstr(3X)</td>
</tr>
<tr>
<td>mvinswch</td>
<td>curs_insstr(3X)</td>
</tr>
</tbody>
</table>

modified 22 Jan 1993
<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>mvinwstr</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>mvwaddnstr</td>
<td>curs_addstr(3X)</td>
</tr>
<tr>
<td>mvwaddnstrw</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>mvwinnstr</td>
<td>curs_instr(3X)</td>
</tr>
<tr>
<td>mvwinnstrw</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>mvwinsnstr</td>
<td>curs_insstr(3X)</td>
</tr>
<tr>
<td>mvwinstr</td>
<td>curs_instr(3X)</td>
</tr>
<tr>
<td>mvwinswch</td>
<td>curs_inschwch(3X)</td>
</tr>
<tr>
<td>mvwinswstr</td>
<td>curs_inschwstr(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>mvwprintw</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_initialscr(3X)</td>
</tr>
<tr>
<td>newwin</td>
<td>curs_window(3X)</td>
</tr>
</tbody>
</table>

SunOS 5.5

modified 22 Jan 1993
nl  
c nocbreak  
nodelay  
n echo  
nonl  
nq iflush  
noraw  
n timeout  
overlay  
overwrite  
pair_content  
pechochar  
pechowchar  
pnoutrefresh  
prefresh  
printw  
putp  
putwin  
qiflush  
raw  
redrawwin  
refresh  
reset_prog_mode  
reset_shell_mode  
resetty  
restartterm  
ripof offline  
savetty  
scanw  
scr_dump  
scr_init  
scr_restore  
scr_set  
scroll  
scrollok  
set_curterm  
set_term  
setscreg  
setsyx  
setterm  
setupterm  
slk_attroff  
slk_attrs  
slk_attrset  
slk_clear
curses (3X)  Miscellaneous Library Functions  SunOS 5.5

slk_init  curs_slk(3X)
slk_label  curs_slk(3X)
slk_noutrefresh  curs_slk(3X)
slk_refresh  curs_slk(3X)
slk_restore  curs_slk(3X)
slk_set  curs_slk(3X)
slk_touch  curs_slk(3X)
srl  curs_scroll(3X)
standend  curs_attr(3X)
standout  curs_attr(3X)
start_color  curs_color(3X)
subpad  curs_pad(3X)
subwin  curs_window(3X)
syncok  curs_window(3X)
termattrs  curs_termattrs(3X)
termmname  curs_termattrs(3X)
tgetent  curs_termcap(3X)
tgetflag  curs_termcap(3X)
tgetnum  curs_termcap(3X)
tgetstr  curs_termcap(3X)
tgoto  curs_termcap(3X)
tigetflag  curs_terminfo(3X)
tigetnum  curs_terminfo(3X)
tigetstr  curs_terminfo(3X)
tigetstr  curs_terminfo(3X)
timeout  curs_inopts(3X)
touchline  curs_touch(3X)
touchwin  curs_touch(3X)
tparm  curs_terminfo(3X)
tputs  curs_terminfo(3X)
typeahead  curs_inopts(3X)
tncrl  curs_util(3X)
ungetch  curs_getch(3X)
ungetwch  curs_getwch(3X)
untouchwin  curs_touch(3X)
use_env  curs_util(3X)
vidattr  curs_terminfo(3X)
vidputs  curs_terminfo(3X)
vwprintw  curs_printw(3X)
vwscanw  curs_scanw(3X)
waddch  curs_addch(3X)
waddchnstr  curs_addchnstr(3X)
waddchstr  curs_addchstr(3X)
waddnstr  curs_addstr(3X)
waddnwstr  curs_addwstr(3X)
waddstr  curs_addstr(3X)

3X-294  modified 22 Jan 1993
waddwch    curs_addwch(3X)
waddwchnstr    curs_addwchstr(3X)
waddwchstr    curs_addwchstr(3X)
waddwstr    curs_addwstr(3X)
wadjcurspos    curs_alecompat(3X)
wattroff    curs_attr(3X)
wattron    curs_attr(3X)
wattrset    curs_attr(3X)
wbkgd    curs_bkgd(3X)
wbkgdset    curs_bkgd(3X)
wborder    curs_border(3X)
wclear    curs_clear(3X)
wclrtobot    curs_clear(3X)
wclrtoeol    curs_clear(3X)
wcursyncup    curs_window(3X)
wdelch    curs_delch(3X)
wdeleteeln    curs_deleteeln(3X)
wechochar    curs_addch(3X)
wechowchar    curs_addwch(3X)
werase    curs_clear(3X)
wgetch    curs_getch(3X)
wgetnstr    curs_getstr(3X)
wgetnwstr    curs_getwstr(3X)
wgetstr    curs_getstr(3X)
wgetwch    curs_getwch(3X)
wgetwstr    curs_getwstr(3X)
whline    curs_border(3X)
winch    curs_inch(3X)
winchnstr    curs_inchstr(3X)
winchstr    curs_inchstr(3X)
winnstr    curs_instr(3X)
winnwstr    curs_inwstr(3X)
winsch    curs_insch(3X)
windselln    curs_deleteeln(3X)
winsertln    curs_deleteeln(3X)
winsnstr    curs_instr(3X)
winsnwstr    curs_inswstr(3X)
winsnstr    curs_instr(3X)
winswstr    curs_inswstr(3X)
winwch    curs_inwch(3X)
winwchnstr    curs_inwchstr(3X)
winwchstr    curs_inwchstr(3X)
winwstr    curs_inwstr(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.

SEE ALSO

terminfo(4) and 3X pages whose names begin with “curs_” for detailed routine descriptions.

NOTES

The header <curses.h> automatically includes the headers <stdio.h> and <unctrl.h>.
NAME

cuserid – get character login name of the user

SYNOPSIS

#include <stdio.h>
char *cuserid(char *s);

MT-LEVEL

MT-Safe

DESCRIPTION

cuserid() generates a character-string representation of the login name that the owner of
the current process is logged in under. If 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].

SEE ALSO

getlogin(3C), getpwnam(3C)

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 ... -ldbm
#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(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), ndbm(3)

NOTES
Use of these interfaces should be restricted to only applications written on BSD platforms. Use of these interfaces with any of the system libraries or in multi-thread applications is unsupported.

The .pag file will contain holes so that its apparent size may be larger than its actual content. Older versions of the UNIX operating system may create real file blocks for these holes when touched. These files cannot be copied by normal means (cp(1), cat(1), tar(1), ar(1)) without filling in the holes.

dptr pointers returned by these subroutines point into static storage that is changed by subsequent calls.

The sum of the sizes of a key/content pair must not exceed the internal block size (currently 1024 bytes). Moreover all key/content pairs that hash together must fit on a single block. store will return an error in the event that a disk block fills with inseparable data.

delete() does not physically reclaim file space, although it does make it available for reuse.

The order of keys presented by firstkey() and nextkey() depends on a hashing function, not on anything interesting.

There are no interlocks and no reliable cache flushing; thus concurrent updating and reading is risky.

The database files (file.dir and file.pag) are binary and are architecture-specific (for example, they depend on the architecture's byte order.) These files are not guaranteed to be portable across architectures.

modified 31 Jan 1994
NAME  decimal_to_floating, decimal_to_single, decimal_to_double, decimal_to_extended, decimal_to_quadruple – convert decimal record to floating-point value

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

void decimal_to_single(single *px, decimal_mode *pm, decimal_record *pd,
                       fp_exception_field_type *ps);

void decimal_to_double(double *px, decimal_mode *pm, decimal_record *pd,
                        fp_exception_field_type *ps);

void decimal_to_extended(extended *px, decimal_mode *pm, decimal_record *pd,
                         fp_exception_field_type *ps);

void decimal_to_quadruple(quadruple *px, decimal_mode *pm, decimal_record *pd,
                           fp_exception_field_type *ps);
```

MT-LEVEL  MT-Safe

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.

`pd->sign` and `pd->fpclass` are always taken into account. `pd->exponent`, `pd->ds` and `pd->ndigits` are used when `pd->fpclass` is `fp_normal` or `fp_subnormal`. In these cases `pd->ds` must contain one or more ascii digits followed by a NULL and `pd->ndigits` is assumed to be the length of the string `pd->ds`. Notice that for efficiency reasons, the assumption that `pd->ndigits == strlen(pd->ds)` is NEVER verified.

On output, *px is set to a correctly rounded approximation to  
```
(pd->sign)*pd->ds*10**(pd->exponent)
```
Thus if `pd->exponent == -2` and `pd->ds == "1234"`, *px will get 12.34 rounded to storage precision. `pd->ds` cannot have more than `DECIMAL_STRING_LENGTH-1` significant digits because one character is used to terminate the string with a NULL. If `pd->more != 0` on input then additional nonzero digits follow those in `pd->ds`; `fp_inexact` is set accordingly on output in *ps.

*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.
`pm->df` and `pm->ndigits` are not used.

`strtod(3C)`, `scanf(3S)`, `fscanf(3S)`, and `sscanf(3S)` all use `decimal_to_double()`.

SEE ALSO  `fscanf(3S)`, `scanf(3S)`, `sscanf(3S)`, `strtod(3C)`
NAME  
demangle – decode a C++ encoded symbol name

SYNOPSIS  
CC[flag...file ... [library...]
#include<demangle.h>
cpp_demangle(const char *in, char *out, size_t size);

DESCRIPTION  
cplus_demangle() decodes the string in, and copies the result to out. in points to a
string representing a name mangled by the C++ compiler. out is a buffer of size that you
specify, which contains the byte size. If the output buffer is too small to contain the
demangled name, cplus_demangle() returns DEMANGLE_ESPACE, and the contents
of out are undefined. Otherwise, if the name is a valid C++ name, cplus_demangle()
returns 0. If in is not a valid C++ mangled name, it is copied unchanged to out and the
function returns DEMANGLE_ENAME.

cplus_demangle() operates on mangled names generated by C++ 3.0.1 and all versions
of C++ 4.0 and above.

SEE ALSO  
cc(1B) in the C++ Library Reference Manual.
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);

MT-LEVEL    Unsafe

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>`.

- **INTRPT** = -1 /* interrupt occurred */
- **D_HUNG** = -2 /* dialer hung (no return from write) */
- **NO_ANS** = -3 /* no answer within 10 seconds */
- **ILL_BD** = -4 /* illegal baud-rate */
- **A_PROB** = -5 /* acu problem (open() failure) */
- **L_PROB** = -6 /* line problem (open() failure) */
- **NO_Ldv** = -7 /* can’t open Devices file */
- **DV_NT_A** = -8 /* requested device not available */
- **DV_NT_K** = -9 /* requested device not known */
- **NO_BD_A** = -10 /* no device available at requested baud */
- **NO_BD_K** = -11 /* no device known at requested baud */
- **DV_NT_E** = -12 /* requested speed does not match */
- **BAD_SYS** = -13 /* system not in Systems file */

**FILES**

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

**SEE ALSO**

uucp(1C), alarm(2), read(2), write(2), 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**(1)s should be checked for (errno==EINTR), and the **read**(1) possibly reissued.

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

modified 22 Jan 1993

3N-303
NAME
difftime – computes the difference between two calendar times

SYNOPSIS
#include <time.h>
double difftime(time_t time1, time_t time0);

MT-LEVEL
MT-Safe

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.

SEE ALSO
ctime(3C)
NAME directory, opendir, readdir, readdir_r, telldir, seekdir, rewinddir, closedir – directory operations

SYNOPSIS

#include <dirent.h>

DIR *opendir(const char *filename);

struct dirent *readdir(DIR *dirp);

long telldir(DIR *dirp);

void seekdir(DIR *dirp, long loc);

void rewinddir(DIR *dirp);

int closedir(DIR *dirp);

Solaris 2.4 struct dirent *readdir_r(DIR *dirp, struct dirent *res);

POSIX cc [ flag ... ] file ... -D_POSIX_PTHREAD_SEMANTICS [ library ... ]

int *readdir_r(DIR *dirp, struct dirent *entry struct dirent **result);

MT-LEVEL See the NOTES section of this page.

DESCRIPTION opendir() opens the directory named by filename and associates a directory stream with it. opendir() returns a pointer to be used to identify the directory stream in subsequent operations. The directory stream is positioned at the first entry. A null pointer is returned if filename cannot be accessed or is not a directory, or if it cannot malloc(3C) enough memory to hold a DIR structure or a buffer for the directory entries.

readdir() 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 filesystems. It returns a NULL pointer upon reaching the end of the directory stream, or upon detecting an invalid location in the directory. readdir() 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. readdir() 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.

readdir_r() has the equivalent functionality as readdir() except that a buffer res must be supplied by the caller to store the result. To allocate res correctly a struct dirent res is not sufficient, thus the size should be sizeof(struct dirent) + _POSIX_PATH_MAX (defined in <limits.h>).

The POSIX readdir_r() function initializes the structure referenced by entry and stores a pointer to this structure in result.
telldir() returns the current location associated with the named directory stream.
seekdir() sets the position of the next readdir() operation on the directory stream. The new position reverts to the position associated with the directory stream at the time the telldir() operation that provides loc was performed. Values returned by telldir() are good only for the lifetime of the DIR pointer from which they are derived. If the directory is closed and then reopened, the telldir() value may be invalidated due to undetected directory compaction. It is safe to use a previous telldir() value immediately after a call to opendir() and before any calls to readdir.
rewinddir() resets the position of the named directory stream to the beginning of the directory. It also causes the directory stream to refer to the current state of the corresponding directory, as a call to opendir() would.
closedir() closes the named directory stream and frees the DIR structure.

RETURN VALUES opendir(), readdir(), and readdir_r() return NULL on failure and set errno to indicate the error. The POSIX readdir_r() returns zero if successful, or an error number to indicate failure. telldir(), seekdir(), and closedir() return -1 on failure and set errno to indicate the error.

ERRORS opendir() will fail if one or more of the following are true:
EACCES Read permission is denied on the specified directory.
EFAULT filename points outside the allocated address space.
ELOOP Too many symbolic links were encountered in translating filename.
ENOTDIR A component of filename is not a directory.
EMFILE The maximum number of file descriptors are currently open.
ENFILE The system file table is full.
ENAMETOOLONG The length of the filename argument exceeds PATH_MAX, or the length of a filename component exceeds NAME_MAX while _POSIX_NO_TRUNC is in effect.
ENOENT A component of filename does not exist or is a null pathname.
EACCES A component of filename denies search permission.
readdir() and readdir_r() will fail if one or more of the following are true:
EAGAIN Mandatory file/record locking was set, O_NDELAY or O_NONBLOCK was set, and there was a blocking record lock.
EAGAIN Total amount of system memory available when reading using raw I/O is temporarily insufficient.
EAGAIN No data is waiting to be read on a file associated with a tty device and O_NONBLOCK was set.
EAGAIN No message is waiting to be read on a stream and O_NDELAY or O_NONBLOCK was set.
EBADF The file descriptor determined by the DIR stream is no longer
valid. This results if the DIR stream has been closed.

EBADMSG Message waiting to be read on a stream is not a data message.

EDEADLK The read() was going to go to sleep and cause a deadlock to occur.

EFAULT buf points to an illegal address.

EINTR A signal was caught during the read() or readv() function.

EINVAL Attempted to read from a file linked to a multiplexor.

EIO A physical I/O error has occurred, or the process is in a background process group and is attempting to read from its controlling terminal, and either the process is ignoring or blocking the SIGTTIN signal or the process group of the process is orphaned.

ENOENT The current file pointer for the directory is not located at a valid entry.

ENOLCK The system record lock table was full, so the read() or readv() could not go to sleep until the blocking record lock was removed.

ENOLINK fildes is a block special or character special file and the value of the file pointer is out of range.

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.

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

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

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

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

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

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

SEE ALSO getdents(2), dirent(4)
NOTES

When compiling multi-thread applications, the _REENTRANT flag must be defined on the compile line. This flag should only be used in multi-thread applications.

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

The POSIX `readdir_r()` interface is as specified in POSIX 1003.1c Draft #10.
NAME
dirname — report the parent directory name of a file path name

SYNOPSIS
cc [ flag ... ] file ... -Igen [ library ... ]
#include <libgen.h>
char *dirname(char *path);

MT-LEVEL
MT-Safe

DESCRIPTION
Given a pointer to a null-terminated character string that contains a file system path
name, dirname() returns a string that is the parent directory of that file. In doing this, it
may place a null byte in the path name after the next to last element, so the content of path
must be disposable. The returned string should not be deallocated by the caller. Trailing
"/" characters in the path are not counted as part of the path.

If path or *path is zero, a pointer to a static constant "." is returned.
dirname() and basename() together yield a complete path name. dirname(path) is the
directory where basename(path) is found.

EXAMPLES
A simple file name and the strings "." and ".." all have "." as their return value.

<table>
<thead>
<tr>
<th>Input string</th>
<th>Output pointer</th>
</tr>
</thead>
<tbody>
<tr>
<td>/usr/lib</td>
<td>/usr</td>
</tr>
<tr>
<td>/usr/</td>
<td>/</td>
</tr>
<tr>
<td>usr</td>
<td>.</td>
</tr>
<tr>
<td>/</td>
<td>/</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>..</td>
<td></td>
</tr>
</tbody>
</table>

The following code reads a path name, changes directory to the parent directory of the
named file (see chdir(2)), and opens the file.

cchar path[100], *pathcopy;
int fd;
gets (path);
pathcopy = strdup (path);
chdir (dirname (pathcopy) );
free (pathcopy);
fd = open (basename (path), O_RDONLY);

SEE ALSO
basename(1), chdir(2), basename(3G)

NOTES
When compiling multi-thread applications, the _REENTRANT flag must be defined on
the compile line. This flag should only be used in multi-thread applications.
NAME  div, ldiv, lldiv – compute the quotient and remainder

SYNOPSIS  #include <stdlib.h>
            div_t div(int numerator, int denominator);
            ldiv_t ldiv(long int numerator, long int denominator);
            lldiv_t lldiv(long long numerator, long long denominator);

MT-LEVEL  MT-Safe

DESCRIPTION  div() computes the quotient and remainder of the division of the numerator `numerator` by the denominator `denominator`. 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 * denominator + remainder` will equal `numerator`.

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. ldiv() 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:
                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:
                long int quot;  /*quotient*/
                long int rem;  /*remainder*/

modified 20 Mar 1994
**NAME**
dladdr – translate address to symbolic information

**SYNOPSIS**
```
cc [ flag ... ] file ... -ldl [ library ... ]
#include <dlfcn.h>
int dladdr(void * address, Dl_info * dip);
```

**MT-LEVEL**
MT-Safe

**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.

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 nearest symbol name 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.

**SEE ALSO**
ld(1), dlclose(3X), dlerror(3X), dlopen(3X), dlsym(3X)
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()).

modified 20 Aug 1993 3X-311
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);

MT-LEVEL
MT-Safe

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.

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-0 value.
More detailed diagnostic information will be available through dlerror().

SEE ALSO
ld(1), dladdr(3X), dlerror(3X), dlopen(3X), dlsym(3X)
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  dlerror – get diagnostic information

SYNOPSIS  cc [ flag ... ] file ... -ldl [ library ... ]
           #include <dlfcn.h>
           char *dlerror(void);

MT-LEVEL  MT-Safe

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.

              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.

SEE ALSO  ld(1), dladdr(3X), dlclose(3X), dlopen(3X), dlsym(3X)

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.

3X-314  modified 22 Jan 1993
NAME
dlopen – open a shared object

SYNOPSIS
cc [ flag …] file … -ldl [ library …]
#include <dlfcn.h>
void * dlopen(const char * pathname, int mode);

MT-LEVEL
MT-Safe

DESCRIPTION
dlopen() 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. These routines are available to dynamically linked processes ONLY.

dlopen() makes a shared object available to a running process. dlopen() returns to the process a "handle" which the process may use on subsequent calls to dlsym() and dlclose(). 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 run-time linker will be used to locate the specified file (see NOTES section below).

If the value of pathname is 0, dlopen() makes the symbols contained in the original a.out, any objects loaded at program startup with the a.out, and any objects that were loaded using dlopen() together with the RTLD_GLOBAL flag, available through dlsym().

When a shared object is brought into the address space of a process, it may contain references to symbols whose addresses are not known until the object is loaded. These references must be relocated before the symbols can be accessed. The 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 shared object. This behavior mimics the normal loading of shared object dependencies by a dynamic executable 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.

Any object loaded by dlopen() that requires relocations against global symbols can reference the symbols in the a.out, any objects loaded at program startup, from the object itself, and from any dependencies the object references. By default, the relocations of an
object loaded by one \texttt{dlopen()} invocation may not reference symbols from objects loaded by a different \texttt{dlopen()} invocation. However, the \textit{mode} parameter may also be \texttt{ored} with the following value to effect the scope of symbol availability:

\begin{itemize}
  \item \texttt{RTLD\_GLOBAL} \quad The objects symbols are made available for the relocation processing of any other object. In addition, symbol lookup using \texttt{dlopen(0,mode)} and an associated \texttt{dlsym()}, allows objects loaded with this \textit{mode} to be searched.
\end{itemize}

\section*{RETURN VALUES}

If \texttt{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 \texttt{pathname} or relocating its symbolic references, \texttt{dlopen()} will return \texttt{NULL}. More detailed diagnostic information will be available through \texttt{dlerror(\)}.

\section*{SEE ALSO}

\texttt{ld(1), dladdr(3X), dlclose(3X), dlerror(3X), dlsym(3X)}

\textit{Linker and Libraries Guide}

\section*{NOTES}

If other shared objects were link edited with \texttt{pathname} when \texttt{pathname} was built (i.e., the \texttt{pathname} has dependencies on other shared objects), those objects will automatically be loaded by \texttt{dlopen(\)}. The directory search path used to find both \texttt{pathname} and the other \textit{needed} objects may be affected by setting the environment variable \texttt{LD\_LIBRARY\_PATH} (which is analyzed once at process startup), or from a run-path setting within the application or the shared object from which the \texttt{dlopen(\)} originated. These search rules will only be applied to pathnames that do not contain an embedded \texttt{"/"}. Objects whose names resolve to the same absolute or relative path name may be opened any number of times using \texttt{dlopen(\)}, however, the object referenced will only be loaded once into the address space of the current process.

Some symbols defined in dynamic executables or shared objects may not be available to the runtime linker. The symbol table created by \texttt{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
cc [ flag ...] file ... -ldl [ library ...]
#include <dlfcn.h>
void *dlsym(void *handle, const char *name);

MT-LEVEL
MT-Safe

DESCRIPTION
dlsym() is one of a family of routines that give the user direct access to the dynamic link-
ing 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.

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 the special flag
RTLD_NEXT. name is the symbol’s name as a character string.

In the former case the corresponding shared object must not have been closed using
dlclose(). dlsym() will search for the named symbol in all shared objects loaded
automatically as a result of loading the object referenced by handle (see dlopen()).

In the latter case dlsym() will search for the named symbol in the objects that were
loaded following the object from which the dlsym() call is being made. If these subse-
quent objects were loaded from dlopen() calls, dlsym() will search the object only if the
caller is part of the same dlopen() dependency hierarchy, or the object was given global
search access (see dlopen() with reference to the mode RTLD_GLOBAL).

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 associ-
ated with handle, dlsym() will return NULL. More detailed diagnostic information will
be 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.

    void     *handle;
    int      *iptr, (*fptr)(int);

    /* open the needed object */
    handle = dlopen("/usr/home/me/libfoo.so.1", RTLD_LAZY);

    /* find the address of function and data objects */
    fptr = (int (*)(int))dlsym(handle, "my_function");
    iptr = (int *)dlsym(handle, "my_object");

    /* invoke function, passing value of integer as a parameter */
    (*fptr)(*iptr);

modified 2 Apr 1993 3X-317
SEE ALSO

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

Linker and Libraries Guide
NAME       doconfig – execute a configuration script

SYNOPSIS   cc [ flag ... ] file ... –lnsl [ library ... ]
            # include <sac.h>
            int doconfig(int fildes, char *script, long rflag);

MT-LEVEL    Unsafe

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 it was and the command fails. If module is the special keyword ALL, then all modules on the stream will be popped. Note that only modules above the topmost driver are affected.

runwait command
The runwait command runs a command and waits for it to complete. command is the pathname of the command to be run. The command is run with /usr/bin/sh –c prepended to it; shell scripts may thus be executed from configuration scripts. The runwait command will fail if command cannot be found or cannot be executed, or if command exits with a non-zero status.

run command
The run command is identical to runwait except that it does not wait for command to complete. command is the pathname of the command to be run. run will not fail unless it is unable to create a child process to execute the command.

Although they are syntactically indistinguishable, some of the commands available to run and runwait are interpreter built-in commands. Interpreter built-ins are used when it is necessary to alter the state of a process within the context of that process. The doconfig() interpreter built-in commands are similar to the shell special commands and, like these, they do not spawn another process for execution. See sh(1). The built-in commands are:

```
cd
ulimit
umask
```

RETURN VALUES
doconfig() returns 0 if the script was interpreted successfully. If a command in the script fails, the interpretation of the script ceases at that point and a positive number is returned; this number indicates which line in the script failed. If a system error occurs, a value of −1 is returned. When a script fails, the process whose environment was being established should not be started.

SEE ALSO
sh(1), pmadm(1M), sacadm(1M)

NOTES
This interface is unsafe in multithreaded applications. Unsafe interfaces should be called only from the main thread.
NAME
drand48, erand48, lrand48, nrand48, mrand48, jrand48, srand48, seed48, lcong48 – generate uniformly distributed pseudo-random numbers

SYNOPSIS
#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]);

MT-LEVEL
Safe

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

\[ X_{n+1} = (aX_n + c) \mod m \quad n \geq 0. \]

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 \texttt{drand48()}, \texttt{erand48()}, \texttt{lrand48()}, \texttt{nrand48()}, \texttt{mrand48()}, or \texttt{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 \texttt{drand48()}, \texttt{lrand48()}, and \texttt{mrand48()} store the last 48-bit \(X_i\) generated in an internal buffer. \(X_i\) must be initialized prior to being invoked. The functions \texttt{erand48()}, \texttt{nrand48()}, and \texttt{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 \texttt{erand48()}, \texttt{nrand48()}, and \texttt{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 \texttt{srand48()} sets the high-order 32 bits of \(X_i\) to the 32 bits contained in its argument. The low-order 16 bits of \(X_i\) are set to the arbitrary value \(330E_{16}\).

The initializer function \texttt{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 \texttt{seed48()}, and a pointer to this buffer is the value returned by \texttt{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 \texttt{seed48()} when the program is restarted.

The initialization function \texttt{lcong48()} allows the user to specify the initial \(X_i\), the multiplier value \(a\), and the addend value \(c\). Argument array elements \texttt{param[0-2]} specify \(X_i\), \texttt{param[3-5]} specify the multiplier \(a\), and \texttt{param[6]} specifies the 16-bit addend \(c\). After \texttt{lcong48()} has been called, a subsequent call to either \texttt{srand48()} or \texttt{seed48()} will restore the “standard” multiplier and addend values, \(a\) and \(c\), specified above.

\textbf{SEE ALSO} \texttt{rand}(3C)
NAME
dup2 – duplicate an open file descriptor

SYNOPSIS
#include <unistd.h>
int dup2(int fildes, int fildes2);

MT-LEVEL
Unsafe
Async-Signal-Safe

DESCRIPTION
dup2( ) causes the file descriptor fildes2 to refer to the same file as fildes. fildes is a file
descriptor referring to an open file, and fildes2 is a non-negative integer less than the
current value for the maximum number of open file descriptors allowed the calling pro-
cess (see getrlimit(2)). If fildes2 already referred to an open file, not fildes, it is closed first. If fildes2 refers to fildes, or if fildes is not a valid open file descriptor, fildes2 will not be
closed first.

RETURN VALUES
Upon successful completion a non-negative integer, namely, the file descriptor, is
returned. Otherwise, a value of −1 is returned and errno is set to indicate the error.

ERRORS
dup2( ) will fail if one or more of the following are true:
EBADF fildes is not a valid open file descriptor.
EINTR a signal was caught during the dup2( ) call.
EMFILE the process has too many open files (see fcntl(2)).

SEE ALSO close(2), creat(2), exec(2), fcntl(2), getrlimit(2), open(2), pipe(2), lockf(3C)

modified 23 Mar 1994

3C-323
NAME
econvert, fconvert, gconvert, seconvert, sfconvert, sgconvert, qeconvert, qfconvert,
qeconvert, fcvt, gcvt, 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 trailing, char *buf);
char *qeconvert(quadruple *value, int ndigit, int *decpt, int *sign, char *buf);
char *qfconvert(quadruple *value, int ndigit, int *decpt, int *sign, char *buf);
char *qgconvert(quadruple *value, int ndigit, int trailing, char *buf);
char *ecvt(double value, int ndigit, int *decpt, int *sign);
char *fcvt(double value, int ndigit, int *decpt, int *sign);
char *gcvt(double value, int ndigit, char *buf);

MT-LEVEL
MT-Safe

DESCRIPTION
econvert() 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.

fconvert 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 accomodate any
double-precision value.

gconvert() 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 trail-
ing!= 0, trailing zeros and a trailing point are retained, as in sprintf(%#g).

3-324
modified 13 Jul 1992
seconvert, sfconvert, and sgconvert() are single-precision versions of these functions, and are more efficient than the corresponding double-precision versions. A pointer rather than the value itself is passed to avoid C’s usual conversion of single-precision arguments to double.

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

ecvt() and fcvt() are obsolete versions of econvert() and fconvert() that create a string in a static data area, overwritten by each call, and return values that point to that static data. These functions are therefore not reentrant.

gcvt() is an obsolete version of gconvert() that always suppresses trailing zeros and point.

IEEE Infinities and NaNs are treated similarly by these functions. “NaN” is returned for NaN, and “Inf” or “Infinity” for Infinity. The longer form is produced when ndigit >= 8.

SEE ALSO sprintf(3S)
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);

MT-LEVEL  Unsafe

DESCRIPTION  ecvt() converts value to a null-terminated string of ndigit digits and returns a pointer thereto. The high-order digit is non-zero, unless the value is zero. The low-order digit is rounded. The position of the decimal point relative to the beginning of the string is stored indirectly through decpt (negative means to the left of the returned digits). The decimal point is not included in the returned string. If the sign of the result is negative, the word pointed to by sign is non-zero, otherwise it is zero.

fcvt() is identical to ecvt(), except that the correct digit has been rounded for printf %f output of the number of digits specified by ndigit.

gcvt() converts the value to a null-terminated string in the array pointed to by buf and returns buf. It attempts to produce ndigit significant digits in %f format if possible, otherwise %e format (scientific notation), ready for printing. A minus sign, if there is one, or a decimal point will be included as part of the returned string. Trailing zeros are suppressed.

SEE ALSO  printf(3S)

NOTES  The values returned by ecvt() and fcvt() point to a single static data array whose content is overwritten by each call.
NAME  elf – object file access library

SYNOPSIS  cc [ flag ... ] file ... -l elf [ library ... ]
#include <libelf.h>

MT-LEVEL  Unsafe

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, etc., 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 described below.

**DATA REPRESENTATION**

Conceptually, two parallel sets of objects support cross compilation environments. One set corresponds to file contents, while the other set corresponds to the native memory image of the program manipulating the file. Type definitions supplied by the headers work on the native machine, which may have different data encodings (size, byte order, etc.) than the target machine. Although native memory objects should be at least as big as the file objects (to avoid information loss), they may be bigger if that is more natural for the host machine.

Translation facilities exist to convert between file and memory representations. Some library routines convert data automatically, while others leave conversion as the program’s responsibility. Either way, programs that create object files must write file-typed objects to those files; programs that read object files must take a similar view. See `elf32_xlatetof` and `elf32_fsize` for more information.

Programs may translate data explicitly, taking full control over the object file layout and semantics. If the program prefers not to have and exercise complete control, the library provides a higher-level interface that hides many object file details. `elf_begin()` and related functions let a program deal with the native memory types, converting between memory objects and their file equivalents automatically when reading or writing an object file.

**ELF VERSIONS**

Object file versions allow ELF to adapt to new requirements. *Three independent versions* can be important to a program. First, an application program knows about a particular version by virtue of being compiled with certain headers. Second, the access library similarly is compiled with header files that control what versions it understands. Third, an ELF object file holds a value identifying its version, determined by the ELF version known by the file’s creator. Ideally, all three versions would be the same, but they may differ.

- If a program’s version is newer than the access library, the program might use information unknown to the library. Translation routines might not work properly, leading to undefined behavior. This condition merits installing a new library.
- The library’s version might be newer than the program’s and the file’s. The library understands old versions, thus avoiding compatibility problems in this case.
- Finally, a file’s version might be newer than either the program or the library understands. The program might or might not be able to process the file properly, depending on whether the file has extra information and whether that information can be safely ignored. Again, the safe alternative is to install a new library that understands the file’s version.
To accommodate these differences, a program must use `elf_version(3E)` to pass its version to the library, thus establishing the working version for the process. Using this, the library accepts data from and presents data to the program in the proper representations. When the library reads object files, it uses each file’s version to interpret the data. When writing files or converting memory types to the file equivalents, the library uses the program’s working version for the file data.

**SYSTEM SERVICES**

As mentioned above, `elf_begin()` and related routines provide a higher-level interface to ELF files, performing input and output on behalf of the application program. These routines assume a program can hold entire files in memory, without explicitly using temporary files. When reading a file, the library routines bring the data into memory and perform subsequent operations on the memory copy. Programs that wish to read or write large object files with this model must execute on a machine with a large process virtual address space. If the underlying operating system limits the number of open files, a program can use `elf_cntl(3E)` to retrieve all necessary data from the file, allowing the program to close the file descriptor and reuse it.

Although the `elf_begin()` interfaces are convenient and efficient for many programs, they might be inappropriate for some. In those cases, an application may invoke the `elf32_xlatetom(3E)` or `elf32_xlatetof(3E)` data translation routines directly. These routines perform no input or output, leaving that as the application’s responsibility. By assuming a larger share of the job, an application controls its input and output model.

**LIBRARY NAMES**

Names associated with the library take several forms.

- **elf_name**: These class-independent names perform some service, `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`.

modified 12 Jul 1995
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`.

**EXAMPLES**

The basic interpretation of an ELF file consists of:

- opening an ELF object file
- obtaining an ELF descriptor
- analyzing the file using the descriptor.

The following example opens the file, obtains the ELF descriptor, and prints out the names of each section in the file.

```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();
```

*modified 12 Jul 1995*
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\0, cnt, 
        (char * )data->d_buf + shdr->sh_name);
}
} /* end main */

static void
failure()
{
    (void) fprintf(stderr, "%s\0, 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 
...

SEE ALSO elf32_fsize(3E), elf32_getshdr(3E), elf32_xlatetof(3E), elf_begin(3E), elf_cntl(3E),
elf_errmsg(3E), elf_fill(3E), elf_getarhdr(3E), elf_getarsym(3E), elf_getbase(3E),
elf_getdata(3E), elf_getident(3E), elf_getscn(3E), elf_hash(3E), elf_kind(3E),
elf_memory(3E), elf_rawfile(3E), elf_strptr(3E), elf_update(3E), elf_version(3E), ar(4)

ANSI C Programmer's Guide

SPARC only a.out(4)

modified 12 Jul 1995
Information in the ELF headers is separated into common parts and processor-specific parts. A program can make a processor’s information available by including the appropriate header: `<sys/elf_NAME.h>` where `NAME` matches the processor name as used in the ELF file header.

<table>
<thead>
<tr>
<th>Name</th>
<th>Processor</th>
</tr>
</thead>
<tbody>
<tr>
<td>M32</td>
<td>AT&amp;T WE 32100</td>
</tr>
<tr>
<td>SPARC</td>
<td>SPARC</td>
</tr>
<tr>
<td>386</td>
<td>Intel 80386</td>
</tr>
<tr>
<td>486</td>
<td>Intel 80486</td>
</tr>
<tr>
<td>860</td>
<td>Intel 80860</td>
</tr>
<tr>
<td>68K</td>
<td>Motorola 68000</td>
</tr>
<tr>
<td>88K</td>
<td>Motorola 88000</td>
</tr>
</tbody>
</table>

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:

```c
#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:

```c
#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);

MT-LEVEL  Unsafe

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>1</td>
<td>sizeof(unsigned char)</td>
</tr>
<tr>
<td>ELF_T_HALF</td>
<td>ELF32_FSZ_HALF</td>
<td>sizeof(Elf32_Half)</td>
</tr>
<tr>
<td>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 zero if the value of type or ver is unknown. See elf32_xlatetof(3E) for a list of the type values.

SEE ALSO  elf(3E), elf32_xlatetof(3E), elf_version(3E)
NAME    elf32_getehdr, elf32_newehdr – retrieve class-dependent object file header

SYNOPSIS    cc [ flag ... ] file ... -l elf [ library ... ]

#include <libelf.h>
Elf32_Ehdr *elf32_getehdr(Elf *elf);
Elf32_Ehdr *elf32_newehdr(Elf *elf);

MT-LEVEL    Unsafe

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[EI_NIDENT];
  Elf32_Half e_type;
  Elf32_Half e_machine;
  Elf32_Word e_version;
  Elf32.Addr e_entry;
  Elf32_Off e_ploff;
  Elf32_Off e_shoff;
  Elf32_Word e_shnum;
  Elf32_Word e_shoffset;
  Elf32_Word 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.

SEE ALSO    elf(3E), elf_begin(3E), elf_flagdata(3E), elf_getident(3E)
NAME
elf32_getphdr, elf32_newphdr – retrieve class-dependent program header table

SYNOPSIS
cc [ flag ...] file ... -lelf [ library ...]
#include <libelf.h>
Elf32_Phdr *elf32_getphdr(Elf *elf);
Elf32_Phdr *elf32_newphdr(Elf *elf, size_t count);

MT-LEVEL
Unsafe

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 zero.

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.

SEE ALSO
elf(3E), elf32_getehdr(3E), elf_begin(3E), elf_flagdata(3E)

modified 22 Jan 1993

3E-335
NAME        elf32_getshdr – retrieve class-dependent section header

SYNOPSIS    cc [ flag . . . ] file . . . -l elf [ library . . . ]
#include <libelf.h>
Elf32_Shdr *elf32_getshdr(Elf_Scn *scn);

MT-LEVEL     Unsafe

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_flags;
    Elf32.Addr    sh_addr;
    Elf32.Off     sh_offset;
    Elf32.Word    sh_size;
    Elf32.Word    sh_link;
    Elf32.Word    sh_info;
    Elf32.Word    sh_addralign;
    Elf32.Word    sh_entsize;

If the program is building a new file, it is responsible for creating the file’s ELF header before creating sections.

SEE ALSO    elf(3E), elf_flagdata(3E), elf_getscn(3E), elf_strptr(3E)
### NAME
elf32_xlatetof, elf32_xlatetom – class-dependent data translation

### SYNOPSIS
```c
cc [ flag ... ] file ... -l elf [ 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);
```

### MT-LEVEL
Unsafe

### 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.

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>d_buf</code></td>
<td>Both the source and destination must have valid buffer pointers.</td>
</tr>
<tr>
<td><code>d_type</code></td>
<td>This member’s value specifies the type of the data to which <code>d_buf</code> points and the type of data to be created in the destination. The program supplies a <code>d_type</code> value in the source; the library sets the destination’s <code>d_type</code> to the same value. These values are summarized below.</td>
</tr>
<tr>
<td><code>d_size</code></td>
<td>This member holds the total size, in bytes, of the memory occupied by the source data and the size allocated for the destination data. If the destination buffer is not large enough, the routines do not change its original contents. The translation routines reset the destination’s <code>d_size</code> member to the actual size required, after the translation occurs. The source and destination sizes may differ.</td>
</tr>
<tr>
<td><code>d_version</code></td>
<td>This member holds version number of the objects (desired) in the buffer. The source and destination versions are independent.</td>
</tr>
</tbody>
</table>

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.
<table>
<thead>
<tr>
<th>Elf_Type</th>
<th>32-Bit Memory Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ELF_T_ADDR</td>
<td>Elf32_Addr</td>
</tr>
<tr>
<td>ELF_T_BYTE</td>
<td>unsigned char</td>
</tr>
<tr>
<td>ELF_T_DYN</td>
<td>Elf32_Dyn</td>
</tr>
<tr>
<td>ELF_T_EHDR</td>
<td>Elf32_Ehdr</td>
</tr>
<tr>
<td>ELF_T_HALF</td>
<td>Elf32_Half</td>
</tr>
<tr>
<td>ELT_T_OFF</td>
<td>Elf32_Off</td>
</tr>
<tr>
<td>ELF_T_PHDR</td>
<td>Elf32_Phdr</td>
</tr>
<tr>
<td>ELF_T_REL</td>
<td>Elf32_Rel</td>
</tr>
<tr>
<td>ELF_T_RELA</td>
<td>Elf32_Rea</td>
</tr>
<tr>
<td>ELF_T_SHDR</td>
<td>Elf32_Shdr</td>
</tr>
<tr>
<td>ELF_T_SWORD</td>
<td>Elf32_Sword</td>
</tr>
<tr>
<td>ELF_T_SYM</td>
<td>Elf32_Sym</td>
</tr>
<tr>
<td>ELF_T_WORD</td>
<td>Elf32_Word</td>
</tr>
</tbody>
</table>

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

**SEE ALSO** elf(3E), elf32_fsize(3E), elf_getdata(3E), elf_getident(3E)
NAME
elf_begin, elf_end, elf_memory, elf_next, elf_rand – process ELF object files

SYNOPSIS
cc [ flag ... ] file ... -l elf [ library ... ]
#include <libelf.h>
Elf *elf_begin(int fd, 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);

MT-LEVEL
Unsafe

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. fd 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, fd 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, \texttt{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 \texttt{elf_end()} once for each activation. See the examples below for more information.

**ELF\_C\_RDWR**

This command duplicates the actions of \texttt{ELF\_C\_READ} and additionally allows the program to update the file image (see \texttt{elf_update(3E)}). That is, using \texttt{ELF\_C\_READ} gives a read-only view of the file, while \texttt{ELF\_C\_RDWR} lets the program read and write the file. \texttt{ELF\_C\_RDWR} is not valid for archive members. If ref is non-null, it must have been created with the \texttt{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 \texttt{cmd} to this value. ref is ignored for this command.

\texttt{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 \texttt{elf_kind(3E)} or \texttt{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, \texttt{elf_begin()} returns a null pointer. Otherwise, the return value is a non-null ELF descriptor.

Before the first call to \texttt{elf_begin()}, a program must call \texttt{elf_version()} to coordinate versions.

\texttt{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 \texttt{elf_update()}. Calling \texttt{elf_end()} removes one activation and returns the remaining activation count. Consequently, a 0 return value indicates the ELF descriptor is no longer valid.

\texttt{elf_memory()} returns a pointer to an Elf descriptor, the Elf image has read operations enabled ( \texttt{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. You may read and modify an Elf image that is mapped in with \texttt{elf_memory()}, but you may not change the Elf image size.

\texttt{elf_next()} provides sequential access to the next archive member. That is, having an ELF descriptor, elf, associated with an archive member, \texttt{elf_next()} prepares the containing archive to access the following member when the program calls \texttt{elf_begin()}. After successfully positioning an archive for the next member, \texttt{elf_next()} returns the value \texttt{ELF\_C\_READ}. Otherwise, the open file was not an archive, elf was null, or an error occurred, and the return value is \texttt{ELF\_C\_NULL}. In either case, the return value may be passed as an argument to \texttt{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 two, and the program calls elf_end() twice to terminate the descriptor.
If the file is an archive, the loop processes each archive member in turn, ignoring those that are not object files.

```c
if (elf_version(EV_CURRENT) == EV_NONE)
{
    /* library out of date */
    /* recover from error */
}
```

```c
cmd = ELF_C_READ;
arf = elf_begin(fildes, cmd, (Elf *)0);
while ((elf = elf_begin(fildes, cmd, arf)) != 0)
{
    if ((ehdr = elf32_getehdr(elf)) != 0)
    {
        /* process the file ... */
    }
    cmd = elf_next(elf);
    elf_end(elf);
}
```

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
e lf_version(EV_CURRENT);
arf = elf_begin(fildes, ELF_C_READ, (Elf *)0);
if (elf_kind(arf) != ELF_K_AR)
{
    /* not an archive */
}
/* initial processing */
/* set offset = ... for desired member header */
while (elf_rand(arf, offset) == offset)
{
    if ((elf = elf_begin(fildes, ELF_C_READ, arf)) == 0)
        break;
    if ((ehdr = elf32_getehdr(elf)) != 0)
    {
        /* process archive member ... */
    }
    /* set offset = ... for desired member header */
}
An archive starts with a “magic string” that has $\text{SARMAG}$ 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)$\text{SARMAG}$) == $\text{SARMAG}$)
        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;
elfrd = elf32_newehdr(elf);
phdr = elf32_newphdr(elf, count);
sdn = elf_newscn(elf);
shdr = elf32_getshdr(sdn);
data = elf_newdata(sdn);
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`, the library uses it to enhance performance, and `mmap` requires a readable file descriptor. Although the library can use a write-only file
descriptor, the application will not obtain the performance advantages of `mmap`.

SEE ALSO `creat(2)`, `lseek(2)`, `mmap(2)`, `open(2)`, `elf(3E)`, `elf32_getehdr(3E)`, `elf_cntl(3E)`, `elf_getarhdr(3E)`, `elf_getarsym(3E)`, `elf_getbase(3E)`, `elf_getdata(3E)`, `elf_getscn(3E)`, `elf_kind(3E)`, `elf_rawfile(3E)`, `elf_update(3E)`, `elf_version(3E)`, `ar(4)`
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);

MT-LEVEL    Unsafe

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 zero. Otherwise elf was null or an error occurred, and the function returns −1.

SEE ALSO   elf(3E), elf_begin(3E), elf_getdata(3E), elf_rawfile(3E)

NOTES      If the program wishes to use the “raw” operations (see elf_rawdata(), which elf_getdata(3E) describes, and elf_rawfile(3E)) after disabling the file descriptor with ELF_C_FDDONE or ELF_C_FDREAD, it must execute the raw operations explicitly beforehand. Otherwise, the raw file operations will fail. Calling elf_rawfile() makes the entire image available, thus supporting subsequent elf_rawdata() calls.

modified 22 Jan 1993
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);

MT-LEVEL  Unsafe

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 zero, 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 zero.

(void)elf_errno();
/* processing ... */
while (more_to_do)
{
    if ((err = elf_errno()) != 0)
    {
        /* print msg */
        msg = elf_errmsg(err);
    }
}

SEE ALSO  elf(3E)
NAME      elf_fill – set fill byte

SYNOPSIS  cc [ flag ...] file ... -l elf [ library ...]
#include <libelf.h>
void elf_fill(int fill);

MT-LEVEL   Unsafe

DESCRIPTION Alignment constraints for ELF ®les sometimes require the presence of “holes.” For exam-
ple, 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.

SEE ALSO  elf(3E), elf_flagdata(3E), elf_getdata(3E), elf_update(3E)

NOTES     An application can assume control of the object ®le organization by setting the
ELF_F_LAYOUT bit (see elf_flagdata(3E)). When this is done, the library does not fill
holes.

modified 22 Jan 1993
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);

MT-LEVEL
Unsafe

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 zero 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 descriptor.

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 associated with the descriptor.
When a flag bit is set for an item, it affects all the subitems as well. Thus, for example, if the program sets the ELF_F_DIRTY bit with `elf_flagelf()`, the entire logical file is “dirty.”

**EXAMPLES**

The following fragment shows how one might mark the ELF header to be written to the output file:

```c
#include <elf.h>

/* dirty ehdr ... */

ehdr = elf32_getehdr(elf);
ell_flagehdr(elf, ELF_C_SET, ELF_F_DIRTY);
```

**SEE ALSO**

`elf(3E)`, `elf32_getehdr(3E)`, `elf_getdata(3E)`, `elf_update(3E)`
NAME  
elf_getarhdr - retrieve archive member header

SYNOPSIS  
cc [ flag . . . ] file . . . -l elf [ library . . . ]
#include <libelf.h>
Elf_Arhdr *elf_getarhdr(Elf *elf);

MT-LEVEL  Unsafe

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 termi-
inating 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.

SEE ALSO  elf(3E), elf_begin(3E), elf_getarsym(3E), ar(4)
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);

MT-LEVEL
Unsafe

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 associ-
ated symbol. Values in as_off may be passed as arguments 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 zero when the return value is NULL. The table’s last entry,
which is included in the count, has a NULL as_name, a zero value for as_off, and ^0UL
for as_hash.

The hash value returned is guaranteed not to be the bit pattern of all ones ( ^0UL).

SEE ALSO
elf(3E), elf_begin(3E), elf_getarhdr(3E), elf_hash(3E), ar(4)
NAME  elf_getbase – get the base offset for an object file

SYNOPSIS  cc [ flag ... ] file ... -l elf [ library ... ]
#include <libelf.h>
off_t elf_getbase(Elf *elf);

MT-LEVEL  Unsafe

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.

SEE ALSO  elf(3E), elf_begin(3E), ar(4)
NAME
df_getdata, df_newdata, df_rawdata – get section data

SYNOPSIS
c[flag...][file...−elf][library...]
#include <libelf.h>
Elf_Data*df_getdata(Elf_Scn*scn, Elf_Data*data);
Elf_Data*df_newdata(Elf_Scn*scn);
Elf_Data*df_rawdata(Elf_Scn*scn, Elf_Data*data);

MT-LEVEL
Unsafe

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.

df_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, df_getdata() returns a null pointer.

df_getdata() translates the data from file representations into memory representations
(see df32_xlatetof(3E)) and presents objects with memory data types to the program,
based on the file’s class (see df(3E)). The working library version (see df_version(3E))
specifies what version of the memory structures the program wishes df_getdata() to
present.

df_newdata() creates a new data descriptor for a section, appending it to any data ele-
ments 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 df_flagdata(3E)). If scn is null or an error occurs, df_newdata() returns a null pointer.

df_rawdata() differs from df_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
df_cntl(3E)) the file descriptor associated with elf before the initial raw operation,
because df_rawdata() might read the data from the file to ensure it doesn’t interfere with
df_getdata(). See df_rawfile(3E) for a related facility that applies to the entire file.
When df_getdata() provides the right translation, its use is recommended over
df_rawdata(). If scn is null or an error occurs, df_rawdata() returns a null pointer.

modified 27 Oct 1994 3E-353
The Elf_Data structure includes the following members:

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

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

**d_buf**
A pointer to the data buffer resides here. A data element with no data has a null pointer.

**d_type**
This member’s value specifies the type of the data to which d_buf points. A section’s type determines how to interpret the section contents, as summarized below.

**d_size**
This member holds the total size, in bytes, of the memory occupied by the data. This may differ from the size as represented in the file. The size will be zero if no data exist. (See the discussion of SHT_NOBITS below for more information.)

**d_off**
This member gives the offset, within the section, at which the buffer resides. This offset is relative to the file’s section, not the memory object’s.

**d_align**
This member holds the buffer’s required alignment, from the beginning of the section. That is, d_off will be a multiple of this member’s value. For example, if this member’s value is four, the beginning of the buffer will be four-byte aligned within the section. Moreover, the entire section will be aligned to the maximum of its constituents, thus ensuring appropriate alignment for a buffer within the section and within the file.

**d_version**
This member holds the version number of the objects in the buffer. When the library originally read the data from the object file, it used the working version to control the translation to memory objects.

**DATA ALIGNMENT**
As mentioned above, data buffers within a section have explicit alignment constraints. Consequently, adjacent buffers sometimes will not abut, causing “holes” within a section. Programs that create output files have two ways of dealing with these holes.

First, the program can use `elf_fill()` to tell the library how to set the intervening bytes. When the library must generate gaps in the file, it uses the fill byte to initialize the data there. The library’s initial fill value is zero, and `elf_fill()` lets the application change that.

Second, the application can generate its own data buffers to occupy the gaps, filling the gaps with values appropriate for the section being created. A program might even use different fill values for different sections. For example, it could set text sections’ bytes to no-operation instructions, while filling data section holes with zero. Using this technique, the library finds no holes to fill, because the application eliminated them.
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_Rela</td>
</tr>
<tr>
<td>SHT_SYMTAB</td>
<td>ELF_T_SYM</td>
<td>Elf32_Sym</td>
</tr>
<tr>
<td>SHT_SUNW_verdef</td>
<td>ELF_T_VDEF</td>
<td>Elf32_Verdef</td>
</tr>
<tr>
<td>SHT_SUNW_verneed</td>
<td>ELT_T_VNEEDElf32_Verneed</td>
<td></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.

SEE ALSO elf, elf32_getehdr(3E), elf32_getshdr(3E), elf32_xlatetof(3E), elf_cntl(3E), elf_fill(3E), elf_flagdata(3E), elf_getscn(3E), elf_rawfile(3E), elf_strptr(3E), elf_version(3E)
NAME    elf_getident – retrieve file identification data

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

MT-LEVEL Unsafe

DESCRIPTION As elf(3E) explains, ELF provides a framework for various classes of files, where basic objects may have 32 bits, 64 bits, etc. To accommodate these differences, without forcing the larger sizes on smaller machines, the initial bytes in an ELF file hold identification information common to all file classes. Every ELF header's e_ident has EI_NIDENT bytes with the following interpretation.

<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>ELFCLASSNONE</td>
<td>File class</td>
</tr>
<tr>
<td></td>
<td>ELFCLASS32</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ELFCLASS64</td>
<td></td>
</tr>
<tr>
<td>EI_DATA</td>
<td>ELFDATA2LSB</td>
<td>Data encoding</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 zero stored through ptr, if ptr is non-null.
SEE ALSO  elf(3E), elf32_getehdr(3E), elf_begin(3E), elf_kind(3E), elf_rawfile(3E)
NAME       elf_getscn, elf_ndxscn, elf_newscn, elf_nextscn – get section information

SYNOPSIS   cc [ flag ... ] file ... -lelf [ library ... ]

#include <libelf.h>

Elf_Scn *elf_getscn(Elf *elf, size_t index);
size_t elf_ndxscn(Elf_Scn *scn);
Elf_Scn *elf_newscn(Elf *elf);
Elf_Scn *elf_nextscn(Elf *elf, Elf_Scn *scn);

MT-LEVEL    Unsafe

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 the first "real" section has index 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 associated data (see elf_getdata(3E)). When creating a new section in this way, the library updates the e_shnum member of the ELF header and sets the ELF_F_DIRTY bit for the section (see elf_flagdata(3E)). If the program is building a new file, it is responsible for creating the file's ELF header (see elf32_getehdr(3E)) before creating new sections.

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.

```c
scn = 0;
while ((scn = elf_nextscn(elf, scn)) != 0)
{
    /* process section */
}
```

SEE ALSO

elf(3E), elf32_getehdr(3E), elf32_getshdr(3E), elf_begin(3E), elf_flagdata(3E), elf_getdata(3E)
NAME     elf_hash – compute hash value

SYNOPSIS  cc [ flag ... ] file ... -l elf [ library ... ]
#include <libelf.h>
unsigned long elf_hash(const char *name);

MT-LEVEL  Unsafe

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").

SEE ALSO  elf(3E), elf_getdata(3E), elf32_xlatetof(3E)
NAME
elf_kind – determine file type

SYNOPSIS
c [ flag ... ] file ... -l elf [ library ... ]
#include <libelf.h>
Elf_Kind elf_kind(Elf *elf);

MT-LEVEL
Unsafe

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.

SEE ALSO elf(3E), elf32_getehdr(3E), elf_begin(3E), elf_getident(3E), ar(4)
NAME  
elf_rawfile – retrieve uninterpreted file contents

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

MT-LEVEL    Unsafe

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 implicit-
ly 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() (see elf_getdata(3E)) is a related function, providing access to sections
within a file.

If ptr is non-null, the library also stores the file’s size, in bytes, in the location to which ptr
points. If no data are present, elf is null, or an error occurs, the return value is a null
pointer, with zero stored through ptr, if ptr is non-null.

SEE ALSO  
elf(3E), elf32_getehdr(3E), elf_begin(3E), elf_cntl(3E), elf_getdata(3E), elf_getident(3E),
elf_kind(3E)

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.
elf_strptr (3E)

NAME
elf_strptr – make a string pointer

SYNOPSIS
c$[flag...].file...−$elf[library...]
#include <libelf.h>
char *elf_strptr(Elf *elf, size_t section, size_t offset);

MT-LEVEL
Unsafe

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.

/* 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)");
}

SEE ALSO
elf(3E), elf32_getshdr(3E), elf32_xlatetof(3E), elf_getdata(3E)

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().

3E-364
modified 22 Jan 1993
NAME
elf_update – update an ELF descriptor

SYNOPSIS
cc [ flag ...] file ... -lelf [ library ...]
#include <libelf.h>
off_t elf_update(EIff *elf, Elf_Cmd cmd);

MT-LEVEL
Unsafe

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_xlateto(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.
## ELF Header members:

<table>
<thead>
<tr>
<th>Member</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>e_ident[EL_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_phoff</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>

## 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>

## 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 zero.

When deciding how to build the output file, elf_update() obeys the alignments of individual data buffers to create output sections. A section’s most strictly aligned data buffer controls the section’s alignment. The library also inserts padding between buffers, as necessary, to ensure the proper alignment of each buffer.

**SEE ALSO**
elf(3E), elf32_fsize(3E), elf32_getehdr(3E), elf32_getshdr(3E), elf32_xlatetof(3E), elf_begin(3E), elf_flagdata(3E), elf_getdata(3E)

**NOTES**
As mentioned above, the ELF_C_WRITE command translates data as necessary, before writing them to the file. This translation is *not* always transparent to the application program. If a program has obtained pointers to data associated with a file (for example, see elf32_getehdr(3E) and elf_getdata(3E)), the program should reestablish the pointers after calling elf_update().

modified 22 Jan 1993
NAME    elf_version – coordinate ELF library and application versions
SYNOPSIS cc [ flag ... ] file ... -l elf [ library ... ]
          #include <libelf.h>
          unsigned elf_version(unsigned ver);
MT-LEVEL Unsafe
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 */
    }

SEE ALSO elf(3E), elf32_wordtof(3E), elf_begin(3E)

NOTES The working version should be the same for all operations on a particular elf descriptor. Changing the version between operations on a descriptor will probably not give the expected results.
NAME    end, _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 stan-
         dard 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      erf, erfc – error functions

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

MT-LEVEL  MT-Safe

DESCRIPTION erf(x) returns the error function of x; where
           \( \text{erf}(x) := \frac{2}{\sqrt{\pi}} \int_0^x \exp(-t^2) \, dt \).

   erfc(x) returns 1.0–erf(x), computed however by other methods that avoid cancellation
   for large x.
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);

MT-LEVEL MT-Safe

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).

modified 13 Jul 1994
FILES
/etc/ethers
/etc/nsswitch.conf

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

BUGS
Programs that call ether_hostton() or ether_ntohost() routines cannot be linked statically since the implementation of these routines requires dynamic linker functionality to access shared objects at run time.
**NAME**  
euclen, euccol, eucscol – get byte length and display width of EUC characters

**SYNOPSIS**  
#include <euc.h>  
int euclen(const unsigned char *s);  
int euccol(const unsigned char *s);  
int eucscol(const unsigned char *str);

**MT-LEVEL**  
Safe

**DESCRIPTION**  
euclen() returns the length in bytes of the Extended Unix Code (EUC) character pointed to by s, including single-shift characters, if present.  
euccol() returns the screen column width of the EUC character pointed to by s.  
eucscol() returns the screen column width of the EUC string pointed to by str.  
For the euclen() and euccol(), routines, s points to the first byte of the character. This byte is examined to determine its codeset. The character type table for the current locale is used for codeset byte length and display width information.

**FILES**  
/usr/lib/locale/locale/LC_CTYPE

**SEE ALSO**  
getwidth(3I)
### NAME
exit – terminate process

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

void exit(int status);
```

### MT-LEVEL
Safe

### DESCRIPTION
The C function `exit()` first calls any functions registered through the `atexit(3C)` function in the reverse order of their registration. `exit()` then calls `_exit`, which circumvents all such functions and cleanup.

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.

### SEE ALSO
`exit(2)`, `atexit(3C)`
NAME  exp, expm1, log, log1p, log10, pow – exponential, logarithm, power

SYNOPSIS  cc [ flag ... ] file ... -lm [ library ... ]
#include <math.h>
double exp(double x);
double expm1(double x);
double log(double x);
double log1p(double x);
double log10(double x);
double pow(double x, double y);

MT-LEVEL  MT-Safe

DESCRIPTION  exp(x) computes the exponential function $e^{**x}$.
expm1(x) computes $(e^{**x})-1$ accurately even for tiny $x$.
log(x) computes the natural logarithm of $x$.
log1p(x) computes $\log(1+x)$ accurately even for tiny $x$.
log10(x) computes the base-10 logarithm of $x$.
pow(x, y) computes $x$ raised to the power $y$.

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

SEE ALSO  matherr(3M)

DIAGNOSTICS  In IEEE754 mode (i.e. the -xlibmieee cc compilation option), log(±0) returns $-\infty$ and raises the division by zero exception; if $x<0$, log(x) returns a NaN and raises the invalid operation exception; if $x = +\infty$ or a quiet NaN, log(x) returns $x$ and raises no exception; if $x$ is a signaling NaN, log(x) returns a quiet NaN and raises the invalid operation exception; log(1) returns 0 and raises no exception; for all other positive $x$, log(x) returns a normalized number and raises the inexact exception.
NAME  fattach – attach a STREAMS-based file descriptor to an object in the file system name space

SYNOPSIS  int fattach(int fildes, const char *path);

MT-LEVEL  MT-Safe

DESCRIPTION  The fattach() routine 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  fildes is not a valid open file descriptor.

EBUSY  path is currently a mount point or has a STREAMS file descriptor attached it.

EINVAL  path is a file in a remotely mounted directory.

EINVAL  fildes 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  path does not exist.
<table>
<thead>
<tr>
<th>ENOTDIR</th>
<th>A component of a path prefix is not a directory.</th>
</tr>
</thead>
<tbody>
<tr>
<td>EPERM</td>
<td>The effective user ID is not the owner of <code>path</code> or a user with the appropriate privileges.</td>
</tr>
</tbody>
</table>

**SEE ALSO**
- `fdetach(1M)`, `chmod(2)`, `stat(2)`, `fdetach(3C)`, `isastream(3C)`, `streamio(7I)`
- *STREAMS Programming Guide*
NAME  fclose, fflush – close or flush a stream

SYNOPSIS  
```c
#include <stdio.h>
int fclose(FILE *stream);
int fflush(FILE *stream);
```

MT-LEVEL  MT-Safe

DESCRIPTION  fclose() causes any buffered data waiting to be written for the named stream (see intro(3)) to be written out, and the stream to be closed. If the underlying file pointer is not already at end of file, and the file is one capable of seeking, the file pointer is adjusted so that the next operation on the open file pointer deals with the byte after the last one read from or written to the file being closed.

fclose() is performed automatically for all open files upon calling exit().

If stream points to an output stream or an update stream on which the most recent operation was not input, fflush() causes any buffered data waiting to be written for the named stream to be written to that file. Any unread data buffered in stream is discarded. The stream remains open. If stream is open for reading, the underlying file pointer is not already at end of file, and the file is one capable of seeking, the file pointer is adjusted so that the next operation on the open file pointer deals with the byte after the last one read from or written to the stream.

When calling fflush(), if stream is a null pointer, all files open for writing are flushed.

RETURN VALUES  Upon successful completion these functions return a value of zero. Otherwise EOF is returned.

SEE ALSO  close(2), exit(2), intro(3), fopen(3S), setbuf(3S), stdio(3S)

modified 22 Jan 1993
NAME  
fdatasync – synchronize a file’s data

SYNOPSIS  
cc [ flag ... ] file ... -lposix4 [ library ... ]  
#include <unistd.h>  
int fdatasync(int fildes);

MT-LEVEL  
Async-Signal-Safe

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 conditions defined for read(2) and write(2).

SEE ALSO  
fcntl(2), open(2), read(2), write(2), fsync(3C), aio_fsync(3R), fcntl(5)

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

modified 19 Aug 1993
NAME fdetach – detach a name from a STREAMS-based file descriptor

SYNOPSIS int fdetach(const char *path);

MT-LEVEL Unsafe

DESCRIPTION The fdetach() routine detaches a STREAMS-based file descriptor from a name in the file system. path is the path name of the object in the file system name space, which was previously attached (see fattach(3C)). The user must be the owner of the file or a user with the appropriate privileges. All subsequent operations on path will operate on the file system node and not on the STREAMS file. The permissions and status of the node are restored to the state the node was in before the STREAMS file was attached to it.

RETURN VALUES If successful, fdetach() returns 0; otherwise it returns -1 and sets errno to indicate an error.

ERRORS Under the following conditions, the function fdetach() fails and sets errno to:

EINVAL path is not attached to a STREAMS file.
ELOOP Too many symbolic links were encountered in translating path.
ENAMETOOLONG The size of path exceeds [PATH_MAX], or a path name component is longer than [NAME_MAX] while [POSIX_NO_TRUNC] is in effect.
ENOENT path does not exist.
ENOTDIR A component of the path prefix is not a directory.
EPERM The effective user ID is not the owner of path or is not a user with appropriate permissions.

SEE ALSO fdetach(1M), fattach(3C), streamio(7I)

STREAMS Programming Guide
# NAME

ferror, feof, clearerr, fileno – stream status inquiries

# SYNOPSIS

```c
#include <stdio.h>

int ferror(FILE *stream);
int feof(FILE *stream);
void clearerr(FILE *stream);
int fileno(FILE *stream);
```

# MT-LEVEL

MT-Safe

# 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).

# SEE ALSO

open(2), intro(3), fopen(3S), stdio(3S)
NAME       ffs – find first set bit
SYNOPSIS   #include <string.h>
            int ffs(const int i);
MT-LEVEL    MT-Safe
DESCRIPTION ffs() finds the first bit set in the argument passed it and returns the index of that bit. Bits are numbered starting at 1 from the low order bit. A return value of zero indicates that the value passed is zero.
NAME       floating_to_decimal, single_to_decimal, double_to_decimal, extended_to_decimal, quadruple_to_decimal – convert floating-point value to decimal record

SYNOPSIS   
#include <floatingpoint.h>

void single_to_decimal(single *px, decimal_mode *pm, decimal_record *pd, 
  fp_exception_field_type *ps);
void double_to_decimal(double *px, decimal_mode *pm, decimal_record *pd, 
  fp_exception_field_type *ps);
void extended_to_decimal(extended *px, decimal_mode *pm, decimal_record *pd, 
  fp_exception_field_type *ps);
void quadruple_to_decimal(quadruple *px, decimal_mode *pm, decimal_record *pd, 
  fp_exception_field_type *ps);

MT-LEVEL    MT-Safe

DESCRIPTION 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 termi-
nate 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().

SEE ALSO econvert(3), fconvert(3), gconvert(3), printf(3S), sprintf(3S)
NAME       flock – apply or remove an advisory lock on an open file

SYNOPSIS  /usr/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.

modified 19 Jul 1994
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);

POSIX cc [ flag ... ] file ... -D_POSIX_PTHREAD_SEMANTICS [ library ... ]
int ftrylockfile(FILE *stream);

MT-LEVEL MT-Safe

DESCRIPTION flockfile() 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.

funlockfile() releases the lock being held by the current thread. In the case of
recursive locking, this function must be called the same number of times flockfile() was
called. After the number of funlockfile() calls is equal to the number of flockfile() calls,
the stream lock is available for other threads to acquire.

ftrylockfile() 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 ftrylockfile() returns zero on success or non-zero to indicate a lock
cannot be acquired.

EXAMPLES For example:

    FILE iop;
    ...
    flockfile(iop);
    fprintf(iop,"hello ");
    fprintf(iop,"world0);
    fputc(iop, 'a');
    funlockfile(iop);

will print everything out together, blocking other threads that might want to write to
the same file between fprintf()s.

An unlocked interface is available in case performance is an issue. For example:

    flockfile(iop);
    while (!feof(iop)) {
        *c++ = getc_unlocked(iop);
    }
    funlockfile(iop);

modified 22 Aug 1995 3S-387
SEE ALSO intro(3), ferror(3S), getc(3S), putc(3S), stdio(3S), ungetc(3S)

NOTES When compiling multi-thread applications, the _REENTRANT flag must be defined on the compile line. This flag should only be used in multi-thread applications. The interfaces on this page are as specified in POSIX 1003.1c Draft #10.
NAME  floor, ceil, rint – round to integral value in floating-point format

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

MT-LEVEL  MT-Safe

DESCRIPTION  ceil(), floor() and rint() convert a double value into an integral value in double format. They vary in how they choose the result when the argument is not already an integral value. Here an “integral value” means a value of a mathematical integer, which however might be too large to fit in a particular computer’s int format. All sufficiently large values in a particular floating-point format are already integral; in IEEE 754 double-precision format, that means all values >= 2**52. Zeros, infinities, and quiet NaNs are treated as integral values by these functions, which always preserve their argument’s sign.

ceil() returns the least integral value greater than or equal to x. This corresponds to IEEE 754 rounding toward positive infinity.

floor() returns the greatest integral value less than or equal to x. This corresponds to IEEE 754 rounding toward negative infinity.

rint() rounds x to an integral value according to the current IEEE 754 rounding direction.
NAME
fmtmsg – display a message on stderr or system console

SYNOPSIS
#include <fmtmsg.h>

int fmtmsg(long classification, const char *label, int severity, const char *text,
const char *action, const char *tag);

MT-LEVEL
Safe

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.

**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:

```
export MSGVERB
```

Valid keywords are: **label**, **severity**, **text**, **action**, and **tag**. If **MSGVERB** contains a keyword for a component and the component’s value is not the component’s null value, **fmtmsg()** includes that component in the message when writing the message to stderr. If **MSGVERB** does not include a keyword for a message component, that component is not included in the display of the message. The keywords may appear in any order. If **MSGVERB** is not defined, if its value is the null-string, if its value is not of the correct format, or if it contains keywords other than the valid ones listed above, **fmtmsg()** selects all components.

---

modified 22 Jan 1993
The first time `fmtmsg()` is called, it examines the `MSGVERB` environment variable to see which message components it is to select when generating a message to write to the standard error stream, `stderr`. The values accepted on the initial call are saved for future calls.

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

`SEV_LEVEL` defines severity levels and associates print strings with them for use by `fmtmsg()` . The standard severity levels shown below cannot be modified. Additional severity levels can also be defined, redefined, and removed using `addseverity()` (see `addseverity(3C)`). If the same severity level is defined by both `SEV_LEVEL` and `addseverity()` , the definition by `addseverity()` is controlling.

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

`SEV_LEVEL` can be set as follows:

```
SEV_LEVEL=[description[description[...]]]
export SEV_LEVEL
```

description is a comma-separated list containing three fields:

- `description=severity_keyword,level,printstring`

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

level is a character string that evaluates to a positive integer (other than 0, 1, 2, 3, or 4, which are reserved for the standard severity levels). If the keyword severity_keyword is used, level is the severity value passed on to the `fmtmsg()` function.

printstring is the character string used by `fmtmsg()` in the standard message format whenever the severity value level is used.

If a description in the colon list is not a three-field comma list, or, if the second field of a comma list does not evaluate to a positive integer, that description in the colon list is ignored.

The first time `fmtmsg()` is called, it examines the `SEV_LEVEL` environment variable, if defined, to see whether the environment expands the levels of severity beyond the five standard levels and those defined using `addseverity()` . The values accepted on the initial call are saved for future calls.
Use in Applications

One or more message components may be systematically omitted from messages generated by an application by using the null value of the argument for that component. The table below indicates the null values and identifiers for `fmtmsg()` arguments.

<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()`:

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

produces a complete message in the standard message format:

UX:cat: ERROR: invalid syntax
TO FIX: refer to manual UX:cat:001

Example 2:

When the environment variable `MSGVERB` is set as follows:

```
MSGVERB=severity:text:action
```

and the Example 1 is used, `fmtmsg()` produces:

ERROR: invalid syntax
TO FIX: refer to manual
Example 3:
When the environment variable SEV_LEVEL is set as follows:

```
SEV_LEVEL=note,5,NOTE
```
the following call to `fmtmsg()`:

```
fmtmsg(MM(Util | MM_PRINT, "UX:cat", 5, "invalid syntax", "refer to manual", "UX:cat:001")
```

produces:

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

SEE ALSO  `fmtmsg(1)`, `addseverity(3C)`, `gettext(3C)`, `printf(3S)`
NAME 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);

MT-LEVEL Safe

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 VALUE 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).

APPLICATION 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().

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_attributes(3N), xfn_status_codes(3N),
xfn(3N)

modified 4 Nov 1994 3N-395
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);

MT-LEVEL
Safe.

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 VALUE
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).

APPLICATION
The attributes in the returned set do not contain the syntax or values of the attributes,
only their identifiers.

USAGE
SEE ALSO
FN_attrset_t(3N), FN_attribute_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_attributes(3N),
xfn_status_codes(3N), xfn(3N)
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);

MT-LEVEL Safe.

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 similar fashion as the fn_ctx_list_names() operations.

RETURN VALUE 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 value 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.

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).

APPLICATION

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, a fn_ctx_handle_destroy() should not be invoked
on ctx until the enumeration has terminated.

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_list_names(3N),
xfn_attributes(3N), xfn_status_codes(3N), xfn(3N)
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);

MT-LEVEL Safe.

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 VALUE | This operation returns 1 if the operation succeeds, 0 if the operation fails. |
| ERRORS       | `fn_attr_modify()` sets `status` as described in `FN_status_t(3N)` and `xfn_status_codes(3N)`. |
| SEE ALSO     | `FN_composite_name_t(3N)`, `FN_ctx_t(3N)`, `FN_attribute_t(3N)`, `FN_status_t(3N)`, `fn_attr_multi_modify(3N)`, `xfn_attributes(3N)`, `xfn_status_codes(3N)`, `xfn(3N)` |
NAME
fn_attr_multi_get, FN_multigetlist_t, fn_multigetlist_next, fn_multigetlist_destroy – return multiple attributes associated with named object

SYNOPSIS
c [ 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);

MT-LEVEL
Safe.

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 attributes associated with the named object are returned. Any attribute values in attr_ids provided by the caller are ignored; only the attribute identifiers are relevant for this operation. Each attribute (identifier, syntax, values) is returned one at a time using an enumeration scheme similar to that for listing a context.

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 enumeration. It may be invoked before the enumeration has completed to terminate the enumeration.

RETURN VALUE
fn_attr_multi_get() returns a pointer to an FN_multigetlist_t object if the enumeration has been initiated successfully; a NULL pointer 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

modified 4 Nov 1994 3N-401
The caller did not have permission to read this attribute.

**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)**.

**EXAMPLES**
The following code fragment illustrates to obtain all attributes associated with a given name using the **fn_attr_multi_get()** operations.

```c
/* 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) {
```
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' */

APPLICATION 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.

SEE ALSO FN_attrset_t(3N), FN_attribute_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_list_names(3N), xfn_attributes(3N), xfn_status_codes(3N), xfn(3N)
NAME fn_attr_multi_modify – modify multiple attributes associated with named object

SYNOPSIS cc [ flag ... ] file ... -l xfn [ 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);

MT-LEVEL Safe.

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 VALUE fn_attr_multi_modify() returns 1 if all the modification operations were performed successfully. The function returns 0 if it 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 is returned in unexecuted_mods, that indicates no modifications were executed.

SEE ALSO FN_attrmodlist_t(3N), FN_composite_name_t(3N), FN_ctx_t(3N), FN_status_t(3N), fn_attr_modify(3N), xfn_attributes(3N), xfn_status_codes(3N), xfn(3N)
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);

MT-LEVEL  Safe.

DESCRIPTION This operation binds the supplied reference ref to the supplied composite name name relative to ctx. The binding is made in the target context — 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 zero, the new binding replaces any existing binding.

RETURN VALUE When the bind operation is successful it returns 1; on error it returns 0.

ERRORS fn_ctx_bind sets status as described in FN_status_t(3N) and xfn_status_codes. Of special relevance for this operation is the following status code:

FN_E_NAME_IN_USE

The supplied name is already in use.

APPLICATION USAGE

The value of ref cannot be NULL. If the intent is to reserve a name using fn_ctx_bind(), a reference containing no address should be supplied. This reference may be name service-specific or it may be the conventional NULL reference defined in the X/Open registry (see fns_references(5)).

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.

SEE ALSO FN_composite_name_t(3N), FN_ctx_t(3N), FN_ref_t(3N), FN_status_t(3N), fn_ctx_lookup(3N), fn_ctx_unbind(3N), xfn_status_codes(3N), xfn(3N)

modified 4 Nov 1994  3N-405
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);

MT-LEVEL

Safe.

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 opera-
tion 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.

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)
fn_ctx_destroy_subcontext – destroy the named context and remove its binding from the parent context

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.

fn_ctx_destroy_subcontext() returns 1 on success and 0 on failure.

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.

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.

modified 4 Nov 1994
### 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_status_codes(3N)`, `xfn(3N)`
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);

MT-LEVEL
Safe.

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().

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)

modified 4 Nov 1994 3N-409
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_attr(FN_ctx_t *ctx,
    const FN_composite_name_t *name, FN_status_t *status);

MT-LEVEL Safe.

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 attri-
butes 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.

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)
<table>
<thead>
<tr>
<th>NAME</th>
<th>fn_ctx_handle_destroy – release storage associated with context handle</th>
</tr>
</thead>
</table>
| SYNOPSIS | cc [ flag ... ] file ... −lxfn [ library ... ]  
#include <xfn/xfn.h>  
void fn_ctx_handle_destroy(FN_ctx_t *ctx); |
| MT-LEVEL | Safe. |
| 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. |
| SEE ALSO | FN_ctx_t(3N), fn_ctx_handle_from_initial(3N), fn_ctx_handle_from_ref(3N), xfn(3N) |
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(FN_status_t *status);

MT-LEVEL MT-Safe.

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 fns_initial_context(5)).

RETURN VALUE fn_ctx_handle_from_initial() returns a pointer to an FN_ctx_t object if the operation succeeds; it returns a NULL pointer (0) otherwise.

ERRORS fn_ctx_handle_from_initial() sets only the status code portion of the status object status.

SEE ALSO FN_ctx_t(3N), FN_status_t(3N), fns_initial_context(5), fn_ctx_get_ref(3N), fn_ctx_handle_from_ref(3N), xfn_status_codes(3N), xfn(3N)
fn_ctx_handle_from_ref – construct a handle to a context object using the given reference

cc [ flag ... ] file ... -lxfn [ library ... ]
#include <xfn/xfn.h>
FN_ctx_t *fn_ctx_handle_from_ref(const FN_ref_t *ref, FN_status_t *status);

<table>
<thead>
<tr>
<th>MT-LEVEL</th>
<th>Safe.</th>
</tr>
</thead>
</table>

This operation creates a handle to an FN_ctx_t object using an FN_ref_t object for that context.

This operations returns a pointer to an FN_ctx_t object if the operation succeeds, otherwise, it returns a NULL pointer (0).

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.

SEE ALSO FN_ctx_t(3N), FN_ref_t(3N), FN_status_t(3N), fn_ctx_handle_destroy(3N), fn_ctx_get_ref(3N), fns_references(5), xfn_status_codes(3N), xfn(3N)
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);

MT-LEVEL
Safe.

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.

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_status_codes(3N), xfn(3N)
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);

MT-LEVEL
Safe.

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 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 VALUE
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.

fn_namelist_next() returns a NULL pointer 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, it indicates that no more names can be returned. status is set in the following way:

FN_SUCCESS
   The enumeration has completed successfully.

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)**.

**EXAMPLE**

The following code fragment illustrates a 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);
}

/* check ’status’ for reason for end of enumeration and report if necessary */

/* clean up */
fn_namelist_destroy(nl, status);
/* report ’status’ */
```

**APPLICATION 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
element 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()` should not be invoked on `ctx` until the enumeration has terminated.

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_status_codes(3N)`, `xfn(3N)`
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);

MT-LEVEL Safe.

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).

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)
### NAME
fn_ctx_lookup_link – look up the link reference bound to a name

### SYNOPSIS
```c
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);
```

### MT-LEVEL
Safe.

### 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` 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 VALUE
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.

### SEE ALSO
- `FN_composite_name_t(3N)`, `FN_ctx_t(3N)`, `FN_ref_t(3N)`, `FN_status_t(3N)`, `fn_ctx_lookup(3N)`, `xfn_status_codes(3N)`, `xfn_links(3N)`, `xfn(3N)`

---

modified 4 Nov 1994
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);

MT-LEVEL
Safe.

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 zero, the operation overwrites any old binding of newname. If exclusive is
nonzero, the operation fails if newname is already bound.

RETURN VALUE
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).

APPLICATION
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 re-
stricted to be an atomic name.

Normal resolution always follows links. In a 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.

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_status_codes(3N), xfn(3N)
### NAME
fn_ctx_unbind – unbind a name from a context

### SYNOPSIS
```c
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);
```

### MT-LEVEL
Safe.

### DESCRIPTION
This operation removes the terminal atomic name in `name` from 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.

### 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)`
NAME  fnmatch – match filename or path name

SYNOPSIS  
#include <fnmatch.h>
int fnmatch( const char *pattern, const char *string, int flags);

MT-LEVEL  MT-Safe

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` 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.

**SEE ALSO** `find(1), glob(3C), wordexp(3C), fnmatch(5)`
fopen(3B)  SunOS/BSD Compatibility Library Functions  SunOS 5.5

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, freopen, fdopen – open a stream

SYNOPSIS

```c
#include <stdio.h>

FILE *fopen(const char *filename, const char *type);
FILE *freopen(const char *filename, const char *type, FILE *stream);
FILE *fdopen(int fildes, const char *type);
```

MT-LEVEL  MT-Safe

DESCRIPTION  `fopen()` opens the file named by `filename` and associates a stream with it. `fopen()` returns a pointer to the FILE structure associated with the stream.

`filename` points to a character string that contains the name of the file to be opened.

type is a character string beginning with one of the following sequences:

- "r" or "rb"  open for reading
- "w" or "wb"  truncate to zero length or create for writing
- "a" or "ab"  append; open for writing at end of file, or create for writing
- "r+", "r+b" or "rb+"  open for update (reading and writing)
- "w+", "w+b" or "wb+"  truncate or create for update
- "a+", "a+b" or "ab+"  append; open or create for update at end-of-file

The "b" is ignored in the above types. The "b" exists to distinguish binary files from text files. However, there is no distinction between these types of files on a UNIX system.

`freopen()` substitutes the named file in place of the open stream. A flush is first attempted, and then the original stream is closed, regardless of whether the open ultimately succeeds. Failure to flush or close stream successfully is ignored. `freopen()` returns a pointer to the FILE structure associated with stream.

`freopen()` 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.

`fdopen()` associates a stream with a file descriptor. File descriptors are obtained from open(2), dup(2), creat(2), or pipe(2), which open files but do not return pointers to a FILE structure stream. Streams are necessary input for almost all of the Section 3S library routines. The type of stream must agree with the mode of the open file. The file position indicator associated with stream is set to the position indicated by the file offset associated with fildes.

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 fflush(), fseek(), fsetpos(), or rewind(), and input may not be directly followed by output without an intervening fseek(), fsetpos(), or rewind(), or an input operation that encounters end-of-file.
When a file is opened for append (that is, when `type` is “a”, “ab”, “a+”, or “ab+”), it is impossible to overwrite information already in the file. `fseek()` may be used to reposition the file pointer to any position in the file, but when output is written to the file, the current file pointer is disregarded. All output is written at the end of the file and causes the file pointer to be repositioned at the end of the output. If two separate processes open the same file for append, each process may write freely to the file without fear of destroying output being written by the other. The output from the two processes will be intermixed in the file in the order in which it is written.

When opened, a *stream* is fully buffered if and only if it can be determined not to refer to an interactive device. The error and end-of-file indicators are cleared for the *stream*.

**RETURN VALUES**

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

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

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

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

**SEE ALSO**

`close(2)`, `creat(2)`, `dup(2)`, `open(2)`, `pipe(2)`, `write(2)`, `fclose(3S)`, `fseek(3S)`, `setbuf(3S)`, `stdio(3S)`
NAME form_cursor, pos_form_cursor – position forms window cursor

SYNOPSIS cc [ flag ... ] file ... -lform -lcurses [ library ... ]
#include <form.h>
int pos_form_cursor(FORM *form);

MT-LEVEL Unsafe

DESCRIPTION pos_form_cursor() moves the form window cursor to the location required by the form driver to resume form processing. This may be needed after the application calls a curses library I/O routine.

RETURN VALUES pos_form_cursor() returns one of the following:

- E_OK – The function returned successfully.
- E_SYSTEM_ERROR – System error.
- E_BAD_ARGUMENT – An argument is incorrect.
- E_NOT_POSTED – The form is not posted.

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

NOTES The header <form.h> automatically includes the headers <eti.h> and <curses.h>. 
NAME  form_data, data_ahead, data_behind – tell if forms field has off-screen data ahead or behind

SYNOPSIS  cc [ flag ... ] file ... -lform -lcurses [ library ... ]
#include <form.h>
int data_ahead(FORM *form);
int data_behind(FORM *form);

MT-LEVEL  Unsafe

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).

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

NOTES  The header <form.h> automatically includes the headers <eti.h> and <curses.h>.
NAME    form_driver – command processor for the forms subsystem

SYNOPSIS  cc [ flag ... ] file ... -lform -lcurses [ library ... ]
#include <form.h>
int form_driver(FORM *form, int c);

MT-LEVEL Unsafe

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 to the end of 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_FLINE  Horizontal scroll forward a line.
REQ_SCR_BLINE  Horizontal scroll backward a line.
REQ_SCR_FHALF  Horizontal scroll forward half a line.
REQ_SCR_BHALF  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
SEE ALSO  
curses(3X), forms(3X)

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

modified 22 Jan 1993
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);

MT-LEVEL  
Unsafe

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.

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

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

modified 22 Jan 1993
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);
chttype field_fore(FIELD *field);
int set_field_back(FIELD *field, chtype attr);
chttype field_back(FIELD *field);
int set_field_pad(FIELD *field, int pad);
int field_pad(FIELD *field);

MT-LEVEL  
Unsafe

DESCRIPTION  
set_field_fore() sets the foreground attribute of field. The foreground attribute is the low-level curses 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 low-level curses 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.

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

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

3X-434  
modified 22 Jan 1993
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);

MT-LEVEL  Unsafe

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.

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

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

modified 22 Jan 1993
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 *fcol, int *nrow, int *nbuf);
int dynamic_field_info(FIELD *field, int *drows, int *dcols, int *max);

MT-LEVEL  Unsafe

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.

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

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

3X-436         modified 22 Jan 1993
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);

MT-LEVEL
Unsafe

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.

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

NOTES
The header <form.h> automatically includes the headers <eti.h> and <curses.h>.
NAME  form_field_new, new_field, dup_field, link_field, free_field, – create and destroy forms fields

SYNOPSIS  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);

MT-LEVEL  Unsafe

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 fol-
lowing:

E_OK  – The function returned successfully.
E_CONNECTED  – The field is already connected to a form.
E_SYSTEM_ERROR  – System error.
E_BAD_ARGUMENT  – An argument is incorrect.

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

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

3X-438  modified 22 Jan 1993
NAME
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, OPTIONSopts);
OPTIONS field_opts(FIELD *field);

MT-LEVEL
Unsafe

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.
SEE ALSO

curses(3X), forms(3X)

NOTES

The header `<form.h>` automatically includes the headers `<eti.h>` and `<curses.h>`.
## NAME
form_field_userptr, set_field_userptr, field_userptr – associate application data with forms

## SYNOPSIS
```
cc [ flag ... ] file ... -lform -lcurses [ library ... ]
#include <form.h>
int set_field_userptr(FIELD *field, char *ptr);
char *field_userptr(FIELD *field);
```

## MT-LEVEL
Unsafe

## DESCRIPTION
Every field has an associated user pointer that can be used to store pertinent data. set_field_userptr() sets the user pointer of 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.

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

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

---

modified 22 Jan 1993
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);

MT-LEVEL  Unsafe

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.

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

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

3X-442 modified 22 Jan 1993
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);

MT-LEVEL  Unsafe

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.

modified 22 Jan 1993
<table>
<thead>
<tr>
<th>SEE ALSO</th>
<th>curses(3X), forms(3X)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOTES</td>
<td>The header <code>&lt;form.h&gt;</code> automatically includes the headers <code>&lt;eti.h&gt;</code> and <code>&lt;curses.h&gt;</code>.</td>
</tr>
</tbody>
</table>

modified 22 Jan 1993
### 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
```c
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);
```

### MT-LEVEL
Unsafe

### 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.
SEE ALSO  curses(3X), forms(3X)

NOTES  The header <form.h> automatically includes the headers <eti.h> and <curses.h>.
<table>
<thead>
<tr>
<th>NAME</th>
<th>form_new, new_form, free_form – create and destroy forms</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYNOPSIS</td>
<td>cc [ flag ... ] file ... -lform -lcurses [ library ... ]</td>
</tr>
<tr>
<td></td>
<td>FORM *new_form(FIELD **fields);</td>
</tr>
<tr>
<td></td>
<td>int free_form(FORM *form);</td>
</tr>
<tr>
<td>MT-LEVEL</td>
<td>Unsafe</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>new_form() creates a new form connected to the designated fields and returns a pointer to the form.</td>
</tr>
<tr>
<td></td>
<td>free_form() disconnects the form from its associated field pointer array and deallocates the space for the form.</td>
</tr>
<tr>
<td>RETURN VALUES</td>
<td>new_form() always returns NULL on error. free_form() returns one of the following:</td>
</tr>
<tr>
<td></td>
<td>E_OK – The function returned successfully.</td>
</tr>
<tr>
<td></td>
<td>E_BAD_ARGUMENT – An argument is incorrect.</td>
</tr>
<tr>
<td></td>
<td>E_POSTED – The form is posted.</td>
</tr>
<tr>
<td>SEE ALSO</td>
<td>curses(3X), forms(3X)</td>
</tr>
<tr>
<td>NOTES</td>
<td>The header &lt;form.h&gt; automatically includes the headers &lt;eti.h&gt; and &lt;curses.h&gt;.</td>
</tr>
</tbody>
</table>
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);

MT-LEVEL  
Unsafe

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.

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

NOTES  
The header <form.h> automatically includes the headers <eti.h> and <curses.h>.
NAME
form_opts, set_form_opts, form_opts_on, form_opts_off – forms option routines

SYNOPSIS
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);

MT-LEVEL
Unsafe

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.

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

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

modified 22 Jan 1993
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);

MT-LEVEL  Unsafe

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.

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

NOTES  The header <form.h> automatically includes the headers <eti.h> and <curses.h>.
NAME
form_post, post_form, unpost_form – write or erase forms from associated subwindows

SYNOPSIS
cc [ flag ...] file ... -lform -lcurses [ library ...]
#include <form.h>
int post_form(FORM *form);
int unpost_form(FORM *form);

MT-LEVEL
Unsafe

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.

SEE ALSO
curses(3X), forms(3X), panel_update(3X), panels(3X)

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

modified 22 Jan 1993
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);

MT-LEVEL  Unsafe

DESCRIPTION  Every form has an associated user pointer that can be used to store pertinent data.
set_form_userptr() sets the user pointer of 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.

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

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

modified 22 Jan 1993
NAME
form_win, set_form_win, set_form_sub, form_sub, scale_form – forms window and subwindow association routines

SYNOPSIS
c
[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);

MT-LEVEL
Unsafe

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.

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

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

modified 22 Jan 1993
NAME       forms – character based forms package
SYNOPSIS   #include <form.h>
MT-LEVEL    Unsafe
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_opt(3X)</td>
</tr>
<tr>
<td>field_opts_off</td>
<td>form_field_opt(3X)</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>------------------------------</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>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>
</tbody>
</table>

modified 22 Jan 1993
set_form_init form_hook(3X)
set_form_opts form_opts(3X)
set_form_page form_page(3X)
set_form_sub form_win(3X)
set_form_term form_hook(3X)
set_form_userptr form_userptr(3X)
set_form_win form_win(3X)
set_max_field form_field_buffer(3X)
set_new_page form_new_page(3X)
unpost_form form_post(3X)

RETURN VALUES
Routines that return a pointer always return NULL on error. Routines that return an integer return one of the following:

E_OK − The function returned successfully.
E_CONNECTED − The field is already connected to a form.
E_SYSTEM_ERROR − System error.
E_BAD_ARGUMENT − An argument is incorrect.
E_CURRENT − The field is the current field.
E_POSTED − The form is posted.
E_NOT_POSTED − The form is not posted.
E_INVALID_FIELD − The field contents are invalid.
E_NOT_CONNECTED − The field is not connected to a form.
E_NO_ROOM − The form does not fit in the subwindow.
E_BAD_STATE − The routine was called from an initialization or termination function.
E_REQUEST_DENIED − The form driver request failed.
E_UNKNOWN_COMMAND − An unknown request was passed to the form driver.

SEE ALSO curses(3X), and 3X pages whose names begin "form_" for detailed routine descriptions.

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

3X-456 modified 22 Jan 1993
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);

MT-LEVEL
MT-Safe

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.

<table>
<thead>
<tr>
<th>Mask</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP_X_INV</td>
<td>invalid operation exception</td>
</tr>
<tr>
<td>FP_X_OFL</td>
<td>overflow exception</td>
</tr>
<tr>
<td>FP_X_UFL</td>
<td>underflow exception</td>
</tr>
<tr>
<td>FP_X_DZ</td>
<td>divide-by-zero exception</td>
</tr>
<tr>
<td>FP_X_IMP</td>
<td>imprecise (loss of precision)</td>
</tr>
</tbody>
</table>

The following floating-point rounding modes are passed to fpsetround() and returned by fpgetround().

<table>
<thead>
<tr>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP_RN</td>
<td>round to nearest representative number</td>
</tr>
<tr>
<td>FP_RP</td>
<td>round to plus infinity</td>
</tr>
<tr>
<td>FP_RM</td>
<td>round to minus infinity</td>
</tr>
<tr>
<td>FP_RZ</td>
<td>round to zero (truncate)</td>
</tr>
</tbody>
</table>

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.

modified 2 May 1994
fpsetsticky() sets (clears) the exception sticky flags and returns the previous setting.

SEE ALSO isnan(3C)

NOTES fpsetsticky() modifies all sticky flags. fpsetmask() changes all mask bits. fpsetmask() clears the sticky bit corresponding to any exception being enabled.
C requires truncation (round to zero) for floating point to integral conversions. The current rounding mode has no effect on these conversions.
One must clear the sticky bit to recover from the trap and to proceed. If the sticky bit is not cleared before the next trap occurs, a wrong exception type may be signaled.
NAME  fread, fwrite – buffered binary input/output

SYNOPSIS  
#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);

MT-LEVEL  MT-Safe

DESCRIPTION  fread() 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. fread() stops reading bytes if an end-of-file or error condition is encountered while reading stream, or if nitems items have been read. fread() increments the data pointer in stream to point to the byte following the last byte read if there is one. fread() does not change the contents of stream. fread() returns the number of items read.

fwrite() 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. fwrite() stops writing when it has written nitems items of data or if an error condition is encountered on stream. fwrite() does not change the contents of the array pointed to by ptr. fwrite() increments the data-pointer in stream by the number of bytes written. fwrite() 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.

The ferror() or feof() routines must be used to distinguish between an error condition and end-of-file condition.

RETURN VALUES  If an error occurs, fread() and fwrite() return 0 and set the error indicator for stream.

SEE ALSO  read(2), write(2), fclose(3S), fopen(3S), getc(3S), gets(3S), printf(3S), putc(3S), puts(3S), scanf(3S), stdio(3S)
NAME frexp, ldexp, logb, modf, modff, nextafter, scalb – manipulate parts of floating-point numbers

SYNOPSIS cc [ flag ... ] file ... -lm [ library ... ]
#include <math.h>
double frexp(double value, int *eptr);
double ldexp(double value, int exp);
double logb(double value);
double modf(double value, double *iptr);
float modff(float value, float *iptr);
double nextafter(double value1, double value2);
double scalb(double value, double exp);

MT-LEVEL MT-Safe

DESCRIPTION Every non-zero number can be written uniquely as \( x \times 2^n \), where the “mantissa” (fraction) \( x \) is in the range \( 0.5 \leq |x| < 1.0 \), and the “exponent” \( n \) is an integer. frexp() returns the mantissa of a double value, and stores the exponent indirectly in the location pointed to by eptr. If value is zero, both results returned by frexp() are zero.

ldexp() and scalb() return the quantity \( value \times 2^{exp} \). The only difference between the two is that scalb() of a signaling NaN will result in the invalid operation exception being raised.

logb() returns the unbiased exponent of its floating-point argument as a double-precision floating-point value.

modf() and modff() (single-precision version) return the signed fractional part of value and store the integral part indirectly in the location pointed to by iptr.

nextafter() returns the next representable double-precision floating-point value following value1 in the direction of value2. Thus, if value2 is less than value1, nextafter() returns the largest representable floating-point number less than value1.

RETURN VALUES If ldexp() would cause overflow, \pm \text{HUGE} \text{ (defined in <math.h>)} is returned (according to the sign of value), and errno is set to ERANGE. If ldexp() would cause underflow, zero is returned and errno is set to ERANGE. If the input value to ldexp() is NaN or infinity, that input is returned and errno is set to EDOM. The same error conditions apply to scalb() except that a signaling NaN as input will result in the raising of the invalid operation exception.
logb() of NaN returns that NaN, logb() of infinity returns positive infinity, and logb() of zero returns negative infinity and results in the raising of the divide by zero exception. In each of these conditions errno is set to EDOM.

If input value1 to nextafter() is positive or negative infinity, that input is returned and errno is set to EDOM. The overflow and inexact exceptions are signalled when input value1 is finite, but nextafter(value1, value2) is not. The underflow and inexact exceptions are signalled when nextafter(value1, value2) lies strictly between $+2^{-1022}$ and $-2^{-1022}$. In both cases errno is set to ERANGE.
NAME  fseek, rewind, ftell – reposition a file pointer in a stream

SYNOPSIS  #include <stdio.h>
int fseek(FILE *stream, long offset, int ptrname);
void rewind(FILE *stream);
long ftell(FILE *stream);

MT-LEVEL  MT-Safe

DESCRIPTION  fseek( ) sets the position of the next input or output operation on the stream (see intro(3)).
The new position is at the signed distance offset bytes from the beginning, from the current position, or from the end of the file, according to a ptrname value of SEEK_SET, SEEK_CUR, or SEEK_END (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.

fseek() allows the file position indicator to be set beyond the end of the existing data in the file. If data is later written at this point, subsequent reads of data in the gap will return zero until data is actually written into the gap. fseek(), by itself, does not extend the size of the file.

rewind(stream) is equivalent to:
   (void) fseek (stream, 0L, SEEK_SET);
except that rewind() also clears the error indicator on stream.

fseek() and rewind() clear the EOF indicator and undo any effects of ungetc() on stream. After fseek() or rewind(), the next operation on a file opened for update may be either input or output.

If stream is writable and buffered data has not been written to the underlying file, fseek() and rewind() cause the unwritten data to be written to the file.

ftell() returns the offset of the current byte relative to the beginning of the file associated with the named stream.

RETURN VALUES  fseek() returns −1 for improper seeks, otherwise zero. An improper seek can be, for example, an fseek() done on a file that has not been opened via fopen(); in particular, fseek() may not be used on a terminal or on a file opened via popen(). After a stream is closed, no further operations are defined on that stream.

SEE ALSO  lseek(2), write(2), intro(3), fopen(3S), popen(3S), stdio(3S), ungetc(3S)

3S-462
modified 22 Jan 1993
NOTES

Although on the UNIX system an offset returned by `fseek()` is measured in bytes, and it is permissible to seek to positions relative to that offset, portability to non-UNIX systems requires that an offset be used by `fseek()` directly. Arithmetic may not meaningfully be performed on such an offset, which is not necessarily measured in bytes.
NAME    fsetpos, fgetpos — reposition a file pointer in a stream

SYNOPSIS #include <stdio.h>
    int fsetpos(FILE *stream, const fpos_t *pos);
    int fgetpos(FILE *stream, fpos_t *pos);

MT-LEVEL MT-Safe

DESCRIPTION fsetpos() sets the position of the next input or output operation on the stream according to the value of the object pointed to by pos. The object pointed to by pos must be a value returned by an earlier call to fgetpos() on the same stream.

fsetpos() clears the end-of-file indicator for the stream and undoes any effects of the ungetc() function on the same stream. After fsetpos(), the next operation on a file opened for update may be either input or output.

fgetpos() stores the current value of the file position indicator for stream in the object pointed to by pos. The value stored contains information usable by fsetpos() for repositioning the stream to its position at the time of the call to fgetpos().

RETURN VALUES If successful, both fsetpos() and fgetpos() return zero. Otherwise, they both return nonzero.

SEE ALSO lseek(2), fseek(3S), ungetc(3S)
**NAME**
fsync – synchronize a file’s in-memory state with that on the physical medium

**SYNOPSIS**
```
#include <unistd.h>
int fsync(int fildes);
```

**MT-LEVEL**
Async-Signal-Safe

**DESCRIPTION**
fsync() 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. fsync() is different from sync(), which schedules disk I/O for all files but returns before the I/O completes. fsync() forces all outstanding data operations to synchronized file integrity completion (see fcntl(5) definition of O_SYNC.).

fsync() should be used by programs that require that a file be in a known state. For example, a program that contains a simple transaction facility might use fsync() to ensure that all changes to a file or files caused by a given transaction were recorded on a storage medium.

**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**
fsync() fails if one or more of the following are true:
- **EBADF**
  fildes is not a valid file descriptor open for writing.
- **EINTR**
  A signal was caught during execution of the fsync() function.
- **EIO**
  An I/O error occurred while reading from or writing to the file system.
- **ENOLINK**
  fildes is on a remote machine and the link to that machine is no longer active.

**SEE ALSO**
sync(2), fdatasync(3R), fcntl(5)

**NOTES**
The way the data reach the physical medium depends on both implementation and hardware. fsync() returns when the device driver tells it that the write has taken place.
NAME  
ftime – get date and time

SYNOPSIS  
#include <sys/types.h>
#include <sys/timeb.h>
int ftime(struct timeb *tp);

DESCRIPTION  
The ftime() entry fills in a structure pointed to by its argument. The structure is defined in <sys/timeb.h> and contains the following members:

```c
    time_t time;
    unsigned short millitm;
    short timezone;
    short dstflag;
```

The structure contains the time since the epoch in seconds, up to 1000 milliseconds of more-precise interval, the local time zone, and a flag that, if nonzero, 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.

SEE ALSO  
date(1), time(2), gettimeofday(3C), ctime(3C), timezone(4)
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);

MT-LEVEL

See the NOTES section of this page.

DESCRIPTION

ftw() 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.

SEE ALSO

stat(2), longjmp(3C), malloc(3C)

**NOTES**

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

ftw() uses malloc(3C) to allocate dynamic storage during its operation. If ftw() is forcibly terminated, such as by longjmp(3C) being executed by fn or an interrupt routine, ftw() will not have a chance to free that storage, so it will remain permanently allocated. A safe way to handle interrupts is to store the fact that an interrupt has occurred, and arrange to have fn return a nonzero value at its next invocation.

ftw() is safe in multi-thread applications. nftw() is safe in multi-thread applications when the FTW_CHDIR flag is not set.
NAME  
getacinfo, getacdir, getacflg, getacmin, getacna, setac, endac – get audit control file information

SYNOPSIS  
cc [flag ...] file ... -lbsm -lsocket -lnsl -lintl [ library ... ]
#include <bsm/libbsm.h>
int getacdir( char *dir, int len);
int getacmin( int *min_val);
int getacflg( char *auditstring, int len);
int getacna( char *auditstring, int len);
void setac( void);
void endac( void);

MT-LEVEL  
Safe.

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

DESCRIPTION  
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.

getacflg() 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(), getacflg(), getacna() and getacmin() return:

0 on success.

−2 on failure and set errno to indicate the error.

getacmin() and getacflg() return:

1 on EOF.

getacdir() returns:

−1 on EOF.

2 if the directory search had to start from the beginning because one of the other
functions was called between calls to getacdir().

These functions return:

−3 if the directory entry format in the audit_control file is incorrect.

getacdir(), getacflg() and getacna() return:

−3 if the input buffer is too short to accommodate the record.

SEE ALSO

audit_warn(1M), bsmconv(1M), inetd(1M), audit_control(4)
NAME  getauclassnam, getauclassent, setauclass, endauclass, getauclassnam_r, getauclassent_r — get audit_class entry

SYNOPSIS  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);

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

MT-LEVEL  MT-Safe with exceptions.
  All of the functions described in this man-page are MT-Safe except getauclassent() and getauclassnam(). The two functions, getauclassent_r() and getauclassnam_r() have the same functionality as the unsafe functions, but have a slightly different function call interface in order to make them MT-Safe.

DESCRIPTION  getauclassent() and getauclassnam() each return an audit_class entry.
getauclassent() 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 system may then close any open audit_class file, deallocate storage, and so forth.
getauclassent_r() and getauclassnam_r() both return a pointer to an audit_class entry as do their similarly named counterparts. They each take an additional argument, a pointer to pre-allocated space for an au_class_ent_t, which is returned if the call is successful. To assure there is enough space for the information returned, the applications programmer should be sure to allocate AU_CLASS_NAME_MAX and AU_CLASS_DESC_MAX bytes for the ac_name and ac_desc elements of the au_class_ent_t data structure.
The internal representation of an audit_user entry is an au_class_ent structure defined in <bsm/libbsm.h> with the following members:

modified 23 Feb 1993
### RETURN VALUES

`getauclassnam()` and `getauclassnam_r()` return a pointer to a `struct au_class_ent` if they successfully locate the requested entry; otherwise they return `NULL`.

`getauclassent()` and `getauclassent_r()` return a pointer to a `struct au_class_ent` if they successfully enumerate an entry; otherwise they return `NULL`, indicating the end of the enumeration.

### FILES

`/etc/security/audit_class`  
Maps audit class numbers to audit class names

### SEE ALSO

`bsmconv(1M), audit_class(4), audit_event(4)`

### NOTES

All information is contained in a static area, so it must be copied if it is to be saved.
NAME  getauditflags, getauditflagsbin, getauditflagschar – convert audit flag specifications

SYNOPSIS  cc [ flag ... ] file ... -lbsm -lsocket -lnsl -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);

MT-LEVEL  MT-Safe.

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

DESCRIPTION  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.

SEE ALSO  bsmconv(1M), audit.log(4), audit_control(4)

BUGS  This is not a very extensible interface.
NAME
getauevent, getauevnam, getauevnum, getauevnonam, setauevent, endauevent,
getauevent_r, getauevnam_r, getauevnum_r – get audit_user entry

SYNOPSIS
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, void);
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);

MT-LEVEL
MT-Safe with exceptions.

The functions getauevent(), getauevnam(), and getauevnum() are not MT-Safe; but, 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.

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

DESCRIPTION
getauevent(), getauevnam(), getauevnum(), getauevent(), getauevnam(), 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 name.
getauevnum() and getauevnum_r() search for an audit_event entry with a given event
number number.
getauevnonum() searches for an audit_event entry with a given event name name and
returns the corresponding event number.
setauevent() “rewinds” to the beginning of the enumeration of audit_event entries. Calls
to getauevnam(), getauevnum(), getauevnonum(), getauevnum_r(), getauevnam_r(), or
getauevnum_r() may leave the enumeration in an indeterminate state; so, setauevent() should be called before the first getauevent() or getauevent_r().
endauevent() may be called to indicate that audit_event processing is complete; the system may then close any open audit_event file, deallocate storage, and so forth.

The three functions getauevent_r(), getauevnam_r(), and getauevnum_r(), each take an argument e which is a pointer to an au_event_ent_t. This pointer is returned on a successful function call. To assure there is enough space for the information returned, the applications programmer should be sure to allocate AU_EVENT_NAME_MAX and AU_EVENT_DESC_MAX bytes for the ae_name and ac_desc elements of the au_event_ent_t data structure.

The internal representation of an audit_event entry is an struct au_event_ent structure defined in <bsm/libbsm.h> with the following members:

```
    au_event_t    ae_number;
    char          *ae_name;
    char          *ae_desc;
    au_class_t    ae_class;
```

RETURN VALUES
getauevent(), getauevnam() and getauevnum() return a pointer to a struct au_event_ent if it successfully locates the requested entry; otherwise it returns NULL.

getauevnonam() returns an event number of type au_event_t if it successfully enumerates an entry; otherwise it returns NULL, indicating it could not find the requested event name.

FILES
/etc/security/audit_event
Maps audit event numbers to audit event names
/etc/passwd
Stores user-id to username mappings

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

NOTES
All information for the functions getauevent(), getauevnam(), and getauevnum() is contained in a static area, so it must be copied if it is to be saved.
NAME
getauusernam, getauuserent, setauuser, endauuser – get audit_user entry

SYNOPSIS
c

#include <sys/param.h>
#include <bsm/libbsm.h>
struct au_user_ent *getauusernam(const char *name);
struct au_user_ent *getauuserent(void);
void setauuser(void);
void endauuser(void);
struct au_user_ent *getauusernam_r(au_user_ent *u, const char *name);
struct au_user_ent *getauuserent_r(au_user_ent *u);

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

MT-LEVEL
MT-Safe with exceptions.
The functions getauusernam() and getauuserent() are not MT-safe. However, the func-
tions getauusernam_r() and getauuserent_r() provide the same functionality with MT-
Safe interfaces.

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 indeter-
minate 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;
au_mask_t au_always;
au_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.

FILES

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

SEE ALSO

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

NOTES

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

getc, getc_unlocked, getchar, getchar_unlocked, fgetc, getw – get character or word from a stream

**SYNOPSIS**

```
#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);
```

**MT-LEVEL**

See the NOTES section of this page.

**DESCRIPTION**

getc() 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. getchar() is defined as getc(stdin). getc() and getchar() are macros.

getc_unlocked() and getchar_unlocked() are respectively variants of getc() and getchar() that do not lock the stream. It is the caller’s responsibility to acquire the stream lock before calling these functions and releasing the lock afterwards; see flockfile(3S) and stdio(3S).

fgetc() behaves like getc(), but is a function rather than a macro. fgetc() runs more slowly than getc(), but it takes less space per invocation and its name can be passed as an argument to a function.

gew() returns the next word (that is, integer) from the named input stream. getw() increments the associated file pointer, if defined, to point to the next word. The size of a word is the size of an integer and varies from machine to machine. getw() assumes no special alignment in the file.

**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 is a valid integer, ferror() should be used to detect getw() errors.

**SEE ALSO**

intro(3), fclose(3S), ferror(3S), flockfile(3S), fopen(3S), fread(3S), gets(3S), putc(3S), scanf(3S), stdio(3S), ungetc(3S)

**NOTES**

If the integer value returned by getc(), getchar(), or fgetc() is stored into a character variable and then compared against the integer constant EOF, the comparison may never succeed, because sign-extension of a character on widening to integer is implementation dependent.

3S-478

modified 22 Jan 1993
The macro version of `getc()` evaluates a `stream` argument more than once and may treat side effects incorrectly. In particular, `getc(*f++)` does not work sensibly. Use `fgetc()` instead.

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

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

`fgetc()`, `getc()`, `getchar()`, `getw()`, and `ungetc()` are MT-Safe in multi-thread applications. `getc_unlocked()` and `getchar_unlocked()` are unsafe in multi-thread applications.
NAME
getcwd – get pathname of current working directory

SYNOPSIS
#include <unistd.h>
extern char ∗getcwd(char ∗buf, size_t size);

MT-LEVEL
Safe

DESCRIPTION
getcwd() returns a pointer to the current directory pathname. The value of 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
getcwd() returns NULL with errno set if size is not large enough, or if an error occurs in a
lower-level function.

ERRORS
getcwd() will fail if one or more of the following are true:
EACCES A parent directory cannot be read to get its name.
EINVAL size is equal to 0.
ERANGE size 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.

#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);
}

SEE ALSO chdir(2), malloc(3C)

NOTES
Using chdir(2) in conjunction with getcwd can give unpredictable results.
NAME  getdate – convert user format date and time

SYNOPSIS  
#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

modified 13 May 1994
<table>
<thead>
<tr>
<th>Conversion</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>%U</code></td>
<td>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</td>
</tr>
<tr>
<td><code>%w</code></td>
<td>weekday as a decimal number ([0,6]), with 0 representing Sunday</td>
</tr>
<tr>
<td><code>%W</code></td>
<td>week number of the year as a decimal number ([0,53]), with Monday as the first day of the week; leading zero is permitted but not required</td>
</tr>
<tr>
<td><code>%x</code></td>
<td>locale’s appropriate date representation</td>
</tr>
<tr>
<td><code>%X</code></td>
<td>locale’s appropriate time representation</td>
</tr>
<tr>
<td><code>%y</code></td>
<td>year within the century ([0,99]); leading zero is permitted but not required</td>
</tr>
<tr>
<td><code>%Y</code></td>
<td>year, including the century (for example, 1993)</td>
</tr>
<tr>
<td><code>%Z</code></td>
<td>time zone name or no characters if no time zone exists</td>
</tr>
</tbody>
</table>

**Modified Conversion Specifications**

Some conversion specifications can be modified by the `E` and `O` modifier characters to indicate that an alternative format or specification should be used rather than the one normally used by the unmodified specification. If the alternative format or specification does not exist in the current locale, the behaviour will be as if the unmodified conversion specification were used.

- `%Ec` locale’s alternative appropriate date and time representation
- `%EC` name of the base year (period) in the locale’s alternative representation
- `%Ex` locale’s alternative date representation
- `%EX` locale’s alternative time representation
- `%Ey` offset from `%EC` (year only) in the locale’s alternative representation
- `%EY` full alternative year representation
- `%Od` day of the month using the locale’s alternative numeric symbols; leading zeros are permitted but not required
- `%Oe` same as `%Od`
- `%OH` hour (24-hour clock) using the locale’s alternative numeric symbols
- `%OI` hour (12-hour clock) using the locale’s alternative numeric symbols
- `%Om` month using the locale’s alternative numeric symbols
- `%OM` minutes using the locale’s alternative numeric symbols
- `%OS` seconds using the locale’s alternative numeric symbols
- `%OU` week number of the year (Sunday as the first day of the week) using the locale’s alternative numeric symbols
- `%Ow` number of the weekday (Sunday=0) using the locale’s alternative numeric symbols
- `%OW` week number of the year (Monday as the first day of the week) using the locale’s alternative numeric symbols
- `%Oy` year (offset from `%C`) in the locale’s alternative representation and using the locale’s alternative numeric symbols

**Internal Format Conversion**

The following rules are applied for converting the input specification into the internal format:

- If only the weekday is given, today is assumed if the given day is equal to the current day and next week if it is less.
If only the month is given, the current month is assumed if the given month is equal to the current month and next year if it is less and no year is given. (The first day of month is assumed if no day is given.)

If no hour, minute, and second are given, the current hour, minute, and second are assumed.

If no date is given, today is assumed if the given hour is greater than the current hour and tomorrow is assumed if it is less.

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 speci®cations 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 speci®cation 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>

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

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

NOTES
Subsequent calls to getdate() alter the contents of getdate_err.

Dates before 1970 and after 2037 are illegal.

The range of values for %S is [00,61] rather than [00,59] to allow for the occasional leap second and even more occasional double leap second.

getdate() makes explicit use of macros described in ctype(3C).
NAME  
getdtablesize – get file descriptor table size

SYNOPSIS  
#include <unistd.h>

int getdtablesize(void);

DESCRIPTION  
Each process has a file descriptor table which is guaranteed to have at least 20 slots. The entries in the descriptor table are numbered with small integers starting at 0. The getdtablesize() function returns the current maximum size of this table by calling the getrlimit() function.

SEE ALSO  
close(2), getrlimit(2), open(2), pipe(2), select(3C) sysconf(3C)
NAME   getenv – return value for environment name

SYNOPSIS   #include <stdlib.h>
            char *getenv(const char *name);

MT-LEVEL   Safe

DESCRIPTION    getenv() searches the environment list (see environ(5)) for a string of the form
                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.

SEE ALSO       exec(2), putenv(3C), environ(5)

NOTES          getenv() can be safely called from a multi-thread program. However, care must still be taken when using getenv() and putenv(3C) in a multi-thread program. These routines examine and modify the environment list. This list is shared by all threads in a program. The system prevents the list from being accessed simultaneously by two different threads. However, it does not prevent two threads from successively accessing the environment list using getenv() or putenv(3C).
NAME
getfauditflags – generates the process audit state

SYNOPSIS
cc [ flag ...] file ... -l bsm -l socket -lnsl -lintl [ library ... ]
#include <sys/param.h>
#include <bsm/libbsm.h>
int getfauditflags(au_mask_t *usremasks, au_mask_t *usrdmasks, au_mask_t *lastmasks);

MT-LEVEL
MT-Safe.

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

DESCRIPTION
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.

SEE ALSO
bsmconv(1M), getacinfo(3), getauditflags(3), getauusernam(3), audit.log(4), audit_control(4), audit_user(4)
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 *result,
                        char *buffer, int buflen);
struct group *getgrent(void);
struct group *getgrent_r(struct group *result, char *buffer, int buflen);
struct group *getgrgid(gid_t gid);
struct group *getgrgid_r(gid_t gid, struct group *result, char *buffer, int buflen);
void setgrent(void);
void endgrent(void);
struct group *fgetgrent(FILE *f);
struct group *fgetgrent_r(FILE *f, struct group *result, char *buffer, int buflen);

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);

MT-LEVEL
See the subsection “Reentrant Interfaces” in the 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
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.

endgrent() may be called to indicate that the caller expects to do no further group entry
retrieval operations; the system may then close the group file, deallocate resources it was
using, and so forth. It is still allowed, but possibly less efficient, for the process to call
more group functions after calling endgrent().

modified 22 Aug 1995

3C-489
The functions `getgrnam()`, `getgrgid()`, `getgrent()`, and `fgetgrent()` use static storage that is re-used in each call, making them unsafe for multithreaded applications. The parallel functions `getgrnam_r()`, `getgrgid_r()`, `getgrent_r()`, and `fgetgrent_r()` provide reentrant interfaces for these operations.

Each reentrant interface performs the same operation as its non-reentrant counterpart, named by removing the "_r" suffix. The reentrant interfaces, however, use buffers supplied by the caller to store returned results, and are safe for use in both single-threaded and multithreaded applications.

Each reentrant interface takes the same parameters as its non-reentrant counterpart, as well as the following additional parameters. The parameter `result` (or `grp` for the POSIX versions) must be a pointer to a `struct group` structure allocated by the caller. On successful completion, the function returns the group entry in this structure. The parameter `buffer` is a pointer to a buffer supplied by the caller, used as storage space for the group data. All of the pointers within the returned `struct group` point to data stored within this buffer; see RETURN VALUES. The buffer must be large enough to hold all the data associated with the group entry. The parameter `buflen` (or `bufsize` for the POSIX versions) should give the size in bytes of `buffer`. The POSIX versions place a pointer to the modified `grp` structure in the `result` parameter, instead of returning a pointer to this structure.

For enumeration in multithreaded applications, the position within the enumeration is a process-wide property shared by all threads. `setgrent()` may be used in a multithreaded application but resets the enumeration position for all threads. If multiple threads interleave calls to `getgrent_r()`, the threads will enumerate disjoint subsets of the group database.

Like their non-reentrant counterparts, `getgrnam_r()` and `getgrgid_r()` leave the enumeration position in an indeterminate state.

**RETURN VALUES**

Group entries are represented by the `struct group` structure defined in `<grp.h>`:

```c
struct group {
    char *gr_name; /* the name of the group */
    char *gr_passwd; /* the encrypted group password */
    gid_t gr_gid; /* the numerical group ID */
    char **gr_mem; /* vector of pointers to member names */
};
```

The functions `getgrnam()`, `getgrnam_r()`, `getgrgid()`, and `getgrgid_r()` each return a pointer to a `struct group` if they successfully locate the requested entry; otherwise they return NULL. The POSIX functions `getgrnam_r()` and `getgrgid_r()` return zero upon success, or the error number in case of failure.
The functions `getgrent()`, `getgrent_r()`, `fgetgrent()`, and `fgetgrent_r()` each return a pointer to a `struct group` if they successfully enumerate an entry; otherwise they return NULL, indicating the end of the enumeration.

The functions `getgrnam()`, `getgrgid()`, `getgrent()`, and `fgetgrent()` use static storage, so returned data must be copied before a subsequent call to any of these functions if the data is to be saved.

When the pointer returned by the reentrant functions `getgrnam_r()`, `getgrgid_r()`, `getgrent_r()`, and `fgetgrent_r()` is non-NULL, it is always equal to the `result` pointer that was supplied by the caller.

**ERRORS**
The reentrant functions `getgrnam_r()`, `getgrgid_r()`, `getgrent_r()`, and `fgetgrent_r()` will return NULL and set `errno` to `ERANGE` (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

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

**NOTES**
Programs that use the interfaces described in this manual page cannot be linked statically since the implementations of these functions employ dynamic loading and linking of shared objects at run time.

When compiling multithreaded applications, see `Intro(3)`, `Notes On Multithread Applications`, for information about the use of the `_REENTRANT` flag.

Use of the enumeration interfaces `getgrent()` and `getgrent_r()` is discouraged; enumeration is supported for the group file, NIS, and NIS+, but in general is not efficient and may not be supported for all database sources. The semantics of enumeration are discussed further in `nsswitch.conf(4)`.

Previous releases allowed the use of “+” and “-” entries in `/etc/group` to selectively include and exclude entries from NIS. The primary usage of these “+/-” entries is superseded by the name service switch, so the “+/-” form may not be supported in future releases.

If required, the “+/-” functionality can still be obtained for NIS by specifying `compat` as the source for `group`.

If the “+/-” functionality is required in conjunction with NIS+, specify both `compat` as the source for `group` and `nisplus` as the source for the pseudo-database `group_compat`. See `group(4)`, and `nsswitch.conf(4)` for details.

The reentrant interfaces `getgrnam_r()`, `getgrgid_r()`, `getgrent_r()`, and `fgetgrent_r()` are as specified in POSIX 1003.1c Draft #10.

modified 22 Aug 1995
NAME
gethostbyname, gethostbyname_r, gethostbyaddr, gethostbyaddr_r, gethostent, gethostent_r, sethostent, endhostent — get network host entry

SYNOPSIS
cc [flag ...] file ... -lnsl [ library ... ]
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>

struct hostent *gethostbyname(const char *name);
struct hostent *gethostbyname_r(const char *name, struct hostent *result, char *buffer, int buflen, int *h_errnop);
struct hostent *gethostbyaddr(const char *addr, int len, int type);
struct hostent *gethostbyaddr_r(const char *addr, int length, int type, struct hostent *result, char *buffer, int buflen, int *h_errnop);
struct hostent *gethostent(void);
struct hostent *gethostent_r(struct hostent *result, char *buffer, int buflen, int *h_errnop);
int sethostent(int stayopen);
int endhostent(void);

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

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().

3N-492 modified 22 May 1994
Successive calls to `gethostbyname()` return either successive entries or NULL, indicating the end of the enumeration.

`endhostent()` may be called to indicate that the caller expects to do no further host entry retrieval operations; the system may then deallocate resources it was using. It is still allowed, but possibly less efficient, for the process to call more host retrieval functions after calling `endhostent()`.

**Reentrant Interfaces**

The functions `gethostbyname()`, `gethostbyaddr()`, and `gethostent()` use static storage that is re-used in each call, making these functions unsafe for use in multithreaded applications.

The functions:

- `gethostbyname_r()`,
- `gethostbyaddr_r()`,

and

- `gethostent_r()`

provide reentrant interfaces for these operations.

Each reentrant interface performs the same operation as its non-reentrant counterpart, named by removing the "_r" suffix. The reentrant interfaces, however, use buffers supplied by the caller to store returned results, and are safe for use in both single-threaded and multithreaded applications.

Each reentrant interface takes the same parameters as its non-reentrant counterpart, as well as the following additional parameters. The parameter `result` must be a pointer to a `struct hostent` structure allocated by the caller. On successful completion, the function returns the host entry in this structure. The parameter `buffer` must be a pointer to a buffer supplied by the caller. This buffer is used as storage space for the host data. All of the pointers within the returned `struct hostent` `result` point to data stored within this buffer (see RETURN VALUES). The buffer must be large enough to hold all of the data associated with the host entry. The parameter `buflen` should give the size in bytes of the buffer indicated by `buffer`. The parameter `h_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>`:

```
struct hostent {
    char *h_name; /* canonical name of host */
    char **h_aliases; /* alias list */
    int h_addrtype; /* host address type */
    int h_length; /* length of address */
    char **h_addr_list; /* list of addresses */
};
```

See the EXAMPLES section below on how to retrieve a “.” separated Internet IP address string from the `h_addr_list` field of `struct hostent`.

The functions `gethostbyname()`, `gethostbyname_r()`, `gethostbyaddr()`, and `gethostbyaddr_r()` each return a pointer to a `struct hostent` if they successfully locate the requested entry; otherwise they return `NULL`.

The functions `gethostent()` and `gethostent_r()` each return a pointer to a `struct hostent` if they successfully enumerate an entry; otherwise they return `NULL`, indicating the end of the enumeration.

The functions `gethostbyname()`, `gethostbyaddr()`, and `gethostent()` use static storage, so returned data must be copied before a subsequent call to any of these functions if the data is to be saved.

When the pointer returned by the reentrant functions `gethostbyname_r()`, `gethostbyaddr_r()`, and `gethostent_r()` is not `NULL`, it is always equal to the `result` pointer that was supplied by the caller.

The functions `sethostent()` and `endhostent()` return `0` on success.

ERRORS

The reentrant functions `gethostbyname_r()`, `gethostbyaddr_r()` and `gethostent_r()` will return `NULL` and set `errno` to `ERANGE` if the length of the buffer supplied by caller is not large enough to store the result. See `intro(2)` for the proper usage and interpretation of `errno` in multithreaded applications.

On failures, the non-reentrant functions `gethostbyname()` and `gethostbyaddr()` set a global integer `h_errno` to indicate one of these error codes (defined in `<netdb.h>`):

- `HOST_NOT_FOUND`
- `TRY_AGAIN`
- `NO_RECOVERY`
- `NO_DATA`
- `NO_ADDRESS`

The reentrant functions `gethostbyname_r()` and `gethostbyaddr_r()` set the integer pointed to by `h_errno` 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

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

The reentrant interfaces `gethostbyname_r()`, `gethostbyaddr_r()`, and `gethostent_r()` are included in this release on an uncommitted basis only, and are subject to change or removal in future minor releases.

NOTES

Programs that use the interfaces described in this manual page cannot be linked statically since the implementations of these functions employ dynamic loading and linking of shared objects at run time.

In order to ensure that they all return consistent results, `gethostbyname()`, `gethostbyname_r()`, and `netdir_getbyname()` are implemented in terms of the same internal library function. This function obtains the system-wide source lookup policy based on the `inet` family entries in `netconfig(4)` and the `hosts` entry in `nsswitch.conf(4)`. Similarly, `gethostbyaddr()`, `gethostbyaddr_r()`, and `netdir_getbyaddr()` are implemented in terms of the same internal library function. If the `inet` family entries in `netconfig(4)` have a "-" in the last column for nametoaddr libraries, then the entry for `hosts` in `nsswitch.conf` will be used; otherwise the nametoaddr libraries in that column will be used, and `nsswitch.conf` will not be consulted.

There is no analogue of `gethostent()` and `gethostent_r()` in the netdir functions, so these enumeration functions go straight to the `hosts` entry in `nsswitch.conf`. Thus enumeration may return results from a different source than that used by `gethostbyname()`, `gethostbyname_r()`, `gethostbyaddr()`, and `gethostbyaddr_r()`.

All the functions that return a `struct hostent` must always return the canonical name in the `h_name` field. This name, by definition, is the well-known and official hostname shared between all aliases and all addresses. The underlying source that satisfies the request determines the mapping of the input name or address into the set of names and addresses in `hostent`. Different sources might do that in different ways. If there is more than one alias and more than one address in `hostent`, no pairing is implied between them.

The system will strive to put the addresses on the same subnet as that of the caller first. When compiling multithreaded applications, see `Intro(3)`, `Notes On Multithread Applications`, for information about the use of the `_REENTRANT` flag.

Use of the enumeration interfaces `gethostent()` and `gethostent_r()` is discouraged; enumeration may not be supported for all database sources. The semantics of enumeration are discussed further in `nsswitch.conf(4)`.

The current implementations of these functions only return or accept addresses for the Internet address family (type `AF_INET`).

The form for an address of type `AF_INET` is a `struct in_addr` defined in `<netinet/in.h>`. The functions described in `inet(3N)`, and illustrated in the `EXAMPLES` section above, are helpful in constructing and manipulating addresses in this form.
<table>
<thead>
<tr>
<th>NAME</th>
<th>gethostid – get unique identifier of current host</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYNOPSIS</td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>#include &lt;unistd.h&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>long gethostid(void);</code></td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td><code>gethostid()</code> 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.</td>
</tr>
<tr>
<td>SEE ALSO</td>
<td><code>hostid(1), sysinfo(2)</code></td>
</tr>
</tbody>
</table>
### NAME
gethostname, sethostname — get/set name of current host

### SYNOPSIS
```c
int gethostname(char *name, int namelen);
int sethostname(char *name, int namelen);
```

### DESCRIPTION
gethostname() returns the standard host name for the current processor, as previously set by sethostname. The parameter namelen specifies the size of the array pointed to by name. The returned name is null-terminated unless insufficient space is provided.

sethostname() sets the name of the host machine to be name, which has length namelen. This call is restricted to the privileged user and is normally used only when the system is bootstrapped.

### RETURN VALUES
If the call succeeds a value of 0 is returned. If the call fails, then a value of −1 is returned and an error code is placed in the global location errno.

### ERRORS
The following error may be returned by these calls:

- **EFAULT** — The name or namelen parameter gave an invalid address.
- **EPERM** — The caller was not the privileged user. Note: this error only applies to sethostname().

### SEE ALSO
uname(2), sysinfo(2), gethostid(3C)

### NOTES
Host names are limited to MAXHOSTNAMELEN characters, currently 256. (See the `<sys/param.h>` header.)
NAME  gethrtime, gethrvtime – get high resolution time

SYNOPSIS  #include <sys/time.h>

hrtime_t gethrtime(void);
hrtime_t gethrvtime(void);

MT-LEVEL  MT-Safe

DESCRIPTION  gethrtime() returns the current high-resolution real time. Time is expressed as nanoseconds since some arbitrary time in the past; it is not correlated in any way to the time of day, and thus is not subject to resetting, drifting, etc. via adjtime(2) or settimeofday(3C). The hi-res timer is ideally suited to performance measurement tasks, where cheap, accurate interval timing is required.

gethrvtme() returns the current high-resolution LWP virtual time, expressed as total nanoseconds of execution time.

gethrtime() and gethrvtime() 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):

```
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);
```

SEE ALSO  adjtime(2), gettimeofday(3C), settimeofday(3C)

NOTES  Although the units of hi-res time are always the same (nanoseconds), the actual resolution is hardware dependent. Hi-res time is guaranteed to be monotonic (it won’t go backward, it won’t periodically wrap) and linear (it won’t occasionally speed up or slow down for adjustment, like the time of day can), but not necessarily unique: two sufficiently proximate calls may return the same value.
NAME  getlogin, getlogin_r – get login name

SYNOPSIS  

```c
#include <stdlib.h>
char *getlogin(void);
char *getlogin_r(char *name, int namelen);
```

POSIX  

```c
int getlogin_r(char *name, size_t namesize);
```

DESCRIPTION  

`getlogin()` returns a pointer to the login name as found in `/var/adm/utmp`. It may be used in conjunction with `getpwnam()` to locate the correct password file entry when the same user id is shared by several login names.

If `getlogin()` is called within a process that is not attached to a terminal, it returns a null pointer. The correct procedure for determining the login name is to call `cuserid()`, or to call `getlogin()` and if it fails to call `getpwuid()`.

`getlogin_r()` 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 `LOGNAME_MAX` bytes in size (defined in `<limits.h>`). The POSIX version of `getlogin_r()` takes a `namesize` parameter of type `size_t`.

RETURN VALUES  

Returns a null pointer if the login name is not found.

ERRORS  

`getlogin_r()` will fail if the following is true:

- **ERANGE**  
  The size of the buffer is smaller than the result to be returned.

The POSIX `getlogin_r()` returns zero if successful, or the error number upon failure.

FILES  

`/var/adm/utmp`

SEE ALSO  

`cuserid(3S), getgrnam(3C), getpwnam(3C), utmp(4)`

NOTES  

When compiling multi-thread applications, the `-D_POSIX_PTHREAD_SEMANTICS [ library... ]` flag should only be used in multi-thread applications.

The return values point to static data whose content is overwritten by each call.

`getlogin()` is unsafe in multi-thread applications. `getlogin_r()` should be used instead.

The new `getlogin_r()` interface is as specified in POSIX 1003.1c Draft #10.

---

3C-500  

modified 22 Aug 1995
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);

MT-LEVEL
Safe

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_TOOLENGTH A line in the file exceeded the internal buffer size of
MNT_LINE_MAX.
MNT_TOOMANY A line in the file contains too many fields.
MNT_TOOFEW A line in the file contains too few fields.

modified 22 Jan 1993
On success, **putmntent()** returns the number of bytes printed to the specified file and on failure returns EOF.

**FILES**
/etc/mnttab

**SEE ALSO**
mnttab(4)

**NOTES**
The members of the **mnttab** structure point to information contained in a static area, so it must be copied if it is to be saved.
NAME  getnetbyname, getnetbyname_r, getnetbyaddr, getnetbyaddr_r, getnetent, getnetent_r, setnetent, endnetent – get network entry

SYNOPSIS  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);

MT-LEVEL  See the subsection “Reentrant Interfaces” in the 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.
endnetent() may be called to indicate that the caller expects to do no further network
entry retrieval operations; the system may then deallocate resources it was using. It is
still allowed, but possibly less efficient, for the process to call more network entry
retrieval functions after calling endnetent().

modified 9 Feb 1994  3N-503
Reentrant Interfaces

The functions `getnetbyname()`, `getnetbyaddr()`, and `getnetent()` use static storage that is re-used in each call, making these routines unsafe for use in multithreaded applications.

The functions:

- `getnetbyname_r()`,
- `getnetbyaddr_r()`,

and

- `getnetent_r()`

provide reentrant interfaces for these operations.

Each reentrant interface performs the same operation as its non-reentrant counterpart, named by removing the "_r" suffix. The reentrant interfaces, however, use buffers supplied by the caller to store returned results, and are safe for use in both single-threaded and multithreaded applications.

Each reentrant interface takes the same parameters as its non-reentrant counterpart, as well as the following additional parameters. The parameter `result` must be a pointer to a struct `netent` structure allocated by the caller. On successful completion, the function returns the network entry in this structure. The parameter `buffer` must be a pointer to a buffer supplied by the caller. This buffer is used as storage space for the network entry data. All of the pointers within the returned `struct netent result` point to data stored within this buffer (see RETURN VALUES). The buffer must be large enough to hold all of the data associated with the network entry. The parameter `buflen` should give the size in bytes of the buffer indicated by `buffer`.

For enumeration in multithreaded applications, the position within the enumeration is a process-wide property shared by all threads. `setnetent()` may be used in a multithreaded application but resets the enumeration position for all threads. If multiple threads interleave calls to `getnetent_r()`, the threads will enumerate disjoint subsets of the network database.

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

RETURN VALUES

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

```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
```

**SEE ALSO**

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

**WARNINGS**

The reentrant interfaces `getnetbyname_r()`, `getnetbyaddr_r()` and `getnetent_r()` are included in this release on an uncommitted basis only, and are subject to change or removal in future minor releases.

**NOTES**

The current implementation of these functions only return or accept network numbers for the Internet address family (type `AF_INET`). The functions described in `inet(3N)` may be helpful in constructing and manipulating addresses and network numbers in this form.

Programs that use the interfaces described in this manual page cannot be linked statically since the implementations of these functions employ dynamic loading and linking of shared objects at run time.

When compiling multithreaded applications, see `Intro(3)`, `Notes On Multithread Applications`, for information about the use of the `_REENTRANT` flag.

Use of the enumeration interfaces `getnetent()` and `getnetent_r()` is discouraged; enumeration may not be supported for all database sources. The semantics of enumeration are discussed further in `nsswitch.conf(4)`.

modified 9 Feb 1994
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);

MT-LEVEL
MT-Safe

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.
**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.

### SEE ALSO

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

*ONC+ Developers Guide*

*Transport Interfaces Programming Guide*
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);

MT-LEVEL
See the DESCRIPTION section of this page.

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;
genetgrent(&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.

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`

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

**WARNINGS**  
The function `getnetgrent_r()` is included in this release on an uncommitted basis only, and is subject to change or removal in future minor releases.

**NOTES**  
Only the Network Information Services, NIS and NIS+, are supported as sources for the `netgroup` database.

Programs that use the interfaces described in this manual page cannot be linked statically since the implementations of these functions employ dynamic loading and linking of shared objects at run time.

When compiling multi-threaded applications, see `Intro(3), Notes On Multithread Applications`, for information about the use of the `_REENTRANT` flag.
NAME  getnetpath, setnetpath, endnetpath – get /etc/netconfig entry corresponding to NETPATH component

SYNOPSIS  
#include <netconfig.h>

struct netconfig *getnetpath(void *handlep);
void *setnetpath(void);
int endnetpath(void *handlep);

MT-LEVEL  MT-Safe

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.

denetpath() 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).

**SEE ALSO**

- `getnetconfig(3N)`, `netconfig(4)`, `environ(5)`
- *ONC+ Developers Guide*
- *Transport Interfaces Programming Guide*
NAME  getopt – get option letter from argument vector

SYNOPSIS  
#include <stdlib.h>
int getopt(int argc, char * const *argv, const char *optstring);
extern char *optarg;
extern int optind, opterr, optopt;

MT-LEVEL  Unsafe

DESCRIPTION  getopt() returns the next option letter in argv that matches a letter in optstring. It supports all the rules of the command syntax standard (see intro(1)). Since all new commands are intended to adhere to the command syntax standard, they should use getopt(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().

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 command that can take the mutually exclusive options a and b, and the option o, which requires an argument:

#include <stdlib.h>
#include <stdio.h>
main (int argc, char **argv) {
  int c;
  extern char *optarg;
  extern int optind;
  int aflg = 0;
  int bflg = 0;
  int errflg = 0;
char *ofile = NULL;

while ((c = getopt(argc, argv, "abo:")) != EOF)
    switch (c) {
    case 'a':
        if (bflg)
            errflg++;
        else
            aflg++;
        break;
    case 'b':
        if (aflg)
            errflg++;
        else
            bflg++;
        break;
    case 'o':
        ofile = optarg;
        (void)printf("ofile = %s\n", ofile);
        break;
    case '?':
        errflg++;
    }
    if (errflg) {
        (void)fprintf(stderr,
                   "usage: cmd [-a|b] [-o <filename>] files ...\n");
        exit (2);
    }
    for ( ; optind < argc; optind++)
        (void)printf("%s\n", argv[optind]);
    return 0;
}

SEE ALSO intro(1), getopts(1), getopt(3C), getsubopt(3C), setlocale(3C), gettext(3I)

NOTES If the application is linked with -lintl, then messages printed from this function are in the
native language specified by the LC_MESSAGES locale category; see setlocale(3C).
The library routine getopt() does not fully check for mandatory arguments. That is,
given an option string ab and the input --a --b, getopt() assumes that --b is the man-
tory 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.
```
<table>
<thead>
<tr>
<th>NAME</th>
<th>gettext — get system page size</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYNOPSIS</td>
<td>#include &lt;unistd.h&gt;</td>
</tr>
<tr>
<td></td>
<td>int getpagesize(void);</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>gettext() returns the number of bytes in a page. Page granularity is the granularity of many of the memory management calls.</td>
</tr>
<tr>
<td></td>
<td>The page size is a system page size and need not be the same as the underlying hardware page size.</td>
</tr>
<tr>
<td>SEE ALSO</td>
<td>pagesize(1), brk(2), mmap(2), sysconf(3C)</td>
</tr>
</tbody>
</table>
NAME
getpass – read a password

SYNOPSIS
#include <stdlib.h>
char *getpass(const char *prompt);

MT-LEVEL
Unsafe

DESCRIPTION
getpass() reads up to a newline or EOF from the file /dev/tty, after prompting on the standard error output with the null-terminated string prompt and disabling echoing. A pointer is returned to a null-terminated string of at most 8 characters. If /dev/tty cannot be opened, a null pointer is returned. An interrupt will terminate input and send an interrupt signal to the calling program before returning.

FILES
/dev/tty

NOTES
The return value points to static data whose content is overwritten by each call.
NAME
getpeername – get name of connected peer

SYNOPSIS
cc [ flag . . . ] file . . . -lsocket -lnsl [ library . . . ]
int getpeername(int s, struct sockaddr *name, int *namelen);

MT-LEVEL
Safe

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 operation
to complete.
ENOTCONN    The socket is not connected.
ENOTSOCK    The argument s is not a socket.

SEE ALSO
accept(3N), bind(3N), getsockname(3N), socket(3N)
NAME
getpriority, setpriority – get/set scheduling priority for process, process group or user

SYNOPSIS
#include <sys/resource.h>

int getpriority(int which, id_t who);
int setpriority(int which, id_t who, int prio);

DESCRIPTION
The scheduling priority of the process, process group, or user, as indicated by which and who is obtained with getpriority() and set with setpriority() The default priority is 0; lower priorities cause more favorable scheduling.

which is one of PRIO_PROCESS, PRIO_PGRP, or PRIO_USER, and who is interpreted relative to which (a process identifier for PRIO_PROCESS, process group identifier for PRIO_PGRP, and a user ID for PRIO_USER). A zero value of who denotes the current process, process group, or user.

getpriority() returns the highest priority (lowest numerical value) enjoyed by any of the specified processes. setpriority() sets the priorities of all of the specified processes to the value specified by prio. If prio is less than −20, a value of −20 is used; if it is greater than 20, a value of 20 is used. Only the privileged user may lower priorities.

RETURN VALUES
Since getpriority() can legitimately return the value −1, it is necessary to clear the external variable errno prior to the call, then check it afterward to determine if a −1 is an error or a legitimate value. The setpriority() call returns 0 if there is no error, or −1 if there is.

ERRORS
getpriority() and setpriority() may return one of the following errors:
EINVAL which was not one of PRIO_PROCESS, PRIO_PGRP, or PRIO_USER.
ESRCH No process was located using the which and who values specified.

In addition to the errors indicated above, setpriority() may fail with one of the following errors returned:
EPERM A process was located, but one of the following is true:

- Neither its effective nor real user ID matched the effective user ID of the caller, and neither the effective nor the real user ID of the process executing the setpriority() was the privileged user.
- The call to getpriority() would have changed a process’ priority to a value lower than its current value, and the effective user ID of the process executing the call was not that of the privileged user.

SEE ALSO
nice(1), renice(1), fork(2)

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

modified 12 Feb 1993
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);

MT-LEVEL  See the NOTES section of this page.

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.byname” and “protocols.bynumber”, and the NIS+ table “protocols”. The sources and their lookup order are specified in the /etc/nsswitch.conf file (see nsswitch.conf(4) for details). Some name services such as NIS will return only one name for a host, whereas others such as NIS+ or DNS will return all aliases.

getprotobyname_r() and getprotobynumber_r() sequentially search from the beginning of the file until a matching protocol name or protocol number is found, or until an EOF is encountered.
getprotobyname() and getprotobynumber() have the same functionality as getprotobyname_r() and getprotobynumber_r() except that a static buffer is used to store returned results. These routines are unsafe in a multi-threaded application.

getprotoent_r() enumerates protocol entries: successive calls to getprotoent_r() will return either successive protocol entries or NULL. Enumeration may not be supported by some sources. Note that if multiple threads call getprotoent_r(), each will retrieve a subset of the protocol database.

getprotoent() has the same functionality as getprotoent_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 getprotoent_r() and getprotobyname_r(). Calls to getprotobyname_r(), getprotobyname(), getprotobynumber_r() and getprotobynumber() may leave the enumeration in an indeterminate state, so setprotoent() should be called before the first getprotoent_r() or getprotoent(). Note that setprotoent() has process-wide scope, and “rewinds” the protocol entries for all threads calling getprotoent_r() as well as main-thread calls to getprotoent().

derprotonent() may be called to indicate that protocol processing is complete; the system may then close any open protocols file, deallocate storage, and so forth. It is legitimate, but possibly less efficient, to call more protocol routines after endprotoent().

The internal representation of a protocol entry is a protoent structure defined in <netdb.h> with the following members:

```
char   *p_name;
char   **p_aliases;
int    p_proto;
```

RETURN VALUES

getprotobyname_r(), getprotobyname(), getprotobynumber_r(), and getprotobynumber() return a pointer to a struct protoent if they successfully locate the requested entry; otherwise they return NULL.

getprotoent_r() and getprotoent() return a pointer to a struct protoent if they successfully enumerate an entry; otherwise they return NULL, indicating the end of the enumeration.

ERRORS

getprotobyname_r(), getprotobynumber_r(), and getprotoent_r() will fail if the following is true:

ERANGE                  length of the buffer supplied by caller is not large enough to store the result.

FILES

/etc/protocols
/etc/nsswitch.conf

modified 29 Mar 1993
SEE ALSO intro(3), nsswitch.conf(4), protocols(4)

NOTES Although getprotobynum_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 getprotobynum_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 getprotobynum(), 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 getprotobynum_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);

MT-LEVEL
Safe

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 “publickey.byname” or the NIS+ table “cred.org_dir”. The sources and their lookup order are specified in the /etc/nsswitch.conf file (see nsswitch.conf(4)).

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.

SEE ALSO
secure_rpc(3N), nsswitch.conf(4), publickey(4)

WARNINGS
If getpublickey() gets the public key from any source other than NIS+, all authenticated NIS+ operations may fail. To ensure that this does not happen, edit the nsswitch.conf(4) file to make sure that the public key is obtained from NIS+.
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

SEE ALSO
getpwnam(3C), passwd(4)

NOTES
If the /etc/passwd and the /etc/group files have the ‘‘+’’ for the NIS entry, then getpwent() and getgwent() will not return NULL when the end of file is reached.

modified 22 Jan 1993

3C-523
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 *result, char *buffer, int buflen);
struct passwd *getpwent(void);
struct passwd *getpwent_r(struct passwd *result, char *buffer, int buflen);
struct passwd *getpwuid(uid_t uid);
struct passwd *getpwuid_r(uid_t uid, struct passwd *result, char *buffer, int buflen);
void setpwent(void);
void endpwent(void);
struct passwd *fgetpwent(FILE *f);
struct passwd *fgetpwent_r(FILE *f, struct passwd *result, char *buffer, int buflen);

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);

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

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)).

gotpwnam() searches for a password entry with the login name specified by the character string parameter name.
getpwuid() searches for a password entry with the (numeric) user id specified by the parameter uid.

The functions setpwent(), getpwent(), and endpwent() 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.

dpwent() 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-524 modified 22 Aug 1995
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 `result` (or `pwd` for the POSIX versions) 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 result` (or `grp`) point to data stored within this buffer; see RETURN VALUES. The buffer must be large enough to hold all the data associated with the password entry. The parameter `bufsize` (or `bufsize` for the POSIX versions) should give the size in bytes of `buffer`. The POSIX versions place a pointer to the modified `grp` structure in the `result` parameter, instead of returning a pointer to this structure.

For enumeration in multithreaded applications, the position within the enumeration is a process-wide property shared by all threads. `setpwent()` may be used in a multithreaded application but resets the enumeration position for all threads. If multiple threads interleave calls to `getpwent_r()`, the threads will enumerate disjoint subsets of the password database.

Like their non-reentrant counterparts, `getpwnam_r()` and `getpwuid_r()` leave the enumeration position in an indeterminate state.

RETURN VALUES

Password entries are represented by the `struct passwd` structure defined in `<pwd.h>`:

```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 22 Aug 1995
The functions `getpwnam()`, `getpwnam_r()`, `getpwuid()`, and `getpwuid_r()` each return a pointer to a `struct passwd` if they successfully locate the requested entry; otherwise they return NULL. The POSIX functions `getpwnam_r()` and `getpwuid_r()` return zero upon success, or the error number in case of failure.

The functions `getpwent()`, `getpwent_r()`, `fgetpwent()`, and `fgetpwent_r()` each return a pointer to a `struct passwd` if they successfully enumerate an entry; otherwise they return NULL, indicating the end of the enumeration.

The functions `getpwnam()`, `getpwuid()`, `getpwent()`, and `fgetpwent()` use static storage, so returned data must be copied before a subsequent call to any of these functions if the data is to be saved.

When the pointer returned by the reentrant functions `getpwnam_r()`, `getpwuid_r()`, `getpwent_r()`, and `fgetpwent_r()` is non-NULL, it is always equal to the result pointer that was supplied by the caller.

**ERRORS**

The reentrant functions `getpwnam_r()`, `getpwuid_r()`, `getpwent_r()`, and `fgetpwent_r()` will return NULL and set `errno` to `ERANGE` (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

**SEE ALSO**

nispasswd(1), passwd(1), yp passwd(1), Intro(2), Intro(3), cuserid(3S), getgrnam(3C), getlogin(3C), getspnam(3C), nsswitch.conf(4), passwd(4), shadow(4)

**NOTES**

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

Programs that use the interfaces described in this manual page cannot be linked statically since, the implementations of these functions employ dynamic loading and linking of shared objects at run time.

When compiling multithreaded applications, see `Intro(3), Notes On Multithread Applications`, for information about the use of the `_REENTRANT` flag.

Use of the enumeration interfaces `getpwent()` and `getpwent_r()` is discouraged; enumeration is supported for the passwd file, NIS, and NIS+, but in general is not efficient and may not be supported for all database sources. The semantics of enumeration are discussed further in `nsswitch.conf(4)`.

Previous releases allowed the use of `+` and `-` entries in `/etc/passwd` to selectively include and exclude NIS entries. The primary usage of these `+/`- entries is superseded by the name service switch, so the `+/`- form may not be supported in future releases.
If required, the `+/-' functionality can still be obtained for NIS by specifying `compat' as the source for `passwd'.

If the `+/-' functionality is required in conjunction with NIS+, specify both `compat' as the source for `passwd' and `nisplus' as the source for the pseudo-database `passwd_compat'.

See `passwd(4), shadow(4), and nsswitch.conf(4) for details.

If the `+/-' is used, both `/etc/shadow' and `/etc/passwd' should have the same `+' and `-' entries to ensure consistency between the password and shadow databases.

If a password entry from any of the sources contains an empty `uid' or `gid' field, that entry will be ignored by the files, NIS, and NIS+ name service switch backends. This will cause the user to appear unknown to the system.

If a password entry contains an empty `gecos', `home directory', or `shell' field, `getpwnam()' and `getpwnam_r()' return a pointer to a null string in the respective field of the `passwd' structure.

If the shell field is empty, `login(1)' automatically assigns the default shell. See `login(1)'.

Reentrant interfaces `getpwnam_r()', `getpwent_r()', `getpwuid_r()', and `fgetpwent_r()' are as specified in POSIX 1003.1c Draft #10.
NAME
getrpcbyname, getrpcbyname_r, getrpcbyname, getrpcbynumber, getrpcbynumber_r, getrpcent,
getrpcent_r, setrpcent, endrpcent - get RPC entry

SYNOPSIS
cc [flag ...] file ... -lnsl [ 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 *getrpcbynumber(const int number);
struct rpcent *getrpcbynumber_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);

MT-LEVEL
See the subsection "Reentrant Interfaces" in the DESCRIPTION section of this page.

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.
getrpcbynumber() 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 getrpcbynumber() 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.
endrpcent() may be called to indicate that the caller expects to do no further RPC entry retrieval operations; the system may then deallocate resources it was using. It is still allowed, but possibly less efficient, for the process to call more RPC entry retrieval functions after calling endrpcent().
Reentrant Interfaces

The functions `getrpcbyname()`, `getrpcbynumber()`, and `getrpcent()` use static storage that is re-used in each call, making these routines unsafe for use in multithreaded applications.

The functions:

```c
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>`:

```c
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.

modified 29 Mar 1993

3N-529
The functions `getrpcbyname()`, `getrpcbynumber()`, and `getrpcent()` use static storage, so returned data must be copied before a subsequent call to any of these functions if the data is to be saved.

When the pointer returned by the reentrant functions `getrpcbyname_r()`, `getrpcbynumber_r()`, and `getrpcent_r()` is non-NULL, it is always equal to the `result` pointer that was supplied by the caller.

**ERRORS**

The reentrant functions `getrpcbyname_r()`, `getrpcbynumber_r()`, and `getrpcent_r()` will return NULL and set `errno` to `ERANGE` if the length of the buffer supplied by caller is not large enough to store the result. See `intro(2)` for the proper usage and interpretation of `errno` in multithreaded applications.

**FILES**

/etc/rpc
/etc/nsswitch.conf

**SEE ALSO**

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

**WARNINGS**

The reentrant interfaces `getrpcbyname_r()`, `getrpcbynumber_r()`, and `getrpcent_r()` are included in this release on an uncommitted basis only, and are subject to change or removal in future minor releases.

**NOTES**

Programs that use the interfaces described in this manual page cannot be linked statically since the implementations of these functions employ dynamic loading and linking of shared objects at run time.

When compiling multithreaded applications, see `Intro(3)`, `Notes On 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)`.
NAME  getrusage – get information about resource utilization

SYNOPSIS  

```c
#include <sys/resource.h>

int getrusage(int who, struct rusage *rusage);
```

DESCRIPTION  

getrusage() returns information about the resources utilized by the current process, or all its terminated child processes. The interpretation for some values reported, such as ru_idrss, are dependent on the clock tick interval. This interval is an implementation dependent value.

The who parameter is one of RUSAGE_SELF or RUSAGE_CHILDREN. The buffer to which rusage points will be filled in with a structure with the following members:

```c
struct timeval ru_utime; /* user time used */
struct timeval ru_stime; /* system time used */
int ru_maxrss; /* maximum resident set size */
int ru_idrss; /* integral resident set size */
int ru_minflt; /* page faults not requiring physical I/O */
int ru_majflt; /* page faults requiring physical I/O */
int ru_nswap; /* swaps */
int ru_inblock; /* block input operations */
int ru_oublock; /* block output operations */
int ru_msgsnd; /* messages sent */
int ru_msgrcv; /* messages received */
int ru_nsignals; /* signals received */
int ru_nvcsw; /* voluntary context switches */
int ru_nivcsw; /* 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.
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_swap  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_signals  The number of signals delivered.

ru_nvcsw  The number of times a context switch resulted due to a process voluntarily giving up the processor before its time slice was completed (usually to await availability of a resource).

ru_nivcsw  The number of times a context switch resulted due to a higher priority process becoming runnable or because the current process exceeded its time slice.

RETURN VALUES  If successful, the value of the appropriate structure is filled in, and 0 is returned. If the call fails, −1 is returned.

ERRORS  getrusage() will fail if:

EFAULT  The address specified by the rusage argument is not in a valid portion of the process’s address space.

EINVAL  The who parameter is not a valid value.

SEE ALSO  sar(1M), read(2), times(2), wait(2), write(2), getpagesize(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.
In addition to hardware detected page faults, the kernel may cause pseudo page faults in order to perform some housekeeping. For example, the kernel may generate page faults, even if the pages exist in physical memory, in order to lock down pages involved in a raw I/O request.

By definition, major page faults require physical I/O, while minor page faults do not require physical I/O. For example, reclaiming the page from the free list would avoid I/O and generate a minor page fault. More commonly, minor page faults occur during process startup as references to pages which are already in memory. For example, if an address space faults on some “hot” executable or shared library, this results in a minor page fault for the address space. Also, any one doing a `read(2)` or `write(2)` to something that is in the page cache will get a minor page fault(s) as well.

There is no way to obtain information about a child process which has not yet terminated.
NAME
gets, fgets – get a string from a stream

SYNOPSIS
#include <stdio.h>
char *gets(char *s);
char *fgets(char *s, int n, FILE *stream);

MT-LEVEL
MT-Safe

DESCRIPTION
gets() 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.

fgets() 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.

SEE ALSO
lseek(2), read(2), ferror(3S), fopen(3S), fread(3S), getc(3S), scanf(3S), stdio(3S), ungetc(3S)
NAME  getservbyname, getservbyname_r, getservbyport, getservbyport_r, getservent, 
getservent_r, setservent, endservent − get service entry

SYNOPSIS  cc [flag . . .] file . . . −lssocket -llnsl [library . . .]
#include <netdb.h>
struct servent *getservbyname(const char *name, const char *proto);
struct servent *getservbyname_r(const char *name, const char *proto,
    struct servent *result, char *buffer, int buflen);
struct servent *getservbyport(int port, const char *proto);
struct servent *getservbyport_r(int port, const char *proto, struct servent *result,
    char *buffer, int buflen);
struct servent *getservent(void);
struct servent *getservent_r(struct servent *result, char *buffer, int buflen);
int setservent(int stayopen);
int endservent(void);

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

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().

modified 17 Jan 1995
getservbyport_r() provides 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 setservent_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 */
};
```

The members of this structure are:

- **s_name**: The official name of the service.
- **s_aliases**: A zero terminated list of alternate names for the service.
- **s_port**: The port number at which the service resides. Port numbers are returned in network byte order.
- **s_proto**: The name of the protocol to use when contacting the service.

The functions `getservbyname()`, `getservbyname_r()`, `getservbyport()`, and `getservbyport_r()` each return a pointer to a `struct servent` if they successfully locate the requested entry; otherwise they return `NULL`.

The functions `getservent()` and `getservent_r()` each return a pointer to a `struct servent` if they successfully enumerate an entry; otherwise they return `NULL`, indicating the end of the enumeration.

The functions `getservbyname()`, `getservbyport()`, and `getservent()` use static storage, so returned data must be copied before a subsequent call to any of these functions if the data is to be saved.

When the pointer returned by the reentrant functions `getservbyname_r()`, `getservbyport_r()`, and `getservent_r()` is non-null, it is always equal to the `result` pointer that was supplied by the caller.

ERRORS

The reentrant functions `getservbyname_r()`, `getservbyport_r()` and `getservent_r()` will return `NULL` and set `errno` to `ERANGE` if the length of the buffer supplied by caller is not large enough to store the result. See `intro(2)` for the proper usage and interpretation of `errno` in multithreaded applications.

FILES

- `/etc/services`: Internet network services
- `/etc/netconfig`: network configuration file
- `/etc/nsswitch.conf`: configuration file for the name-service switch

SEE ALSO

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

WARNINGS

The reentrant interfaces `getservbyname_r()`, `getservbyport_r()`, and `getservent_r()` are included in this release on an uncommitted basis only, and are subject to change or removal in future minor releases.

modified 17 Jan 1995
NOTES

The functions that return `struct servent` return the least significant 16-bits of the `s_port` field in network byte order. `getservbyport()` and `getservbyport_r()` also expect the input parameter `port` in the network byte order. See `htons(3N)` for more details on converting between host and network byte orders.

Programs that use the interfaces described in this manual page cannot be linked statically since the implementations of these functions employ dynamic loading and linking of shared objects at run time.

In order to ensure that they all return consistent results, `getservbyname()`, `getservbyname_r()`, and `netdir_getbyname()` are implemented in terms of the same internal library function. This function obtains the system-wide source lookup policy based on the `inet` family entries in `netconfig(4)` and the `services: entry in nsswitch.conf(4)`. Similarly, `getservbyport()`, `getservbyport_r()`, and `netdir_getbyaddr()` are implemented in terms of the same internal library function. If the `inet` family entries in `netconfig(4)` have a “-” in the last column for nametoaddr libraries, then the entry for `services` in `nsswitch.conf` will be used; otherwise the name-toaddr libraries in that column will be used, and `nsswitch.conf` will not be consulted.

There is no analogue of `getservent()` and `getservent_r()` in the netdir functions, so these enumeration functions go straight to the `services` entry in `nsswitch.conf`. Thus enumeration may return results from a different source than that used by `getservbyname()`, `getservbyname_r()`, `getservbyport()`, and `getservbyport_r()`.

When compiling multithreaded applications, see `intro(3), Notes On Multithread Applications`, for information about the use of the `_REENTRANT` flag.

Use of the enumeration interfaces `getservent()` and `getservent_r()` is discouraged; enumeration may not be supported for all database sources. The semantics of enumeration are discussed further in `nsswitch.conf(4)`.
NAME  
getsockname – get socket name

SYNOPSIS  
cc [ flag ...] file ... -lsocket -lnsl [ library ...]
#include <sys/types.h>
#include <sys/socket.h>
int getsockname(int s, struct sockaddr *name, int *namelen);

MT-LEVEL  
Safe

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.

SEE ALSO  
bind(3N), getpeername(3N), socket(3N)
getsockopt (3N)  Network Functions  SunOS 5.5

NAME
getsockopt, setsockopt – get and set options on sockets

SYNOPSIS
cc [ flag ... ] file ... -lsocket -lssl [ 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);

MT-LEVEL
Safe

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 getprotobynumber(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 setsockopt(), 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

3N-540  modified 8 Mar 1994
SO_BROADCAST enable/disable permission to transmit broadcast messages
SO_OOBINLINE enable/disable reception of out-of-band data in band
SO_SNDBUF set buffer size for output
SO_RCVBUF set buffer size for input
SO_TYPE get the type of the socket (get only)
SO_ERROR get and clear error on the socket (get only)
SO_DEBUG enables debugging in the underlying protocol modules. SO_REUSEADDR indicates that the rules used in validating addresses supplied in a bind(3N) call should allow reuse of local addresses. SO_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. The Internet protocols place an absolute limit of 64 Kbytes on these values for UDP and TCP sockets.

Finally, SO_TYPE and SO_ERROR are options used only with getsockopt(). SO_TYPE returns the type of the socket (for example, SOCK_STREAM). It is useful for servers that inherit sockets on startup. SO_ERROR returns any pending error on the socket and clears the error status. It may be used to check for asynchronous errors on connected datagram sockets or for other asynchronous errors.

**RETURN VALUES**

If successful, getsockopt() returns 0; otherwise, it returns −1 and sets errno to indicate the error.
### ERRORS

The call succeeds unless:

- **EBADF**  
  The argument 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 is not a socket.

### SEE ALSO

- close(2), ioctl(2), bind(3N), getprotobynmae(3N), socket(3N)
NAME  
getspnam, getspnam_r, getspent, getspent_r, setspent, endspent, fgetspent, fgetspent_r — 
geet password entry

SYNOPSIS  
#include <shadow.h>

struct spwd *getspnam(const char *name);
struct spwd *getspnam_r(const char *name, struct spwd *result, char *buffer, int buflen);
struct spwd *getspent(void);
struct spwd *getspent_r(struct spwd *result, char *buffer, int buflen);
void setspent(void);
void endspent(void);
struct spwd *fgetspent(FILE *fp);
struct spwd *fgetspent_r(FILE *fp, struct spwd *result, char *buffer, int buflen);

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

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

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

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

setspent() sets (or resets) the enumeration to the beginning of the set of shadow password entries. This function should be called before the first call to getspent(). Calls to getspnam() leave the enumeration position in an indeterminate state. Successive calls to getspent() return either successive entries or NULL, indicating the end of the enumeration.

endspent() may be called to indicate that the caller expects to do no further shadow password retrieval operations; the system may then close the shadow password file, deallocate resources it was using, and so forth. It is still allowed, but possibly less efficient, for the process to call more shadow password functions after calling endspent().

fgetspent(), unlike the other functions above, does not use nsswitch.conf; it reads and parses the next line from the stream f, which is assumed to have the format of the shadow file (see shadow(4)).

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

modified 28 Mar 1993
The functions:

```
getspnam_r(),
gspent_r(),
```

and

```
gfgetspent_r()  
```

provide reentrant interfaces for these operations.

Each reentrant interface performs the same operation as its non-reentrant counterpart, named by removing the ‘‘_r’’ suffix. The reentrant interfaces, however, use buffers supplied by the caller to store returned results, and are safe for use in both single-threaded and multithreaded applications.

Each reentrant interface takes the same parameters as its non-reentrant counterpart, as well as the following additional parameters. The parameter `result` must be a pointer to a `struct spwd` structure allocated by the caller. On successful completion, the function returns the shadow password entry in this structure. The parameter `buffer` must be a pointer to a buffer supplied by the caller. This buffer is used as storage space for the shadow password data. All of the pointers within the returned `struct spwd` `result` point to data stored within this buffer (see RETURN VALUES). The buffer must be large enough to hold all of the data associated with the shadow password entry. The parameter `buflen` should give the size in bytes of the buffer indicated by `buffer`.

For enumeration in multithreaded applications, the position within the enumeration is a process-wide property shared by all threads. `setspent()` may be used in a multithreaded application but resets the enumeration position for all threads. If multiple threads interleave calls to `getspent_r()`, the threads will enumerate disjoint subsets of the shadow password database.

Like its non-reentrant counterpart, `getspnam_r()` leaves the enumeration position in an indeterminate state.

**RETURN VALUES**

Password entries are represented by the `struct spwd` structure defined in `<shadow.h>`:

```
struct spwd{
    char *sp_namp; /* login name */
    char *sp_pwdp; /* encrypted passwd */
    long sp_lstchg; /* date of last change */
    long sp_min; /* min days to passwd change */
    long sp_max; /* max days to passwd change*/
    long sp_warn; /* warning period */
    long sp_inact; /* max days inactive */
    long sp_expire; /* account expiry date */
    unsigned long sp_flag; /* not used */
};
```

See `shadow(4)` for more information on the interpretation of this information.

The functions `getspnam()`and `getspnam_r()` each return a pointer to a `struct spwd` if they successfully locate the requested entry; otherwise they return NULL.
The functions \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 NULL, indicating the end of the enumeration.

The functions \texttt{getspnam()}, \texttt{getspent()}, and \texttt{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 \texttt{getspnam_r()}, \texttt{getspent_r()}, and \texttt{fgetspent_r()} is non-NULL, it is always equal to the \textit{result} pointer that was supplied by the caller.

**ERRORS**

The reentrant functions \texttt{getspnam_r()}, \texttt{getspent_r()}, and \texttt{fgetspent_r()} will return NULL and set \texttt{errno} to \texttt{ERANGE} if the length of the buffer supplied by caller is not large enough to store the result. See \texttt{intro(2)} for the proper usage and interpretation of \texttt{errno} in multithreaded applications.

**FILES**

/etc/shadow
/etc/nsswitch.conf
/etc/passwd

**SEE ALSO**

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

**WARNINGS**

The reentrant interfaces \texttt{getspnam_r()}, \texttt{getspent_r()}, and \texttt{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 \texttt{intro(3), Notes On Multithread Applications}, for information about the use of the \texttt{_REENTRANT} flag.

Use of the enumeration interfaces \texttt{getspent()} and \texttt{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 \texttt{nsswitch.conf(4)}.

Access to shadow password information may be restricted in a manner depending on the database source being used. Access to the \texttt{/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 “\texttt{passwd.byname}” map. This map stores only the information for the \texttt{sp_namp} and \texttt{sp_pwdp} fields of the \texttt{struct spwd} structure. Shadow password entries obtained from NIS will contain the value \texttt{-1} in the remainder of the fields.

modified 28 Mar 1993
When NIS+ is used as the database source, and the caller lacks the permission needed to retrieve the encrypted password from the NIS+ "passwd.org_dir" table, the NIS+ service returns the string "*NP*" instead of the actual encrypted password string. The functions described on this page will then return the string "*NP*" to the caller as the value of the member `sp_pwdp` in the returned shadow password structure.
NAME  
getsubopt – parse suboptions from a string

SYNOPSIS  
#include <stdlib.h>

int getsubopt(char **optionp, const char * const *tokens, char **valuep);

MT-LEVEL  
MT-Safe

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 variable 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 character 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[] = {
#define READONLY 0
    "ro",
#define READWRITE 1
    "rw",
#define WRITESIZE 2
    "wsize",
#define READSIZE 3
    "rsize",
    NULL
};

main(argc, argv)
    int argc;
    char **argv;
{
    int sc, c, errflag;
    char *options, *value;
    extern char *optarg;
    extern int optind;

    while((c = getopt(argc, argv, "abf:o:")) != -1) {
        switch (c) {
            case 'a': /* process a option */
                break;
            case 'b': /* process b option */
                break;
            case 'f':
                ofile = optarg;
                break;
            case '?':
                errflag++;
                break;
            case 'o':
                options = optarg;
                while (*options != '\0') {
                    switch(getsubopt(&options,myopts,&value) {
                        case READONLY : /* process ro option */
                            break;
                        case READWRITE : /* process rw option */
                            break;
                    }
                }
        }
    }
}
case WRITESIZE : /* process wsize option */
    if (value == NULL) {
        error_no_arg();
        errflag++;  
    } else
        write_size = atoi(value);
    break;

case READSIZE : /* process rsize option */
    if (value == NULL) {
        error_no_arg();
        errflag++;  
    } else
        read_size = atoi(value);
    break;

default :
    /* process unknown token */
    error_bad_token(value);
    errflag++;  
    break;

    break;

if (errflag) {
/* print usage instructions etc. */
    
}  
for (; optind<argc; optind++) {
    /* process remaining arguments */
    
    .
    .
    .

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

NOTES  
During parsing, commas in the option input string are changed to null characters. White space in tokens or token-value pairs must be protected from the shell by quotes.
**NAME**
gettext, dgettext, dcgettext, textdomain, bindtextdomain – message handling functions

**SYNOPSIS**
cc [ flag ...] file ... -lintl [ library ...]
#include <libintl.h>
#include <locale.h>

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);

**MT-LEVEL**
Safe with exceptions

**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 dcgettext() simply return the message string that was passed. In a locale other than "C", if NLSPATH is not defined or if a message catalogue is not found in any of the components specified by NLSPATH, the routines search for the message catalogue 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().

3I-550 modified 12 May 1995
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. Ifmsgid 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() returnmsgid.

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 dgettext() calls where LC_XXX is LC_CTYPE, LC_NUMERIC,
LC_TIME, LC_COLLATE, LC_MONETARY, or LC_MESSAGES.
**dirname**

location for file containing messages for domain *domainname* and path predicate *dirname* after a successful call to **bindtextdomain()**

**dirname**/*LC_XXX*/domainname*.mo

location for files containing messages for domain *domainname*, language *locale*, and path predicate *dirname* after a successful call to **bindtextdomain()** for **dcgettext()** calls where *LC_XXX* is one of **LC_CTYPE**, **LC_NUMERIC**, **LC_TIME**, **LC_COLLATE**, **LC_MONETARY**, or **LC_MESSAGES**.

**SEE ALSO**

msgfmt(1), xgettext(1), setlocale(3C), 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.

modified 18 Feb 1993
NAME
gettimeofday, settimeofday – get or set the date and time

SYNOPSIS
#include <sys/time.h>
int gettimeofday(struct timeval *tp, void *);
int settimeofday(struct timeval *tp, void *);

MT-LEVEL
MT-Safe

DESCRIPTION
gettimeofday() gets and settimeofday() sets the system’s notion of the current time. The
current time is expressed in elapsed seconds and microseconds since 00:00 Universal
Coordinated Time, January 1, 1970. The resolution of the system clock is hardware
dependent; the time may be updated continuously or in clock ticks.

.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.
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 privileged user may set the time of day.

SEE ALSO
adjtime(2), ctime(3C), TIMEZONE(4)

RETURN VALUES
A -1 return value indicates that an error occurred and errno has been set.

ERRORS
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 or
time zone.

NOTES
The implementation of settimeofday() ignores the tv_usec field of tp. If the time needs
to be set with better than one second accuracy, call settimeofday() for the seconds and
then adjtime() for finer accuracy.
NAME
gettxt – retrieve a text string

SYNOPSIS
#include <nl_types.h>
char *gettxt(const char *msgid, const char *dflt_str);

MT-LEVEL
Safe with exceptions

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 vari-
able 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 argu-
ment dflt_str points to. A pointer to the second argument is returned if the second argu-
ment 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.

modified 22 Jan 1993
FILES

/usr/lib/locale/C/LC_MESSAGES/* contains default message files created by `mkmsgs`

/usr/lib/locale/locale/LC_MESSAGES/* contains message files for different languages created by `mkmsgs`

SEE ALSO

`exstr(1)`, `mkmsgs(1)`, `srttxt(1)`, `gettext(3I)`, `fmtmsg(3C)`, `setlocale(3C)`, `environ(5)`

NOTES

It is recommended that `gettext(3I)` be used in place of this routine.
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.

modified 30 Jan 1995  3C-557
**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);
```

**MT-LEVEL**
Unsafe

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

`setutent()` resets the input stream to the beginning of the file. This reset should be done before each search for a new entry if it is desired that the entire file be examined.

`endutent()` closes the currently open file.

`utmpname()` allows the user to change the name of the file examined, from `/var/adm/utmp` to any other file. It is most often expected that this other file will be `/var/adm/wtmp`. If the file does not exist, this will not be apparent until the first attempt to reference the file is made. `utmpname()` does not open the file. It just closes the old file if it is currently open and saves the new file name.

**RETURN VALUES**

A null pointer is returned upon failure to read, whether for permissions or having reached the end of file, or upon failure to write. If the file name given is longer than 79 characters, `utmpname()` returns 0. Otherwise, it returns 1.

**FILES**

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

**SEE ALSO**

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

**NOTES**

The most current entry is saved in a static structure. Multiple accesses require that it be copied before further accesses are made. On each call to either `getutid()` or `getutline()`, the routine examines the static structure before performing more I/O. If the contents of the static structure match what it is searching for, it looks no further. For this reason, to use `getutline()` to search for multiple occurrences, it would be necessary to zero out the static area after each success, or `getutline()` would just return the same structure over and over again. There is one exception to the rule about emptying the structure before further reads are done. The implicit read done by `pututline()` (if it finds that it is not already at the correct place in the file) will not hurt the contents of the static structure returned by the `getutent()`, `getutid()` or `getutline()` routines, if the user has just modified those contents and passed the pointer back to `pututline()`.

These routines use buffered standard I/O for input, but `pututline()` uses an unbuffered non-standard write to avoid race conditions between processes trying to modify the `utmp` and `wtmp` files.
NAME     getutxent, getutxid, getutxline, pututxline, setutxent, endutxent, utmpxname, getutmp, getutmpx, updwtmp, updwtmpx – access utmpx file entry

SYNOPSIS     #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);

MT-LEVEL     Unsafe

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 */
                      /* (usually line #) */
char ut_line[32];          /* device name (console, lnxx) */
pid_t ut_pid;              /* process id */
short ut_type;             /* type of entry */
struct exit_status ut_exit; /* exit status of a process */
                      /* marked as DEAD_PROCESS */
struct timeval ut_tv;      /* time entry was made */
long ut_session;           /* session ID, used for windowing */
long pad[5];               /* reserved for future use */
short ut_syslen;           /* significant length of ut_host */
                      /* including terminating null */
char ut_host[257];         /* host name, if remote */

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,
### SunOS 5.5 C Library Functions  getutxent (3C)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>USER_PROCESS, or DEAD_PROCESS</td>
<td>then getutxid() 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.</td>
</tr>
<tr>
<td>getutxline()</td>
<td>Searches forward from the current point in the utmpx file until it finds an entry of the type LOGIN_PROCESS or USER_PROCESS which also has a ut_line string matching the line→ut_line string. If the end of file is reached without a match, it fails.</td>
</tr>
<tr>
<td>pututxline()</td>
<td>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.</td>
</tr>
<tr>
<td>setutxent()</td>
<td>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.</td>
</tr>
<tr>
<td>endutxent()</td>
<td>Closes the currently open file.</td>
</tr>
<tr>
<td>utmpxname()</td>
<td>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.</td>
</tr>
<tr>
<td>getutmp()</td>
<td>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)</td>
</tr>
<tr>
<td>getutmpx()</td>
<td>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)</td>
</tr>
<tr>
<td>updwtmp()</td>
<td>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.</td>
</tr>
</tbody>
</table>

modified 4 Apr 1995 3C-561
updwtmpx() Checks the existence of wfilex and its parallel file, whose name is obtained by truncating the final “x” from wfilex. If only one of them exists, the second one is created and initialized to reflect the state of the existing file. utmpx is written to wfilex, and the corresponding 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
/var/adm/utmp contains current user access and adminstrative information (old format)
/var/adm/utmpx contains current user access and administration information (new format)
/var/adm/wtmp contains a history of user access and adminstrative information.
/var/adm/wtmpx contains a history of user access and administrative information.

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

NOTES The most current entry is saved in a static structure. Multiple accesses require that it be copied before further accesses are made. On each call to either getutxid() or getutxline(), the routine examines the static structure before performing more I/O. If the contents of the static structure match what it is searching for, it looks no further. For this reason, to use getutxline() to search for multiple occurrences it would be necessary to zero out the static after each success, or getutxline() would just return the same structure over and over again. There is one exception to the rule about emptying the structure before further reads are done. The implicit read done by pututxline() (if it finds that it is not already at the correct place in the file) will not hurt the contents of the static structure returned by the getutxent(), getutxid(), or getutxline() routines, if the user has just modified those contents and passed the pointer back to pututxline().

These routines use buffered standard I/O for input, but pututxline() uses an unbuffered write to avoid race conditions between processes trying to modify the utmpx and wtmpx files.
NAME
getvfsent, getvfsfile, getvfsspec, getvfsany – get vfstab file entry

SYNOPSIS
#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, void *vref);

MT-LEVEL
Safe

DESCRIPTION
getvfsent() fills 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.

modified 22 Jan 1993
NOTES
The members of the `vfstab` structure point to information contained in a static area, so it must be copied if it is to be saved.

FILES
`/etc/vfstab`

SEE ALSO
`vfstab(4)`
NAME  
getwc, getwchar, fgetwc – convert EUC character from the stream to Process Code

SYNOPSIS  
cc [ flag … ] file … –lw [ library … ]
#include <stdio.h>
#include <widec.h>

wint_t getwc(FILE *stream);
wint_t getwchar(void);
wint_t fgetwc(FILE *stream);

MT-LEVEL  
MT-Safe

DESCRIPTION  
getwc() and fgetwc() convert the next Extended Unix Code (EUC) character from the
named input stream into a wchar_t Process Code character, and return it as an integer.
They also move the file pointer, if defined, ahead one EUC character in the stream.
getwchar() is defined as fgetwc(stdin). getwc() and getwchar() are macros.

RETURN VALUES  
These functions return the integer constant EOF at end-of-file or upon an error. The end-
of-file condition is remembered, even on a terminal, and all subsequent attempts to read
will return EOF until the condition is cleared with clearerr() (see ferror(3S)).

SEE ALSO  
ferror(3S), fopen(3S), fread(3S), getws(3I), putwc(3I), scanf(3S), ungetwc(3I)

WARNING  
If the integer value returned by getwc(), getwchar(), or fgetwc() is stored into a wchar_t
variable and then compared against the integer constant EOF, the comparison will not
succeed, because wchar_t is defined as unsigned.

modified 22 Jan 1993
NAME
getwd – get current working directory pathname

SYNOPSIS
#include <sys/param.h>
#include <unistd.h>
char *getwd(char *pathname);

DESCRIPTION
getwd() copies the absolute pathname of the current working directory to pathname and returns a pointer to the result.

RETURN VALUES
getwd() returns zero and places a message in pathname if an error occurs. pathname points to a character array of at least MAXPATHLEN length.

SEE ALSO
getcwd(3C)

modification details:
modified 3 Mar 1995
NAME

getwidth – get codeset information

SYNOPSIS

cc [ flag ...] file ... -lw [ library ... ]
#include <euc.h>
#include <getwidth.h>
void getwidth(eucwidth_t *ptr);

MT-LEVEL

MT-Safe with exceptions

DESCRIPTION

The getwidth() function reads the character class table for the current locale, generated by wchrtbl() (see chrtbl(1M)) to get information on the supplementary codesets. getwidth() sets this information into the struct eucwidth_t. This struct is defined in <euc.h> and has the following members:

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.

SEE ALSO

chrtbl(1M), euclen(3I), setlocale(3C)

NOTES

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

modified 22 Jan 1993
NAME
getws, fgetws – convert a string of EUC characters from the stream to Process Code

SYNOPSIS
c
#include <stdio.h>
#include <widec.h>
wchar_t *getws(wchar_t *s);
wchar_t *fgetws(wchar_t *s, int n, FILE *stream);

MT-LEVEL
MT-Safe

DESCRIPTION getws() reads a string of Extended Unix Code (EUC) characters from the standard input
stream, stdin, converts it to process code, and writes it to the array pointed to by 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.
getws() returns its argument.

fgetws() 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. fgetws() 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.

SEE ALSO ferror(3S), fread(3S), getwc(3I), putws(3I), scanf(3S)
NAME
glob, globfree – generate path names matching a pattern

SYNOPSIS
#include <glob.h>

int glob( const char *pattern, int flags, int(*errfunc)(const char *path, int eerrno),
glob_t *pglob);
void globfree(glob_t *pglob);

MT-LEVEL
MT-Safe

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_DOOFFS 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

modified 26 Jan 1995
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.

The following constants are defined as error return values for glob():

GLOB_ABORTED The scan was stopped because GLOB_ERR was set or (*errfunc) returned non-zero.
GLOB_NOMATCH The pattern does not match any existing path name, and
GLOB_NOCHECK was not set in flags.
GLOB_NOSPACE An attempt to allocate memory failed.

If (*errfunc) is called and returns non-zero, or if the GLOB_ERR flag is set in flags, glob() stops the scan and returns GLOB_ABORTED after setting gl_pathc and gl_pathv in pglob to reflect the paths already scanned. If GLOB_ERR is not set and either errfunc is a NULL pointer or (*errfunc) returns 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_DOOFFS, NULL, &globbuf);
globbuf.gl_pathv[0] = "ls";
globbuf.gl_pathv[1] = "-l";
execvp("ls", &globbuf.gl_pathv[0]);
```

Using the same example:

```
ls -l *.c *.h
```

could be approximately simulated using `GLOB_APPEND` as follows:

```c
globbuf.gl_offs = 2;
glob("*.c", GLOB_DOOFFS, NULL, &globbuf);
glob("*.h", GLOB_DOOFFS | GLOB_APPEND, NULL, &globbuf);
...
```

**SEE ALSO** `execv(2)`, `stat(2)`, `fnmatch(3C)`, `opendir(3C)`, `readdir(3C)`, `wordexp(3C)`
### NAME

`gmatch` – shell global pattern matching

### SYNOPSIS

```c
cc [... file ... -lgen [ library ... ]
#include <libgen.h>
int gmatch(const char *str, const char *pattern);
```

### MT-LEVEL

MT-Safe

### 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.

```c
char *s;

    gmatch(s, "*[a-]");
```

### SEE ALSO

`sh(1)`

### 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 22 Jan 1993

---

3G-573
NAME  grantpt – grant access to the slave pseudo-terminal device

SYNOPSIS  int grantpt(int fildes);

MT-LEVEL  Safe

DESCRIPTION  The function grantpt() changes the mode and ownership of the slave pseudo-terminal device associated with its master pseudo-terminal counter part. 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, writable by the owner and writeable by the group.

RETURN VALUES  Upon successful completion, the function grantpt() returns 0; otherwise it returns -1. Failure could occur if fildes is not an open file descriptor, if fildes is not associated with a master pseudo-terminal device, or if the corresponding slave device could not be accessed. grantpt() will also fail if it is unable to successfully invoke the setuid root program.

SEE ALSO  open(2), setuid(2), ptsname(3C), unlockpt(3C)

STREAMS Programming Guide
NAME  hsearch, hcreate, hdestroy – manage hash search tables

SYNOPSIS  
```c
#include <search.h>
ENTRY *hsearch(ENTRY item, ACTION action);
int hcreate (size_t melenents);
void hdestroy(void);
```

MT-LEVEL  Safe

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.
```c
#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 */
};
```

modified 22 Jan 1993
#define NUM_EMPL 5000  /* # of elements in search table */

main( )
{
    /* space to store strings */
    char string_space[NUM_EMPL*20];
    /* space to store employee info */
    struct info info_space[NUM_EMPL];
    /* next avail space in string_space */
    char *str_ptr = string_space;
    /* next avail space in info_space */
    struct info *info_ptr = info_space;
    ENTRY item, *found_item;
    /* name to look for in table */
    char name_to_find[30];
    int i = 0;

    /* create table */
    (void) hcreate(NUM_EMPL);
    while (scanf("%s%d%d", str_ptr, &info_ptr->age,
                &info_ptr->room) != EOF && i++ < NUM_EMPL) {
        /* put info in structure, and structure in item */
        item.key = str_ptr;
        item.data = (void *)info_ptr;
        str_ptr += strlen(str_ptr) + 1;
        info_ptr++;
        /* put item into table */
        (void) hsearch(item, ENTER);
    }

    /* access table */
    item.key = name_to_find;
    while (scanf("%s", item.key) != EOF) {
        if ((found_item = hsearch(item, FIND)) != NULL) {
            /* if item is in the table */
            (void) printf("found %s, age = %d, room = %d\n", 
                          found_item->key,
                          ((struct info *)found_item->data)->age,
                          ((struct info *)found_item->data)->room);
        } else {
            (void) printf("no such employee %s\n", 
                          name_to_find);
        }
    }
    return 0;
}
SEE ALSO  

**hsearch( )** and **hcreate( )** use **malloc(3C)** to allocate space.  
Only one hash search table may be active at any given time.
NAME
hyperbolic, sinh, cosh, tanh, asinh, acosh, atanh – hyperbolic functions

SYNOPSIS
#include <math.h>
double sinh(double x);
double cosh(double x);
double tanh(double x);
double asinh(double x);
double acosh(double x);
double atanh(double x);

MT-LEVEL
MT-Safe

DESCRIPTION
These functions compute the designated direct and inverse hyperbolic functions for real arguments.

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

SEE ALSO
matherr(3M)

DIAGNOSTICS
In IEEE 754 mode (i.e. the −xlibmeecc compilation option), sinh() and cosh() return ±∞ as appropriate on overflow and raise the overflow exception; acosh() returns a NaN and raises the invalid operation exception if its argument is less than 1; atanh() returns a NaN and raises the invalid operation exception if its argument has absolute value greater than 1; atanh(±1) returns ±∞ and raises the division by zero exception.
NAME    hypot – Euclidean distance

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

MT-LEVEL    MT-Safe

DESCRIPTION    hypot(x,y) returns
                \( \sqrt{x^2 + y^2} \),
                taking precautions against unwarranted IEEE exceptions. On IEEE overflow, hypot(x,y) may also set errno and call matherr(3M). hypot(±∞,y) is \(+∞\) for any y, even a NaN, and is exceptional only for a signaling NaN.

hypot(x,y) and atan2(y,x) (see trig(3M)) convert rectangular coordinates (x,y) to polar (r,θ); hypot(x,y) computes r, the modulus or radius.

SEE ALSO    matherr(3M), trig(3M)
NAME iconv – code conversion function

SYNOPSIS

```c
#include <iconv.h>
size_t iconv(iconv_t cd, const char **inbuf, size_t *inbytesleft, char **outbuf,
size_t *outbytesleft);
```

MT-LEVEL MT-Safe

DESCRIPTION

The `iconv()` function converts the sequence of characters from one codeset, in the array specified by `inbuf`, into a sequence of corresponding characters in another codeset, in the array specified by `outbuf`. The codesets 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 codeset, conversion stops after the previous successfully converted character. If the input buffer ends with an incomplete character or shift sequence, conversion stops after the previous successfully converted bytes. If the output buffer is not large enough to hold the entire converted input, conversion stops just prior to the input bytes that would cause the output buffer to overflow. The variable pointed to by `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 codeset, `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 codeset.
- **E2BIG**: Input conversion stopped due to lack of space in the output buffer.
- **EINVAL**: Input conversion stopped due to an incomplete character or shift sequence at the end of the input buffer.

The `iconv()` function may fail if:

- **EBADF**: The `cd` argument is not a valid open conversion descriptor.

FILES

`/usr/lib/iconv/*.so` conversion modules

SEE ALSO

`iconv(1), iconv_open(3), iconv_close(3), iconv(5)`
NAME  iconv_close – code conversion deallocation function

SYNOPSIS  
```
#include <iconv.h>
int iconv_close(iconv_t cd);
```

MT-LEVEL  MT-Safe

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.
```

SEE ALSO  
`iconv(3), iconv_open(3)`
NAME  iconv_open – code conversion allocation function

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

iconv_t iconv_open(const char *tocode, const char *fromcode);
```

MT-LEVEL  MT-Safe

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()` 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.

SEE ALSO  `iconv(3)`, `iconv_close(3)`, `malloc(3C)`

NOTES  `iconv_open()` uses `malloc(3C)` to allocate space for internal buffer areas. `iconv_open()` may fail if there is insufficient storage space to accommodate these buffers.

Portable applications must assume that conversion descriptors are not valid after a call to one of the `exec` functions.
NAME  ieee_functions, ilogb, isnan, copysign, fabs, fmod, nextafter, remainder, scalbn – appendix and related miscellaneous functions for IEEE arithmetic

SYNOPSIS  cc [flag ...] file ... -lm [library ...]
#include <math.h>
int ilogb(double x);
int isnan(double x);
double copysign(double x, double y);
double fabs(double x);
double fmod(double x, double y);
double nextafter(double x, double y);
double remainder(double x, double y);
double scalbn(double x, int n);

MT-LEVEL  MT-Safe

DESCRIPTION  Most of these functions provide capabilities required by ANSI/IEEE Std 754-1985 or suggested in its appendix.

ilogb(x) returns the unbiased exponent of x in integer format. ilogb(±∞) = MAXINT and ilogb(0) = -MAXINT; <values.h> defines MAXINT as the largest int. ilogb(x) never generates an exception. When x is subnormal, ilogb(x) returns an exponent computed as if x were first normalized.

isnan(x) returns 1 if x is NaN; otherwise it returns 0.
copysign(x,y) returns a value with the magnitude of x and with the sign bit of y.
fabs(x) returns the absolute value of x.

nextafter(x,y) returns the next machine representable number from x in the direction y.
remainder(x,y) and fmod(x,y) return a remainder of x with respect to y; that is, the result r is one of the numbers that differ from x by an integral multiple of y. Thus (x−r)/y is an integral value, even though it might exceed MAXINT, the largest int defined in <values.h> if it were explicitly computed as an int. Both functions return one of the two such r smallest in magnitude. remainder(x,y) is the operation specified in ANSI/IEEE Std 754-1985; the result of fmod(x,y) may differ from remainder’s result by ±y. The magnitude of remainder’s result cannot exceed half that of y; its sign might not agree with either x or y. The magnitude of fmod’s result is less than that of y; its sign agrees with that of x. Neither function can generate an exception as long as both arguments are normal or subnormal.

remainder(x,0), fmod(x,0), remainder(∞,y), and fmod(∞,y) are invalid operations; in IEEE 754 mode (i.e. the −xlibmieee cc compilation option), a NaN is returned.
**scalbn** returns \( x \times 2^{\ast n} \) computed by exponent manipulation rather than by actually performing an exponentiation or a multiplication. Thus

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

for every \( x \) except 0, \( \infty \), and \( \text{NaN} \).

**RETURN VALUES**

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

**SEE ALSO**

**matherr**(3M)
NAME  ieee_test, logb, scalb, significand – IEEE test functions for verifying standard compliance

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

MT-LEVEL  MT-Safe

DESCRIPTION  These functions allow users to verify compliance to ANSI/IEEE Std 754-1985 by running certain test vectors distributed by the University of California.

\texttt{logb(x)} returns the unbiased exponent of \( x \) in floating-point format, for exercising the \texttt{logb(L)} test vector. \( \texttt{logb(\pm\infty)} = +\infty; \texttt{logb(0)} = -\infty \) with a division by 0 exception.

\texttt{scalb}(x, (\texttt{double})n) returns \( x \times 2^{\times n} \) computed by exponent manipulation rather than by actually performing an exponentiation or a multiplication, for exercising the \texttt{scalb(S)} test vector. Thus

\[ 0 \leq \texttt{scalb}(|x|, -\texttt{logb}(x)) < 2 \]

for every \( x \) except 0, \( \infty \) and \( NaN \). \texttt{scalb(x,y)} is not defined when \( y \) is not an integral value.

If \( x \) equals \( \texttt{sig} \times 2^{\times n} \) with \( 1 \leq \texttt{sig} < 2 \), then \texttt{significand}(x) returns \( \texttt{sig} \) for exercising the fraction-part(F) test vector. \texttt{significand}(x) is not defined when \( x \) is either 0, \( \pm\infty \) or \( NaN \).

RETURN VALUES  For exceptional cases, \texttt{matherr}(3M) tabulates the values to be returned as dictated by various Standards.

SEE ALSO  \texttt{matherr}(3M)

3M-586  modified 4 Mar 1994
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  string(3C), bstring(3C), malloc(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.

modified 3 March 1995
inet (3N)  Network Functions  SunOS 5.5

NAME    inet, inet_addr, inet_network, inet_makeaddr, inet_lnaof, inet_netof, inet_ntoa – Internet address manipulation

SYNOPSIS  cc [flag ...] file ... -lnsl [ library ... ]
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

unsigned long inet_addr(const char *cp);
unsigned long inet_network(const char *cp);
struct in_addr inet_makeaddr(const int net, const int lna);
int inet_lnaof(const struct in_addr in);
int inet_netof(const struct in_addr in);
char *inet_ntoa(const struct in_addr in);

MT-LEVEL  Safe

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” described below.

All Internet addresses are returned in network order (bytes ordered from left to right). All network numbers and local address parts are returned as machine format integer values.

INTERNET ADDRESSES  Values specified using the ‘.’ 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”.

3N-588  modified 28 Jan 1994
When a two part address is supplied, the last part is interpreted as a 24-bit quantity and placed in the right most three bytes of the network address. This makes the two part address format convenient for specifying Class A network addresses as "net.host".

When only one part is given, the value is stored directly in the network address without any byte rearrangement.

All numbers supplied as "parts" in a "." notation may be decimal, octal, or hexadecimal, as specified in the C language (that is, a leading 0x or 0X implies hexadecimal; otherwise, a leading 0 implies octal; otherwise, the number is interpreted as decimal).

**RETURN VALUES**

The value −1 is returned by `inet_addr()` and `inet_network()` for malformed requests.

The routines `inet_netof()` and `inet_lnaof()` break apart Internet host addresses, returning the network number and local network address part, respectively.

The routine `inet_ntoa()` returns a pointer to a string in the base 256 notation "d.d.d.d" described below.

**SEE ALSO**

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

**NOTES**

The return value from `inet_ntoa()` points to a buffer which is overwritten on each call. This buffer is implemented as thread-specific data in multithreaded applications.

**BUGS**

The problem of host byte ordering versus network byte ordering is confusing. A simple way to specify Class C network addresses in a manner similar to that for Class B and Class A is needed.

Modified 28 Jan 1994
NAME
initgroups – initialize the supplementary group access list

SYNOPSIS
#include <grp.h>
#include <sys/types.h>
int initgroups(const char *name, gid_t basegid);

MT-LEVEL
Unsafe

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 supple-
mentary 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.

SEE ALSO
getgroups(2), getgrnam(3C)
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);

MT-LEVEL
Unsafe

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.
NAME  isastream – test a file descriptor

SYNOPSIS  
#include <stropts.h>
int isastream(int fildes);

MT-LEVEL  MT-Safe

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.

SEE ALSO  streamio(7I)
STREAMS Programming Guide
isencrypt — determine whether a buffer of characters is encrypted

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, isencrypt( ) 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.

SEE ALSO
setlocale(3C)

NOTES
When compiling multi-thread applications, the _REENTRANT flag must be defined on the compile line. This flag should only be used in multi-thread applications.
NAME
isnan, isnand, isnanf, finite, fpclass, unordered – determine type of floating-point number

SYNOPSIS
#include <ieeefp.h>
int isnand(double dsrc);
int isnanf(float fsrc);
int finite(double dsrc);
fpclass_t fpclass(double dsrc);
int unordered(double dsrcl, double dsrch);
#include <math.h>
int isnan(double dsrc);

MT-LEVEL
MT-Safe

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 dsrc belongs to. The 10 possible classes are as follows:

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP_SNAN</td>
<td>signaling NaN</td>
</tr>
<tr>
<td>FP_QNAN</td>
<td>quiet NaN</td>
</tr>
<tr>
<td>FP_NINF</td>
<td>negative infinity</td>
</tr>
<tr>
<td>FP_PINF</td>
<td>positive infinity</td>
</tr>
<tr>
<td>FP_NDENORM</td>
<td>negative denormalized non-zero</td>
</tr>
<tr>
<td>FP_PDENORM</td>
<td>positive denormalized non-zero</td>
</tr>
<tr>
<td>FP_NZERO</td>
<td>negative zero</td>
</tr>
<tr>
<td>FP_PZERO</td>
<td>positive zero</td>
</tr>
<tr>
<td>FP_NNORM</td>
<td>negative normalized non-zero</td>
</tr>
<tr>
<td>FP_PNORM</td>
<td>positive normalized non-zero</td>
</tr>
</tbody>
</table>

None of these routines generate any exception, even for signaling NaNs.

RETURN VALUES
isnan(), isnand(), and isnanf() return true (1) if the argument dsrc or fsrc is a NaN; otherwise they return false (0).
finite() returns true (1) if the argument dsrc 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.

SEE ALSO
fpgetround(3C)
**NAME**
iswalpha, iswupper, iswlower, iswdigit, iswxdigit, iswalnum, iswspace, iswpunct, iswprint, iswcntrl, iswascii, iswgraph, isphonogram, isideogram, isenglish, isnumber, isspecial

**SYNOPSIS**
`cc [ flag ... ] file ... -lw [ library ... ]`

```
#include <widec.h>
#include <wctype.h>

int iswalpha(wint_t c);
```

**MT-LEVEL**
MT-Safe with exceptions

**WIDE CHARACTER CLASSIFICATION**
These functions classify Process Code characters (wchar_t) from the primary and supplementary codesets by table lookup. Each is a predicate returning nonzero for true, zero for false. The lookup table, generated by wchrtbl(), contains values for both ASCII and supplementary codesets.

- `iswalpha(c)`
  - c is a Latin alphabet Process Code character, from either the primary or supplementary codesets.

- `iswupper(c)`
  - c is an upper case Latin alphabet Process Code character, from either the primary or supplementary codesets.

- `iswlower(c)`
  - c is a lower case Latin alphabet Process Code character, from either the primary or supplementary codesets.

- `iswdigit(c)`
  - c is a Process Code digit [0-9], from either the primary or supplementary codesets.

- `iswxdigit(c)`
  - c is an ASCII hexadecimal Process Code digit [0-9], [A-F], or [a-f].

- `iswalnum(c)`
  - c is a Process Code Latin letter or a digit, from either the primary or supplementary codesets.

- `iswspace(c)`
  - c is a Process Code space, tab, carriage return, newline, vertical tab, or formfeed, from either the primary or supplementary codesets.

- `iswpunct(c)`
  - c is an ASCII Process Code punctuation character (neither control nor alphanumeric).

- `iswprint(c)`
  - c is a Process Code printing character, from either the primary or supplementary codesets. It includes the space character.

- `iswgraph(c)`
  - c is a Process Code visible graphic character, from the primary or supplementary codesets. It does not include the space character.

- `iswcntrl(c)`
  - c is a Process Code ASCII delete character or ordinary control character, or a control character from a supplementary codeset.

- `iswascii(c)`
  - c is a Process Code ASCII character.

- `isphonogram(c)`
  - c is a Process Code phonetic language character from a supplementary codeset.

Modified 20 May 1994
iswalpha(3I)  Wide Character Functions

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

SEE ALSO  setlocale(3C), stdio(3S), wconv(3I), ascii(5)

NOTES  iswalpha(), iswupper(), iswlower(), iswdigit(), iswxdigit(), iswalnum(), iswspace(),
iswpunct(), iswprint(), iswcntrl(), iswascii(), iswgraph(), isphonogram(), isideogram(),
isenglish(), isnumber() and isspecial() can be used safely in a multi-thread application, as long as setlocale(3C) is not being called to change the locale.
NAME iswctype – test character for specified class

SYNOPSIS cc [ flag ... ] file ... -lw [ library ... ]
#include <wchar.h>
int iswctype(wint_t wc, wchar_t charclass);

MT-LEVEL MT-Safe

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(3I) or charclass is invalidated by a subsequent call to setlocale(3C) that has affected category LC_CTYPE), the result is implementation-dependent.

RETURN VALUE iswctype() returns 0 for FALSE and non-zero for TRUE.

USAGE The twelve strings — "alnum", "alpha", "blank", "cntrl", "digit", "graph", "lower", "print", "punct", "space", "upper" and "xdigit" — are reserved for the standard character classes. In the table below, the functions in the left column are equivalent to the functions in the right column.

| 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.

SEE ALSO iswalnum(3I), iswalpha(3I), iswcntrl(3I), iswdigit(3I), iswgraph(3I), iswlower(3I), iswprint(3I), iswpunct(3I), iswspace(3I), iswupper(3I), iswxdigit(3I), setlocale(3C), wctype(3I)

modified 16 May 1994
NAME
kerberos, krb_mk_req, krb_rd_req, krb_kntoln, krb_set_key, krb_get_cred, krb_mk_safe,
krb_rd_safe, krb_mk_err, krb_rd_err – Kerberos authentication library

SYNOPSIS
c [ 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 u_long checksum);

int krb_rd_req(const KTEXT authent, const char *service, char *instance,
const u_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);

MT-LEVEL Unsafe

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 check-
sum. 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 authenti-
cator 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

SEE ALSO kerberos(1), kerberos_rpc(3N), krb.realmofhost(3N), krb.sendauth(3N), krb.set_tkt_string(3N), krb.conf(4), krb.realms(4)

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

BUGS The caller of krb_rd_req() and krb_rd_safe() must check time order and for replay attempts.

AUTHORS Clifford Neuman, MIT Project Athena
Steve Miller, MIT Project Athena/Digital Equipment Corporation

RESTRICTIONS COPYRIGHT 1985,1986,1989 Massachusetts Institute of Technology
NAME

kerberos_rpc, authkerb_getucred, authkerb_seccreate, svc_kerb_reg – library routines for remote procedure calls using Kerberos authentication

MT-LEVEL

Unsafe

DESCRIPTION

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.

```
c
#include <rpc/rpc.h>
#include <sys/types.h>

int authkerb_getucred(const struct svc_rqst *rqst, uid_t *uidp, gid_t *gidp,
                      short *gidlenp, int gidlist[NGROUPS]);
```

`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 param-
eter 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 resynchroniza-
tions because of clock drift.

The fifth parameter, timehost, is optional. If it is NULL, then the authentication
system will assume that the local clock is always in sync with the timehost clock,
and will not attempt resynchronizations. If a timehost is supplied, however, then
the system will consult with the remote time service whenever resynchronization
is required. This parameter is usually the name of the host on which the server is
running.

The final parameter status is also optional. If status is supplied, then it will be
used to return a Kerberos error status codes if an error occurs. If status is NULL,
then no detailed error codes will be returned.

If authkerb_seccreate() fails, it returns NULL.

int svc_kerb_reg(const SVCXPRT *xprt, const char *name, const char *inst,
    const char *realm);

svc_kerb_reg() performs registration tasks in the server which are required
before AUTH_KERB requests can be processed. xprt is the RPC transport to which
this information is to be associated. If xprt is NULL then this registration will be
effective for any requests arriving on transports that have not been specifically
registered.

The other parameters describe the Kerberos principal identity that this server will
take on. This must be the same identity that the clients will use when requesting
Kerberos tickets for authentication. name is the principal name of the service and
must be provided. inst is the instance. This parameter may be NULL to specify
the NULL instance of the service. Most common would be for inst to be "*" which
allows the Kerberos library to determine the correct instance to use, such as the
hostname that the service is running on. realm is the Kerberos realm name to use
in validating tickets. If it is NULL, then the local default realm will be used.
svc_kerb_reg() should generally be called immediately before svc_run(). It returns 0 if it succeeds, and −1 if it fails.

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

NOTES  These interfaces are unsafe in multithreaded applications. Unsafe interfaces should be called only from the main thread.
NAME    killpg – send signal to a process group

SYNOPSIS  #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)
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);

MT-LEVEL Unsafe

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

3N-606 modified 22 Jan 1993
equal to 3) comes from the map.

krb_get_admhst() fills in host with the hostname of the nth 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 nth realm of the local host, as specified in the
configuration file. realm should be at least REALM_SZ (from <kerberos/krb.h>) characters
long. The return value is either KSUCCESS or KFAILURE.

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

FILES /etc/krb.realms translation file for host-to-realm mapping.
/etc/krb.conf local realm-name and realm/server configuration file.

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

BUGS The current convention for instance names is too limited; the full domain name should be
used.

krb_get_lrealm() currently only supports n equal to 1. It should really consult the user’s
ticket cache to determine the user’s current realm, rather than consulting a file on the
host.
krb_sendauth (3N)  Network Functions  SunOS 5.5

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_DATA *msg_data, CREDENTIALS *cred, Key_schedule schedule,
const struct sockaddr_in *laddr, const struct sockaddr_in *faddr,
const char *version);

int krb_recvauth(const long options, const int fd, KTEXT ktext, const char *service,
char *inst, const struct sockaddr_in *faddr, const struct sockaddr_in *laddr,
AUTH_DAT *auth_data, const char *filename, Key_version schedule,
char *version);

int krb_net_write(const int fd, const char *buf, const int len);
int krb_net_read(const int fd, char *buf, const int len);

MT-LEVEL
Unsafe

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.

modified 22 Jan 1993
The `filename` argument specifies the filename which the service program should use to obtain its service key. `krb_recvauth()` passes `filename` to the `krb_rd_req()` function, see `kerberos(3N)`. If you set this argument to "", `krb_rd_req()` looks for the service key in the file `/etc/srvtab`.

If the client and server are performing mutual authentication, the `schedule` argument should point to an allocated `Key_schedule`. Otherwise it is ignored and may be NULL.

The `version` argument should point to a character array of at least `KSEND_VNO_LEN` characters. It is filled in with the version string passed by the client to `krb_sendauth()`.

### krb_net_write() and krb_net_read()

The `krb_net_write()` function emulates the `write(2)` system call, but guarantees that all data specified is written to `fd` before returning, unless an error condition occurs.

The `krb_net_read()` function emulates the `read(2)` system call, but guarantees that the requested amount of data is read from `fd` before returning, unless an error condition occurs.

### SEE ALSO

`read(2)`, `write(2)`, `kerberos(3N)`, `kerberos_rpc(3N)`, `krb_realmofhost(3N)`

### NOTES

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

### BUGS

`krb_sendauth()`, `krb_recvauth()`, `krb_net_write()`, and `krb_net_read()` will not work properly on sockets set to non-blocking I/O mode.

### AUTHOR

John T. Kohl, MIT Project Athena

### RESTRICTIONS

Copyright 1988, Massachusetts Institute of Technology. For copying and distribution information, please see the header `<kerberos/mit-copyright.h>`.
NAME  krb_set_tkt_string – set Kerberos ticket cache file name

SYNOPSIS  cc [ flag ... ] file ... -lkrb [ library ... ]
#include <kerberos/krb.h>
void krb_set_tkt_string(const char *filename);

MT-LEVEL  Unsafe

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 KRBTOKFILE is set. uid denotes the user's uid, in decimal.

SEE ALSO  kerberos(3N)

NOTES  This interface is unsafe in multithreaded applications. Unsafe interfaces should be called only from the main thread.
NAME | kstat – kernel statistics facility

DESCRIPTION | The kstat facility is a general-purpose mechanism for providing kernel statistics to users.

The kstat model | The kernel maintains a linked list of statistics structures, or kstats. Each kstat has a common header section and a type-specific data section. The header section is defined by the kstat_t structure:

```c
typedef long kid_t; /* unique kstat id */
typedef struct kstat {
    /* Fields relevant to both kernel and user */
    hrtime_t ks_crtime; /* creation time */
    struct kstat *ks_next; /* kstat chain linkage */
    kid_t ks_kid; /* unique kstat ID */
    char ks_module[KSTAT_STRLEN]; /* module name */
    uchar_t ks_resv; /* reserved */
    int ks_instance; /* module's instance */
    char ks_name[KSTAT_STRLEN]; /* kstat name */
    uchar_t ks_type; /* kstat data type */
    char ks_class[KSTAT_STRLEN]; /* kstat class */
    uchar_t ks_flags; /* kstat flags */
    void *ks_data; /* kstat type-specific data */
    ulong_t ks_ndata; /* # of data records */
    ulong_t ks_data_size; /* size of kstat data section */
    hrtime_t ks_snapt ime; /* time of last data snapshot */
    /*
     * Fields relevant to kernel only
     */
    int (*ks_update)(struct kstat *, int);
    void *ks_private;
    int (*ks_snapshot)(struct kstat *, void *, int);
    void *ks_lock;
} kstat_t;
```

The fields that are of significance to the user are:

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

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

**ks_next**

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

**ks_kid**

A unique identifier for the kstat.

**ks_module, ks_instance**

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

**ks_name**

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

**ks_type**

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

**ks_class**

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

**ks_data, ks_ndata, ks_data_size**

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

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

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

**ks_snaptime**

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

\[
\text{rate} = \frac{\text{new\_count} - \text{old\_count}}{\text{new\_snaptime} - \text{old\_snaptime}};
\]

**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 (e.g., KSTAT_TYPE_IO is type 3 -- see above).
Raw kstat

KSTAT_TYPE_RAW: raw data

The "raw" kstat type is just treated as an array of bytes. This is generally used to export well-known structures, like `sysinfo`.

Name=value kstat

KSTAT_TYPE_NAMED: A list of arbitrary name=value statistics.

typedef struct kstat_named {
    char name[KSTAT_STRLEN]; /* name of counter */
    uchar_t data_type; /* data type */
    union {
        char c[16]; /* enough for 128-bit ints */
        long l;
        ulong_t ul;
        longlong_t ll;
        u_longlong_t ull;
        float f;
        double d;
    } value; /* value of counter */
} kstat_named_t;

#define KSTAT_DATA_CHAR 0
#define KSTAT_DATA_LONG 1
#define KSTAT_DATA_ULONG 2
#define KSTAT_DATA_LONGLONG 3
#define KSTAT_DATA_ULONGLONG 4
#define KSTAT_DATA_FLOAT 5
#define KSTAT_DATA_DOUBLE 6

Interrupt kstat

KSTAT_TYPE_INTR: Interrupt statistics.

An interrupt is a hard interrupt (sourced from the hardware device itself), a soft interrupt (induced by the system via the use of some system interrupt source), a watchdog interrupt (induced by a periodic timer call), spurious (an interrupt entry point was entered but there was no interrupt to service), or multiple service (an interrupt was detected and serviced just prior to returning from any of the other types).

#define KSTAT_INTR_HARD 0
#define KSTAT_INTR_SOFT 1
#define KSTAT_INTR_WATCHDOG 2
#define KSTAT_INTR_SPURIOUS 3
#define KSTAT_INTR_MULTSVC 4
#define KSTAT_NUM_INTRS 5

typedef struct kstat_intr {
    ulong_t intrs[KSTAT_NUM_INTRS]; /* interrupt counters */
} kstat_intr_t;
Event timer kstat

KSTAT_TYPE_TIMER: Event timer statistics.

These provide basic counting and timing information for any type of event.

typedef struct kstat_timer {
    char name[KSTAT_STRLEN]; /* event name */
    uchar_t resv; /* reserved */
    u_longlong_t num_events; /* number of events */
    hrtime_t elapsed_time; /* cumulative elapsed time */
    hrtime_t min_time; /* shortest event duration */
    hrtime_t max_time; /* longest event duration */
    hrtime_t start_time; /* previous event start time */
    hrtime_t stop_time; /* previous event stop time */
} kstat_timer_t;

I/O kstat

KSTAT_TYPE_IO: I/O statistics.

typedef struct kstat_io {
    /*
     * Basic counters.
     */
    u_longlong_t nread; /* number of bytes read */
    u_longlong_t nwritten; /* number of bytes written */
    ulong_t reads; /* number of read operations */
    ulong_t writes; /* number of write operations */
    /*
     * Accumulated time and queue length statistics.
     *
     * Time statistics are kept as a running sum of "active" time.
     * Queue length statistics are kept as a running sum of the
     * product of queue length and elapsed time at that length --
     * i.e., a Riemann sum for queue length integrated against time.
     *
     * Queue Length
     *=
     |__________|
     8   | i4 |
     |     |
     Queue Length
     6
     4   | i2 |
     |     |
     2   | i3 |
     |     |
     | i1 |
     Time-> t1 t2 t3 t4
     * At each change of state (entry or exit from the queue),

modified 26 May 1994
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.

```
hrttime_t wtime; /* cumulative wait (pre-service) time */
hrttime_t wlenctime; /* cumulative wait length*time product */
hrttime_t wlastupdate; /* last time wait queue changed */
hrttime_t rtime; /* cumulative run (service) time */
hrttime_t rlentime; /* cumulative run length*time product */
hrttime_t rlastupdate; /* last time run queue changed */
ulong_t wcnt; /* count of elements in wait state */
ulong_t rcnt; /* count of elements in run state */
```
**kc_chain** points to your copy of the kstat chain. You typically walk the chain to find and process a certain kind of kstat. For example, to display all I/O kstats:

```
kstat_ctl_t *kc;
kstat_t *ksp;
kstat_io_t kio;

kc = kstat_open();
for (ksp = kc->kc_chain; ksp != NULL; ksp = ksp->ks_next) {
    if (ksp->ks_type == KSTAT_TYPE_IO) {
        kstat_read(kc, ksp, &kio);
        my_io_display(kio);
    }
}
```

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

**FILES**
/dev/kstat
/usr/include/kstat.h
/usr/include/sys/kstat.h

**SEE ALSO**
ioctl(2), gethrtime(3C), kstat_chain_update(3K), kstat_close(3K), kstat_data_lookup(3K), kstat_lookup(3K), kstat_open(3K), kstat_read(3K), kstat_write(3K)
NAME  
kstat_chain_update – update the kstat header chain

SYNOPSIS  
cc [ 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)
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 correspond-
              ing 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);

MT-LEVEL  
Unsafe

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.

3K-622

modified 22 Jan 1993
SEE ALSO  
\text{\texttt{kvm\_nextproc}(3K), \texttt{kvm\_open}(3K), \texttt{kvm\_read}(3K), \texttt{malloc}(3C)}

NOTES  
If \texttt{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
#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);

MT-LEVEL Unsafe

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() rewinds the process list, enabling kvm_nextproc() to rescan from the beginning of the system process table. kvm_setproc() will always flush the process structure cache, allowing an application to re-scan the process table of a running system.

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.

SEE ALSO kvm_getu(3K), kvm_open(3K), kvm_read(3K)
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);

MT-LEVEL    Unsafe

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-
             minated 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.

SEE ALSO   nlist(3B), nlist(3E), kvm_open(3K), kvm_read(3K)
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);

MT-LEVEL
Unsafe

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

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

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

modified 11 Apr 1994
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.

3K-628 modified 7 Jul 1995
SEE ALSO

kvm_getu(3K), kvm_nlist(3K), kvm_open(3K)
NAME lckpwdf, ulckpwdf – manipulate shadow password database lock file

SYNOPSIS
#include <shadow.h>
int lckpwdf(void);
int ulckpwdf(void);

MT-LEVEL MT-Safe

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

SEE ALSO getpwnam(3C), getspnam(3C)

NOTES These routines are for internal use only; compatibility is not guaranteed.

3C-630 modified 30 Aug 1991
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 */);

MT-LEVEL
MT-safe

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. Optionnally, lfmt() will display the output on the console, with a date and time stamp.
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, lfmt() will attempt to retrieve the message from the C locale. If this second retrieval fails, lfmt() uses the <defmsg> part of the format argument.
If <catalog> is omitted, lfmt() 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>.
lfmt() will output Message not found!!\n as format string if <catalog> is not a valid catalog name, if no catalog is specified (either explicitely 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.

modified 08 December 1993
The flags are composed of several groups, and can take the following values (one from each group): Output format control

**MM_NOSTD**  
Do not use the standard message format, interpret `format` as a `printf()` format. Only catalog access control flags, console display control and logging information should be specified if `MM_NOSTD` is used; all other flags will be ignored.

**MM_STD**  
Output using the standard message format (default, value 0).

*Catalog access control*

**MM_NOGET**  
Do not retrieve a localized version of `format`. In this case, only the `<defmsg>` part of the `format` is specified.

**MM_GET**  
Retrieve a localized version of `format`, from the `<catalog>`, using `<msgid>` as the index and `<defmsg>` as the default message (default, value 0).

*Severity (standard message format only)*

**MM_HALT**  
generates a localized version of **4HALT**, but does not halt the machine.

**MM_ERROR**  
generates a localized version of **ERROR** (default, value 0).

**MM_WARNING**  
generates a localized version of **WARNING**.

**MM_INFO**  
generates a localized version of **INFO**.

Additional severities can be defined. Add-on severities can be defined with number-string pairs with numeric values from the range [5-255], using `addsev()`. The numeric value ORed with other flags will generate the specified severity.

If the severity is not defined, `lfmt()` used the string `SEV=N` where `N` is replaced by the integer severity value passed in flags.

Multiple severities passed inf flags will not be detected as an error. Any combination of severities will be summed and the numeric value will cause the display of either a severity string (if defined) or the string `SEV=N` (if undefined).

*Action*

**MM_ACTION**  
specifies an action message. Any severity value is superseded and replaced by a localized version of **TO FIX**.

*Console display control*

**MM_CONSOLE**  
display the message to the console in addition to the specified stream.

**MM_NOCONSOLE**  
do not display the message to the console in addition to the specified stream (default, value 0).
Logging information

Major classification
Identifies the source of the condition. Identifiers are:

- **MM_HARD** (hardware),
- **MM_SOFT** (software), and
- **MM_FIRM** (firmware).

Message source subclassification
Identifies the type of software in which the problem is spotted. Identifiers are:

- **MM_APPL** (application),
- **MM_UTIL** (utility),
- **MM_OPSYS** (operating system).

### STANDARD ERROR MESSAGE FORMAT

`lfmt()` displays error messages in the following format:

```
label: severity: text
```

If no label was defined by a call to `setlabel()`, the message is displayed in the format:

```
severity: text
```

If `lfmt()` is called twice to display an error message and a helpful action or recovery message, the output can look like:

```
label: severity: text
label: TO FIX: text
```

### RETURN VALUE

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

- `-1` write error to stream.
- `-2` cannot log and/or display at console.

### EXAMPLES

**Example 1:**

```c
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:**

```c
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**

modified 08 December 1993

3C-633
NOTES
Since `lfmt()` uses `gettext(3C)`, it is recommended that `lfmt()` not be used.

SEE ALSO `addsev(3C)`, `gettext(3C)`, `pfmt(3C)`, `printf(3S)`,
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);

MT-LEVEL
See the NOTES section of this page.

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.

SEE ALSO
matherr(3M)

NOTES
Although lgamma_r() is not mentioned by POSIX.4a Draft 6, it was added to complete the functionality provided by similar thread-safe functions. This interface is subject to change to be compatible with the "spirit" of POSIX.4a when it is approved as a standard. When compiling multi-thread applications, the _REENTRANT flag must be defined on the compile line. This flag should only be used in multi-thread applications. lgamma() is unsafe in multithreaded applications. lgamma_r() should be used instead.

modified 18 Jul 1994

3M-635
NAME
libthread_db, td_log, td_ta_new, td_ta_delete, td_init, td_ta_get_ph, td_ta_get_nththreads,
td_ta_tsd_iter, td_ta_thr_iter, td_thr_validate, td_thr_tsd, td_thr_get_info,
td_thr_getfregs, td_thr_getxregsize, td_thr_getregs, td_thr_sigsetmask, td_thr_setprio,
td_thr_setsigpending, td_thr_setfregs, td_thr_setxregs, td_ta_map_id2thr,
td_ta_map_lwp2thr, td_thr_getgregs, td_thr_setgregs – interface to libthread threads
information

SYNOPSIS
cc [ flag ...] file ... /lib/libthread_db.so.1 [ library ... ]
#include <proc_service.h>
#include <thread_db.h>
void td_log(const int on_off);

#err_e td_ta_new(const struct ps_prochandle *ph, td_thragent_t **ta_p);
#err_e td_ta_delete(const td_thragent_t *ta_p);
#err_e td_init();
#err_e td_ta_get_ph(const td_thragent_t *ta_p, struct ps_prochandle **ph_p);
#err_e td_ta_get_nththreads(const td_thragent_t *ta_p, int *nthread_p);
#err_e td_ta_tsd_iter(const td_thragent_t *ta_p, td_key_iter_f *cb, void *cbdata_p);
#err_e td_ta_thr_iter(const td_thragent_t *ta_p, td_thr_iter_f *cb, void *cbdata_p,
   td_thr_state_e state, int ti_pri, sigset_t *ti_sigmask_p, unsigned ti_user_flags);
#err_e td_thr_validate(const td_thrhandle_t *th_p);
#err_e td_thr_tsd(const td_thrhandle_t *th_p, const thread_key_t key,
   void **data_p);
#err_e td_thr_get_info(const td_thrhandle_t *th_p, td_thrinfo_t *ti_p);
#err_e td_thr_getfregs(const td_thrhandle_t *th_p, prfpregset_t *fregset);
#err_e td_thr_getxregsize(const td_thrhandle_t *th_p, int *xregsize);
#err_e td_thr_setfregs(const td_thrhandle_t *th_p, const prfpregset_t *fregset);
#err_e td_thr_setxregs(const td_thrhandle_t *th_p, const caddr_t *xregset);
#err_e td_thr_sigsetmask(const td_thrhandle_t *th_p, const sigset_t *ti_sigmask);
#err_e td_thr_setprio(const td_thrhandle_t *th_p, const int ti_pri);
#err_e td_thr_setsigpending(const td_thrhandle_t *th_p,
   const uchar_t ti_pending_flag, const sigset_t ti_pending);
#err_e td_thr_setfregs(const td_thrhandle_t *th_p, const prfpregset_t *fregset);
#err_e td_thr_setxregs(const td_thrhandle_t *th_p, const caddr_t *xregset);
#err_e td_ta_map_id2thr(const td_thragent_t *ta_p, thread_t tid,
   td_thrhandle_t *th_p);
#err_e td_ta_map_lwp2thr(const td_thragent_t *ta_p, lwpid_t lwpid,
   td_thrhandle_t *th_p);

3T-636 modified 30 Aug 1995
libthread_db is a library of functions for accessing information about threads in a process under inspection (PUI). The typical use for this library is by a debugger that requires information about the threads in a process. To some extent it can also be used for self inspection of threads within the same process. libthread_db utilizes functions provided by the user for interrogating a process (i.e., reading, writing, stopping, continuing, etc. a process). The prototypes for these user provided functions, referred to as the proc_service functions are described in <proc_service.h>. libthread_db should be accessed by an explicit version of the library (e.g., libthread_db.so.1) to ensure the correct version is used.

Support Level
All interfaces in libthread_db are UNCOMMITTED interfaces and are subject to change in future releases.

FUNCTIONS

**td_log()** turns logging on and off. A function in the proc_service interface is provided by the user for logging execution points in libthread_db. If logging is turned on, this proc_service function is called. The nominal name of the function is **ps_plog, on_off = 0** turns logging off, non-zero turns logging on. proc_service functions called: none.

**td_ta_new()** allocates a thread agent for the given process handle and return a pointer to it. proc_service functions called: **ps_pglobal_lookup**.

**td_ta_delete()** deallocates the thread agent. proc_service functions called: none.

**td_init()** performs initialization for libthread_db interface. proc_service functions called: none

**td_ta_get_ph()** gets the process handle out of a thread agent and returns it. proc_service functions called: none

**td_ta_get_nthreads()** gets the total number of threads in a process. This number includes both user and system threads. proc_service functions called: **ps_pglobal_lookup, ps_pread, ps_pstop, ps_pcontinue**.

**td_ta_tsd_iter()** iterates over the set of global TSD keys. The call back function is called with three arguments, a key, a pointer to a destructor function, and an extra pointer which can be NULL depending on the call back. The call back function is called once for each key. If return value of **cb** is non-zero, terminate iterations. proc_service functions called: **ps_pglobal_lookup, ps_pread, ps_pstop, ps_pcontinue**.

**td_ta_thr_iter()** iterates over all threads. For each thread call the function pointed to by **cb** with a pointer to a thread handle, and a pointer to data which can be NULL. Only call **cb on threads** which match the properties of **state, ti_pri, ti_sigmask_p, and ti_user_flags**. If **cb** returns a non-zero value, iterations terminate. The call back function is defined by user. **td_thr_iter_f** takes a thread handle and **cbdata_p** as a parameter. **state** is the state of threads of interest. A value of **TD_THR_ANY_STATE** from enum **td_thr_state_e** does not restrict iterations by state. **ti_pri** is the lower bound of priorities of threads of interest.
value of `TD_THR_LOWEST_PRIORITY` defined in `thread_db.h` does not restrict iterations by priority. A thread with priority less than `ti_pri` will NOT be passed to the callback function. `ti_sigmask_p` is the signal mask of threads of interest. A value of `TD_SIGNO_MASK` defined in `thread_db.h` does not restrict iterations by signal mask. `ti_user_flags` is the user flags of threads of interest. A value of `TD_THR_ANY_USER_FLAGS` defined in `thread_db.h` does not restrict iterations by user flags.

proc_service functions called: `ps_pdread`, `ps_pstop`, `ps_pcontinue`.

`td_thr_validate()` validates the thread handle. `td_thr_validate()` checks that a thread exists in the thread agent/process that corresponds to thread with handle `*th_p`. Return value: `TD_OK` implies thread handle is valid. `TD_NOTHR` implies thread handle not valid. proc_service functions called: `ps_pdread`, `ps_pstop`, `ps_pcontinue`.

`td_thr_tsd()` gets a thread’s private binding to a given thread specific data (TSD) key (see `thr_getspecific(3T)`). If the thread doesn’t have a binding for a particular key, then NULL is returned. proc_service functions called: `ps_pglobal_lookup`, `ps_pdread`, `ps_pstop`, `ps_pcontinue`.

`td_thr_get_info()` updates the thread information struct. All fields in a thread information structure (`td_thrinfo_t`) will be updated to be consistent with properties of its respective thread. proc_service functions called: `ps_pdread`, `ps_pstop`, `ps_pcontinue`.

`td_thr_getfpregs()` gets the floating point registers for the given thread. proc_service functions called: `ps_pdread`, `ps_pstop`, `ps_pcontinue`, `ps_lgetfpregs`.

`td_thr_getxregsize()` gets the size of the extra register set for the given thread. proc_service functions called: `ps_pdread`, `ps_pstop`, `ps_pcontinue`, `ps_lgetxregsize`.

`td_thr_getxregs()` gets the extra registers for the given thread. proc_service functions called: `ps_pdread`, `ps_pstop`, `ps_pcontinue`, `ps_lgetxregs`.

`td_thr_sigsetmask()` changes a thread’s signal mask to the value specified by `ti_sigmask`. proc_service functions called: `ps_pstop`, `ps_pcontinue`, `ps_pdwrite`.

`td_thr_setprio()` changes a thread’s priority to the value specified by `ti_pri`. proc_service functions called: `ps_pdwrite`.

`td_thr_setsigpending()` changes a thread’s pending signal state to that specified by `ti_pending_flag` and `ti_pending`. A null value for `ti_pending_flag` indicates that there are no pending signals for the thread. proc_service functions called: `ps_pstop`, `ps_pcontinue`, `ps_pdwrite`.

`td_thr_setfpregs()` sets the floating point registers for a given thread. proc_service functions called: `ps_pdread`, `ps_pstop`, `ps_lsetfpregs`, `ps_pcontinue`.

`td_thr_setxregs()` sets the extra registers for the given thread. proc_service functions called: `ps_pcontinue`, `ps_pstop`, `ps_pdread`, `ps_pdwrite`, `ps_lsetxregs`.

`td_ta_map_id2thr()` returns the thread handle corresponding to the given thread identifier tid. proc_service functions called: `ps_pdread`, `ps_pstop`, `ps_pcontinue`.

`td_ta_map_lwp2thr()` returns a thread handle for the given thread agent and lwp id. proc_service functions called: `ps_pdread`, `ps_lgetregs`.

modified 30 Aug 1995
**td_thr_getregs()** gets the general registers for a given thread. For a thread that is currently executing on an LWP, (*td_thr_state_e* TD_THR_ACTIVE), all registers in *regset* will be read for the thread. For a thread not executing on an LWP, only the following registers will be read.

**SPARC**
- %i0-%i7,
- %l0-%l7,
- %g7, %pc, %sp (%o6).

**x86**
- %ebp, %edi, %esi, %ebx, %sp, %pc.

%pc and %sp will be the program counter and stack pointer at the point where the thread will resume execution when it becomes active, (*td_thr_state_e* TD_THR_ACTIVE). proc_service functions called: **ps_pstop**, **ps_pcontinue**, **ps_pdread**, **ps_lgetregs**.

**td_thr_setregs()** sets the general registers for a given thread. For a thread that is currently executing on an LWP, (*td_thr_state_e* TD_THR_ACTIVE), all registers in *regset* will be written for the thread. For a thread not executing on an LWP, only the following registers will be written.

**SPARC**
- %i0-%i7,
- %l0-%l7,
- %pc, %sp (%o6).

**x86**
- %ebp, %edi, %esi, %ebx, %sp, %pc.

%pc and %sp will be the program counter and stack pointer at the point where the thread will resume execution when it becomes active, (*td_thr_state_e* TD_THR_ACTIVE). proc_service functions called: **ps_pstop**, **ps_pcontinue**, **ps_pdread**, **ps_lsetregs**.

**RETURN VALUES**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TD_OK</td>
<td>The call completed successfully.</td>
</tr>
<tr>
<td>TD_ERR</td>
<td>The call failed and no other error code applies.</td>
</tr>
<tr>
<td>TD_NOTHR</td>
<td>The call failed because the thread handle does not correspond to any thread known to libthread_db. The thread may no longer exist or the user may be using a corrupted thread handle.</td>
</tr>
<tr>
<td>TD_NOSV</td>
<td>If the synchronization variable passed as a parameter does not correspond to any synchronization variable known to libthread_db the call will return this value. The synchronization variable may no longer exist or the user may be using a corrupted synchronization handle.</td>
</tr>
<tr>
<td>TD_NOLWP</td>
<td>If there is no LWP that is part of the process that corresponds to the given lwpid the call will fail and return this value. This can occur if the lwpid is corrupted or if the lwp has been returned to the OS.</td>
</tr>
<tr>
<td>TD_BADPH</td>
<td>The call failed because the process handle is invalid. This generally indicates that a NULL values was passed as a process handle.</td>
</tr>
</tbody>
</table>

modified 30 Aug 1995
<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TD_BADTH</td>
<td>The call failed because the thread handle is invalid.</td>
</tr>
<tr>
<td>TD_BADSH</td>
<td>The call failed because the synchronization handle is invalid.</td>
</tr>
<tr>
<td>TD_BADKEY</td>
<td>The call failed because the key for the thread specific data area is invalid.</td>
</tr>
<tr>
<td>TD_NOMSG</td>
<td>If there is not event message information, this value is returned.</td>
</tr>
<tr>
<td>TD_NOFPREGS</td>
<td>If the floating point registers are not available for this thread the call returns this value.</td>
</tr>
<tr>
<td>TD_NOLIBTHREAD</td>
<td>If the applications is not threaded (i.e., not linked with libthread) this value is returned.</td>
</tr>
<tr>
<td>TD_NOEVENT</td>
<td>This value is returned if an address was requested for an event which is not supported.</td>
</tr>
<tr>
<td>TD_NOCAPAB</td>
<td>This value indicates that the requested service is not supported. This generally indicates a limitation with the current library that will be rectified in a future release.</td>
</tr>
<tr>
<td>TD_DBERR</td>
<td>The call failed because an error occurred during a request for service from the user's proc_service interface.</td>
</tr>
<tr>
<td>TD_NOAPPLIC</td>
<td>This value is returned if the requested function does not apply to the given variable.</td>
</tr>
<tr>
<td>TD_NOTSD</td>
<td>The call failed because there is no thread specific data for this thread. A key may exist without an entry for every thread.</td>
</tr>
<tr>
<td>TD_MALLOC</td>
<td>The call failed because a malloc failed.</td>
</tr>
<tr>
<td>TD_PARTIALREG</td>
<td>An operation on a register set was performed on only part of the register set. For example, at some points in the execution of a process not all the general purpose registers may be available.</td>
</tr>
<tr>
<td>TD_NOXREGS</td>
<td>If the extra registers are not available for this thread the call returns this value.</td>
</tr>
</tbody>
</table>

**EXAMPLES**

```c
/* Specify libthread_db.so.1 on link line to access correct version.
 * cc thisfile.c /lib/libthread_db.so.1
 */
#include <stdio.h>
#include <sys/types.h>
#include <thread_db.h>
#include <dlfcn.h>

static int thread_cb( const td_thrhandle_t *th_p, void *s );
struct ps_prochandle {
    pid_t pid;
};
```

modified 30 Aug 1995
struct ps_prochandle ph = {1};
td_thragent_t *ta_p;

/*
 * libthread_db example.
 * Initialize libthread_db
 * Create a thread agent
 * Call thread iterator
 */
int main()
{
    td_err_e td_return;
    /*
    * td_init()
    */
    td_return = td_init();
    if ( td_return != TD_OK ) {
        printf("Initialization error on td_init() call\n");
        return 0;
    }
    /*
    * td_ta_new()
    */
    td_return = td_ta_new(&ph, &ta_p);
    if ( td_return == TD_OK ) {
        /*
        * td_ta_thr_iter()
        */
        (void) td_ta_thr_iter(ta_p, thread_cb, "Import calls test",
            TD_THR_ANY_STATE, TD_THR_LOWEST_PRIORITY,
            TD_SIGNO_MASK, TD_THR_ANY_USER_FLAGS);
    }
    return 0;
}

static int thread_cb( const td_thrhandle_t *th_p, void *s )
/*
 * Description: 
 * Call back function for iterator
 *
 * Input:
 * *th_p - thread handle
 * *s - data
 * */
# Include Files

```c
#include <stdio.h>
#include <sys/types.h>
#include <thread_db.h>
#include <dlfcn.h>
```

## Function Definitions

### `thread_cb` Function

```c
static int thread_cb( const td_thrhandle_t *th_p, void *s )
```

### `struct ps_prochandle`

```c
struct ps_prochandle {
  pid_t pid;
};
```

### `struct ps_prochandle` Example

```c
struct ps_prochandle ph = {1};
```

### `td_thragent_t` Function

```c
td_thragent_t *ta_p;
```

### `td_err_e` Functions

- `init(void)`
- `ta_new(const struct ps_prochandle *ph_p, td_thragent_t **ta_pp)`
- `thr_iter(const td_thragent_t *ta_p, td_thr_iter_f *cb, void *cbdata_p, td_thr_state_e state, int ti_pri, sigset_t *ti_sigmask_p, unsigned ti_user_¯ags)`
- `get_info(const td_thrhandle_t *th_p, td_thrinfo_t *ti_p)`

## Main Function

```c
int main()
```

{ 
    td_err_e td_return;
    void *handle;
    /*
     * Access a specific version of libthread_db.
     */
    handle = dlopen("libthread_db.so.1", RTLD_LAZY);
    if ( !handle ) {
        printf("dlopen error: %s\n", dlerror());
        return 1;
    }
    else {
        init = (td_err_e (*)(void))dlsym(handle, "td_init");
        ta_new = (td_err_e (*)(const struct ps_prochandle *, td_thragent_t **))
            dlsym(handle, "td_ta_new");
        thr_iter = (td_err_e (*)(const td_thragent_t *, td_thr_iter_f *,
            void *, td_thr_state_e, int, sigset_t *, unsigned ))
            dlsym(handle, "td_ta_thr_iter");
        get_info = (td_err_e (*)(const td_thrhandle_t *, td_thrinfo_t *))
            dlsym(handle, "td_thr_get_info");
    }
    /*
     * td_init()
     */
    td_return = (*init)();
    if ( td_return != TD_OK ) {
        printf("Initialization error on td_init() call\n");
        return 0;
    }
    /*
     * td_ta_new()
     */
    td_return = (*ta_new>(&ph, &ta_p);
    if ( td_return == TD_OK ) {
        /*
         * td_ta_thr_iter()
         */
        (void) (*thr_iter)(ta_p, thread_cb, "Import calls test",
            TD_THR_ANY_STATE, TD_THR_LOWEST_PRIORITY,
            TD_SIGNAL_MASK, TD_THR_ANY_USER_FLAGS);
    }
    return 0;
}
static int thread_cb( const td_thrhandle_t *th_p, void *s )
/*
 * Description:
 * Call back function for iterator
 *
 * Input:
 * *th_p - thread handle
 * *s - data
 *
 * Output:
 * none
 *
 */
{
 int return_val = 0;
 td_err_e td_return;
 td_thrinfo_t to;
 td_return = (*get_info)(th_p, &to);
 if ( td_return == TD_ERR ) {
   printf("Thread update failed\n");
   return_val = 1;
 }
 return return_val;
}

FILES /usr/lib/libthread_db.so.1

SEE ALSO proc_service(3T), thr_getspecific(3T)
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 */
};

MT-LEVEL
MT-Safe

DESCRIPTION
lio_listio() 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)`.

**LIO_NOP**
causes the `list` entry to be ignored.

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

When `mode` has the value `LIO_WAIT`, a pointer to a signal control structure, `sig`, is used to define both the signal to be generated and how the calling process will be notified upon I/O completion. If `sig->sigev_notify` is `SIGEV_NONE`, then no signal will be posted upon I/O completion, but the error status and the return status for the operation will be set appropriately. If `sig->sigev_notify` is `SIGEV_SIGNAL`, then the signal specified in `sig->sigev_signo` will be sent to the process. If the `SA_SIGINFO` flag is set for that signal number, then the signal will be queued to the process and the value specified in `sig->sigev_value` will be the `si_value` component of the generated signal (see `siginfo(5)`).

The behavior of this function is altered according to the definitions of synchronized I/O data integrity completion and synchronized I/O file integrity completion if synchronized I/O is enabled on the file associated with `aio_fildes`. (see `fcntl(5)` definitions of `O_DSYNC` and `O_SYNC`.)

**RETURN VALUES**

If the `mode` argument has the value `LIO_NOWAIT`, and the I/O operations are successfully queued, `lio_listio()` returns 0; otherwise, it returns -1, and sets `errno` to indicate the error condition.

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

In either case, the return value only indicates the success or failure of the `lio_listio()` call itself, not the status of the individual I/O requests. In some cases, one or more of the I/O requests contained in the list may fail. Failure of an individual request does not prevent completion of any other individual request. To determine the outcome of each I/O request, the application must examine the error status associated with each `aiocb` control block. Each error status so returned is identical to that returned as a result of an `aio_read(3R)` or `aio_write(3R)` function.

**ERRORS**

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

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

**EINVAL**
The `mode` argument is an improper value.
The value of \( nent \) 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; aio_return(3R) to determine if any request has completed; or aio_error(3R) to determine if any request was canceled.

**EIO**
One or more of the individual I/O operations failed. Using aio_error(3R) with each aiocb structure will determine the individual request(s) that failed.

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

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

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

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

**EINPROGRESS**
The requested I/O is in progress.

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

**NOTES**
Applications compiled under Solaris 2.3 and 2.4 and using POSIX aio must be recompiled to work correctly when Solaris supports the Asynchronous Input and Output 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
listen – listen for connections on a socket

SYNOPSIS
cc [ flag . . . ] file . . . −lsocket −lsocket [ library . . . ]
#include <sys/types.h>
#include <sys/socket.h>
int listen(int s, int backlog);

MT-LEVEL
Safe

DESCRIPTION
To accept connections, a socket is first created with socket(3N), a backlog for incoming
cannections 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().

SEE ALSO
accept(3N), connect(3N), socket(3N)

NOTES
There is currently no backlog limit.
localeconv – get numeric formatting information

**SYNOPSIS**

```c
#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
elements of mon_grouping are interpreted according to the rules described
under grouping.

char *positive_sign
The string used to indicate a nonnegative-valued formatted monetary quantity.

char *negative_sign
The string used to indicate a negative-valued formatted monetary quantity.

char int_frac_digits
The number of fractional digits (those to the right of the decimal point) to be
displayed in an internationally formatted monetary quantity.

char frac_digits
The number of fractional digits (those to the right of the decimal point) to be
displayed in a formatted monetary quantity.

char p_cs_precedes
Set to 1 or 0 if the currency_symbol respectively precedes or succeeds the value
for a nonnegative formatted monetary quantity.
char p_sep_by_space
Set to 1 or 0 if the currency_symbol respectively is or is not separated by a space from the value for a nonnegative formatted monetary quantity.

char n_cs_precedes
Set to 1 or 0 if the currency_symbol respectively precedes or succeeds the value for a negative formatted monetary quantity.

char n_sep_by_space
Set to 1 or 0 if the currency_symbol respectively is or is not separated by a space from the value for a negative formatted monetary quantity.

char p_sign_posn
Set to a value indicating the positioning of the positive_sign for a nonnegative formatted monetary quantity. The value of p_sign_posn is interpreted according to the following:

0 Parentheses surround the quantity and currency_symbol.
1 The sign string precedes the quantity and currency_symbol.
2 The sign string succeeds the quantity and currency_symbol.
3 The sign string immediately precedes the currency_symbol.
4 The sign string immediately succeeds the currency_symbol.

char n_sign_posn
Set to a value indicating the positioning of the negative_sign for a negative formatted monetary quantity. The value of n_sign_posn is interpreted according to the rules described under p_sign_posn.

RETURN VALUES localeconv() returns a pointer to the filled-in object. The structure pointed to by the return value may be overwritten by a subsequent call to localeconv().

EXAMPLES The following table illustrates the rules used by four countries to format monetary quantities.

<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.1.234</td>
<td>-L.1.234</td>
<td>ITL.1.234</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;&quot;</td>
<td>&quot;&quot;</td>
<td>&quot;&quot;</td>
<td>&quot;&quot;</td>
</tr>
<tr>
<td>positive_sign</td>
<td>&quot;&quot;</td>
<td>&quot;&quot;</td>
<td>&quot;&quot;</td>
<td>&quot;&quot;</td>
</tr>
</tbody>
</table>

modified 18 Feb 1994
<table>
<thead>
<tr>
<th>negative_sign</th>
<th>&quot;-&quot;</th>
<th>&quot;-&quot;</th>
<th>&quot;-&quot;</th>
<th>&quot;C&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>int_frac_digits</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>frac_digits</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>p_cs_precedes</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>p_sep_by_space</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>n_cs_precedes</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>n_sep_by_space</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>p_sign_posn</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>n_sign_posn</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

**FILES**

/usr/lib/locale/locale/LC_MONETARY/monetary

LC_MONETARY database for locale

/usr/lib/locale/locale/LC_NUMERIC/numeric

LC_NUMERIC database for locale

**SEE ALSO**

chrtbl(1M), montbl(1M), setlocale(3C)

**NOTES**

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

**LC_MONETARY**

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

**LC_NUMERIC**

Determines how numeric formats are handled. In the "C" locale, numeric handling follows the U.S. rules.
NAME
lockf – record locking on files

SYNOPSIS
#include <unistd.h>

int lockf(int fd, int function, long size);

MT-LEVEL
MT-Safe

DESCRIPTION
lockf() allows sections of a file to be locked; advisory or mandatory write locks depending
on the mode bits of the file (see chmod(2)). Locking calls from other processes that
attempt to lock the locked file section will either return an error value or be put to sleep
until the resource becomes unlocked. All the locks for a process are removed when the
process terminates. (See fcntl(2) for more information about record locking.)

fd 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.

size 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 posi-
tive 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.

F_ULOCK requests may, in whole or in part, release one or more locked sections controlled by the process. When sections are not fully released, the remaining sections are still locked by the process. Releasing the center section of a locked section requires an additional element in the table of active locks. If this table is full, an errno is set to EDEADLK and the requested section is not released.

A potential for deadlock occurs if a process controlling a locked resource is put to sleep by requesting another process’s locked resource. Thus calls to lockf() or fcntl(2) scan for a deadlock prior to sleeping on a locked resource. An error return is made if sleeping on the locked resource would cause a deadlock.

Sleeping on a resource is interrupted with any signal. The alarm(2) function may be used to provide a timeout facility in applications that require this facility.

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

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

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

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

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

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

3C-654 modified 3 May 1994
NAME | lsearch, lfind – linear search and update  
SYNOPSIS | \texttt{#include <search.h>}  
\texttt{void \ast lsearch(const void \ast key, void \ast base, size_t \ast nelp, size_t width,}  
\texttt{int (\ast compar)(const void \ast, const void \ast));}  
\texttt{void \ast lfind(const void \ast key, const void \ast base, size_t \ast nelp, size_t width,}  
\texttt{int (\ast compar)(const void \ast, const void \ast));}  
MT-LEVEL | Safe  
DESCRIPTION | \texttt{lsearch()} is a linear search routine generalized from Knuth (6.1) Algorithm S. (See \textit{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. \texttt{key} points to the datum to be sought in the table. \texttt{base} points to the first element in the table. \texttt{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. \texttt{width} is the size of an element in bytes. \texttt{compar} is a pointer to the comparison function that the user must supply (\texttt{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.  
\texttt{lfind()} is the same as \texttt{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 \texttt{TABSIZE} strings of length less than \texttt{ELSIZE} and store them in a table, eliminating duplicates, and then will print each entry.  
\texttt{#include <search.h>}  
\texttt{#include <string.h>}  
\texttt{#include <stdlib.h>}  
\texttt{#include <stdio.h>}  
\texttt{#define TABSIZE 50}  
\texttt{#define ELSIZE 120}  
\texttt{main()}  
\texttt{\{}  
\texttt{char line[ELSIZE];} \texttt{\};}  
\texttt{/* buffer to hold input string */}  
\texttt{char tab[TABSIZE][ELSIZE];} \texttt{\};}  
\texttt{/* table of strings */}  
modified 22 Jan 1993
size_t nel = 0; /* number of entries in tab */
int i;

while (fgets(line, ELSIZE, stdin) != NULL &&
    nel < TABSIZE)
    (void) lsearch(line, tab, &nel, ELSIZE, mycmp);
for( i = 0; i < nel; i++ )
    (void) fputs(tab[i], stdout);
return 0;
}

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


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);

MT-LEVEL
MT-Safe

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.

modified 22 Jan 1993 3-657
MADV_DONTNEED
    Tells the kernel that the specified address range is no longer needed, so the system starts to free the resources associated with the address range.

`madvise()` should be used by programs with specific knowledge of their access patterns over a memory object, such as a mapped file, to increase system performance.

RETURN VALUES
`madvise()` returns:
0 on success.
-1 on failure and sets `errno` to indicate the error.

ERRORS
`EINVAL` `addr` is not a multiple of the page size as returned by `sysconf(3C)`.
The length of the specified address range is less than or equal to 0, or the advice was invalid.

`EIO` An I/O error occurred while reading from or writing to the file system.

`ENOMEM` Addresses in the range `[addr, addr + len)` are outside the valid range for the address space of a process, or specify one or more pages that are not mapped.

`ESTALE` Stale nfs file handle.

SEE ALSO
`cat(1), cp(1), mmap(2), sysconf(3C)`
NAME  maillock – manage lockfile for user’s mailbox

SYNOPSIS  cc [ flag  ...] file  ...  -lmail [ library  ... ]
#include <maillock.h>
int maillock(const char *user, int retrycnt);
int mailunlock(void);

MT-LEVEL  Unsafe

DESCRIPTION  The maillock() function attempts to create a lockfile for the user’s mailfile. If a lockfile already exists, and it has not been modified in the last 5 minutes, maillock() will remove the lockfile and set it’s own lockfile.

It is crucial that programs locking mail files refresh their locks at least every minute. Lockfiles are refreshed by calling the routine touchlock() with no arguments. It should be called before doing anything that might keep the system busy for a long time. It should be called at least every 3 minutes or the process may lose its lock on the mail file. In practice, arranging to call it once a minute works well.

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. maillock() assumes that users’ 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 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 22 Jan 1993  3X-659
maillock() 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  
```c
#include <ucontext.h>

void makecontext(ucontext_t *ucp, void(*)(void), int argc, ...);
int swapcontext(ucontext_t *oucp, ucontext_t *ucp);
```

MT-LEVEL  MT-Safe

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.

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 return a value of zero. Otherwise, a value of −1 is returned and errno is set to indicate the error.

ERRORS  These functions will fail if either of the following is true:

EFAULT  ucp or oucp points to an invalid address.
ENOMEM  ucp does not have enough stack left to complete the operation.

SEE ALSO  exit(2), getcontext(2), sigaction(2), sigprocmask(2), ucontext(5)

NOTES  The size of the ucontext_t structure may change in future releases. To remain binary compatible, users of these features must always use makecontext() or getcontext() to create new instances of them.
NAME
makedev, major, minor – manage a device number

SYNOPSIS
#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);

MT-LEVEL
MT-Safe

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.

SEE ALSO
mknod(2), stat(2)
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);

MT-LEVEL Safe

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. Note: if the allocated block is beyond the current stack limit, the resulting behavior is undefined.

modified 4 May 1994
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.

SEE ALSO
brk(2), getrlimit(2), bsdmalloc(3X), malloc(3X), mapmalloc(3X)

WARNINGS
Undefined results will occur if the size requested for a block of memory exceeds the maximum size of a process’s heap, which may be obtained with getrlimit().

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

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

free() does not set errno.

modified 4 May 1994
NAME    malloc, free, realloc, calloc, mallopt, mallinfo – memory allocator

SYNOPSIS 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);

MT-LEVEL Safe

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 initial-
ized 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 max-
fast 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 \texttt{grain} is set.

\textbf{M\_KEEP}  Preserve data in a freed block until the next \texttt{malloc()}, \texttt{realloc()}, or \texttt{calloc()}. This option is provided only for compatibility with the old version of \texttt{malloc()} and is not recommended.

These values are defined in the \texttt{<malloc.h>} header.

\texttt{mallopt()} may be called repeatedly, but may not be called after the first small block is allocated.

\texttt{mallinfo()} provides instrumentation describing space usage. It returns the \texttt{mallinfo} structure with the following members:

\begin{verbatim}
  int arena;       /* total space in arena */
  int ordblks;    /* number of ordinary blocks */
  int smblks;     /* number of small blocks */
  int hblkhd;     /* space in holding block headers */
  int hblks;      /* number of holding blocks */
  int usmblks;    /* space in small blocks in use */
  int fsmbks;     /* space in free small blocks */
  int ordblk;     /* space in ordinary blocks in use */
  int fordblks;   /* space in free ordinary blocks */
  int keepcost;   /* space penalty if keep option */
                  /* is used */
\end{verbatim}

The \texttt{mallinfo} structure is defined in the \texttt{<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**  \texttt{malloc()}, \texttt{realloc()}, and \texttt{calloc()} return a NULL pointer if there is not enough available memory. When \texttt{realloc()} returns NULL, the block pointed to by \texttt{ptr} is left intact. If \texttt{mallopt()} is called after any allocation or if \texttt{cmd} or \texttt{value} are invalid, non-zero is returned. Otherwise, it returns zero.

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

\begin{itemize}
  \item \texttt{ENOMEM}  size bytes of memory exceeds the physical limits of your system, and cannot be allocated.
  \item \texttt{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.
\end{itemize}

**SEE ALSO**  \texttt{brk(2), malloc(3C), bsdmalloc(3X)}

**NOTES**  Note that unlike \texttt{malloc(3C)}, this package does not preserve the contents of a block when it is freed, unless the \texttt{M\_KEEP} option of \texttt{mallopt()} is used.

3X-666  modified 11 Feb 1993
Undocumented features of malloc(3C) have not been duplicated. Function prototypes for malloc(), realloc(), calloc(), and free() are also defined in the <malloc.h> header for compatibility with old applications. New applications should include <stdlib.h> to access the prototypes for these functions.

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

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

free() does not set errno.
NAME mapmalloc, calloc, cfree, free, realloc, − memory allocator
SYNOPSIS cc [ 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);

MT-LEVEL Safe

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/libmapmalloc

SEE ALSO
brk(2), getrlimit(2), mmap(2), realloc(3C)
NAME  
matherr – math library exception-handling function

SYNOPSIS  
#include <math.h>

int matherr(struct exception *exc);

MT-LEVEL  
MT-Safe

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

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

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

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOMAIN</td>
<td>argument domain exception</td>
</tr>
<tr>
<td>SING</td>
<td>argument singularity</td>
</tr>
<tr>
<td>OVERFLOW</td>
<td>overflow range exception</td>
</tr>
<tr>
<td>UNDERFLOW</td>
<td>underflow range exception</td>
</tr>
<tr>
<td>TLOSS</td>
<td>total loss of significance</td>
</tr>
<tr>
<td>PLOSS</td>
<td>partial loss of significance</td>
</tr>
</tbody>
</table>

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

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

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

SVID3 STANDARD CONFORMANCE  
If matherr() is not supplied by the user, the default matherr exception-handling mechanisms, summarized in the table below, will be invoked upon exception:

<table>
<thead>
<tr>
<th>Domain</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOMAIN</td>
<td>0.0 is usually returned, errno is set to EDOM, and a message is usually printed on standard error.</td>
</tr>
</tbody>
</table>
**SING**

The largest finite single-precision number, **HUGE** of appropriate sign is returned, **errno** is set to **EDOM**, and a message is printed on standard error.

**OVERFLOW**

The largest finite single-precision number, **HUGE** of appropriate sign is usually returned, **errno** is set to **ERANGE**.

**UNDERFLOW**

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

**TLOSS**

0.0 is returned, **errno** is set to **ERANGE**, and a message is printed on standard error.

In general, **errno** is not a reliable error indicator in that it may be unexpectedly set by a function in a handler for an asynchronous signal.

---

### DEFAULT ERROR HANDLING PROCEDURES (SVID3)

<table>
<thead>
<tr>
<th>Types of Errors</th>
<th>&lt;math.h&gt; type</th>
<th>DOMAIN</th>
<th>SING</th>
<th>OVERFLOW</th>
<th>UNDERFLOW</th>
<th>TLOSS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>errno</strong></td>
<td></td>
<td>EDOM</td>
<td>EDOM</td>
<td>ERANGE</td>
<td>ERANGE</td>
<td>ERANGE</td>
</tr>
<tr>
<td>IEEE Exception</td>
<td></td>
<td>Invalid Operation</td>
<td>Division by Zero</td>
<td>Overflow</td>
<td>Underflow</td>
<td></td>
</tr>
<tr>
<td>fp_exception_type</td>
<td></td>
<td>fp_invalid</td>
<td>fp_division</td>
<td>fp_overflow</td>
<td>fp_underflow</td>
<td></td>
</tr>
<tr>
<td>ACOS, ASIN (</td>
<td>x</td>
<td>&gt; 1):</td>
<td>Md, 0.0</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>ACOSH (x &lt; 1), ATANH (</td>
<td>x</td>
<td>&gt; 1):</td>
<td>NaN</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>ATAN2 (0,0):</td>
<td>Md, 0.0</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>COSH, SINH:</td>
<td>–</td>
<td>–</td>
<td>±HUGE</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>EXP:</td>
<td>–</td>
<td>–</td>
<td>+HUGE</td>
<td>0.0</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>FMOD (x,0):</td>
<td>x</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>HYPT:</td>
<td>–</td>
<td>–</td>
<td>+HUGE</td>
<td>–</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>
<td>–</td>
</tr>
<tr>
<td>LGAMMA:</td>
<td></td>
<td>–</td>
<td>–</td>
<td>+HUGE</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>usual cases</td>
<td></td>
<td>–</td>
<td>–</td>
<td>Ms, +HUGE</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>(x = 0 or –integer)</td>
<td></td>
<td>–</td>
<td>Ms, +HUGE</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>LOG, LOG10:</td>
<td></td>
<td>Md, –HUGE</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>(x &lt; 0)</td>
<td></td>
<td>–</td>
<td>Ms, –HUGE</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>(x = 0)</td>
<td></td>
<td>–</td>
<td>Ms, –HUGE</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>POW:</td>
<td></td>
<td>–</td>
<td>–</td>
<td>±HUGE</td>
<td>±0.0</td>
<td>–</td>
</tr>
<tr>
<td>usual cases</td>
<td></td>
<td>–</td>
<td>–</td>
<td>±HUGE</td>
<td>±0.0</td>
<td>–</td>
</tr>
<tr>
<td>(x &lt; 0) ** ++ (y not an integer)</td>
<td>Md, 0.0</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>0 ** 0</td>
<td>Md, 0.0</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>0 ** (y &lt; 0)</td>
<td>Md, 0.0</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>REMAINDER (x,0):</td>
<td></td>
<td>NaN</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>SCALB:</td>
<td>–</td>
<td>–</td>
<td>±HUGE, VAL</td>
<td>±0.0</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>SQRT (x &lt; 0):</td>
<td></td>
<td>Md, 0.0</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Y0, Y1, YN:</td>
<td>Md, –HUGE</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>(x &lt; 0)</td>
<td></td>
<td>–</td>
<td>Md, –HUGE</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>(x = 0)</td>
<td></td>
<td>–</td>
<td>Md, –HUGE</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>(x &gt; X_TLOSS)</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>Mt, 0.0</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>
### ABBREVIATIONS

- **Md**: Message is printed (DOMAIN error).
- **Ms**: Message is printed (SING error).
- **Mt**: Message is printed (TLOSS error).
- **NaN**: IEEE NaN result and invalid operation exception.
- **HUGE**: Maximum finite single-precision floating-point number.
- **HUGE_VAL**: IEEE ±∞ result and division-by-zero exception.
- **X_TLOSS**: The value X_TLOSS is defined in `<values.h>.

The interaction of IEEE arithmetic and `matherr()` is not defined when executing under IEEE rounding modes other than the default round to nearest: `matherr()` may not be called on overflow or underflow, and the SUN-provided `matherr()` may return results that differ from those in this table.

### X/Open (XPG3) Standard Conformance

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

<table>
<thead>
<tr>
<th>Types of Errors</th>
<th>&lt;math.h&gt; type</th>
<th>DOMAIN</th>
<th>SING</th>
<th>OVERFLOW</th>
<th>UNDERFLOW</th>
<th>TLOSS</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>errno</code></td>
<td></td>
<td>EDOM</td>
<td>EDOM</td>
<td>ERANGE</td>
<td>ERANGE</td>
<td>ERANGE</td>
</tr>
<tr>
<td><code>acos</code>, <code>asin</code></td>
<td>(</td>
<td>x</td>
<td>&gt; 1):</td>
<td>0.0</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td><code>atan2(0,0)</code>:</td>
<td>0.0</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td><code>cosh</code>, <code>sinh</code>:</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>±HUGE_VAL</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td><code>exp</code>:</td>
<td>−</td>
<td>−</td>
<td>(HUGE_VAL)</td>
<td>0.0</td>
<td>−</td>
<td></td>
</tr>
<tr>
<td><code>fmod(x,0)</code>:</td>
<td>[NaN]</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td><code>hypot</code>:</td>
<td>−</td>
<td>−</td>
<td>(HUGE_VAL)</td>
<td>−</td>
<td>−</td>
<td></td>
</tr>
<tr>
<td><code>j0</code>, <code>j1</code>, <code>jn</code> (</td>
<td>x</td>
<td>&gt; X_TLOSS):</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td><code>lgamma</code>:</td>
<td>usual cases</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>
<td></td>
</tr>
<tr>
<td><code>log</code>, <code>log10</code>:</td>
<td>(x &lt; 0)</td>
<td>−HUGE_VAL</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td>(x = 0)</td>
<td>−</td>
<td>−HUGE_VAL</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td><code>pow</code>:</td>
<td>usual cases</td>
<td>−</td>
<td>−</td>
<td>±HUGE_VAL</td>
<td>±0.0</td>
<td>−</td>
</tr>
<tr>
<td>(x &lt; 0) ** (y not an integer)</td>
<td>0.0</td>
<td>−</td>
<td>−</td>
<td>±HUGE_VAL</td>
<td>±0.0</td>
<td>−</td>
</tr>
<tr>
<td>0 ** 0</td>
<td>[1.0]</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>
<td>−</td>
</tr>
<tr>
<td><code>sqrt(x &lt; 0)</code>:</td>
<td>0.0</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td><code>y0</code>, <code>y1</code>, <code>yn</code>:</td>
<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>
<td>−</td>
</tr>
<tr>
<td>(x &gt; X_TLOSS)</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>0.0</td>
</tr>
</tbody>
</table>

3M-672 modified 4 Mar 1994
### ABBREVIATIONS

| [...]| **errno** is not to be relied upon in all braced cases. |
| NaN  | IEEE NaN result and invalid operation exception. |
| HUGE_VAL | IEEE ∞ result and division-by-zero exception. |
| X_TLOSS | The value X_TLOSS is defined in <values.h>. |

### ANSI/ISO-C STANDARD CONFORMANCE

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

The following table summarizes the values returned in the exceptional cases.

<table>
<thead>
<tr>
<th>ANSI/ISO-C ERROR HANDLING PROCEDURES (compile with &quot;cc -Xc&quot;)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Types of Errors</td>
</tr>
<tr>
<td>&lt;math.h&gt; type</td>
</tr>
<tr>
<td><strong>errno</strong></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>(x &lt; 0)</td>
</tr>
<tr>
<td>(x = 0)</td>
</tr>
<tr>
<td>(x = 0)</td>
</tr>
<tr>
<td>(y is not an integer)</td>
</tr>
<tr>
<td>(x &lt; 0) ** (y &lt; 0)</td>
</tr>
<tr>
<td>SQRT(x &lt; 0):</td>
</tr>
</tbody>
</table>

### 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);
```

modified 4 Mar 1994
abort();
break;
}

return (0); /* all other exceptions, execute default procedure */
}
NAME  mbchar, mbtowc, mblen, wctomb − multibyte character handling

SYNOPSIS  #include <stdlib.h>
#include <limits.h>

int mbtowc(wchar_t *pwc, const char *s, size_t n);
int mblen(const char *s, size_t n);
int wctomb(char *s, wchar_t wchar);

MT-LEVEL  MT-Safe with exceptions

DESCRIPTION  Multibyte characters are used to represent characters in an extended character set. This is needed for locales where 8 bits are not enough to represent all the characters in the character set.

The multibyte character handling functions provide the means of translating multibyte characters into wide characters and back again. Wide characters have type wchar_t (defined in <stdlib.h>), which is an integral type whose range of values can represent distinct codes for all members of the largest extended character set specified among the supported locales.

A maximum of 3 extended character sets are supported for each locale. The number of bytes in an extended character set is defined by the LC_CTYPE category of the locale (see setlocale(3C)). However, the maximum number of bytes in any multibyte character will never be greater than MB_LEN_MAX, which is defined in <limits.h>. The maximum number of bytes in a character in an extended character set in the current locale is given by the macro, MB_CUR_MAX, defined in <stdlib.h>.

mbtowc() determines the number of bytes that comprise the multibyte character pointed to by s. Also, if pwc is not a null pointer, mbtowc() converts the multibyte character to a wide character and places the result in the object pointed to by pwc. (The value of the wide character corresponding to the null character is zero.) At most n bytes will be examined, starting at the byte pointed to by s.

If s is a null pointer, mbtowc() simply returns 0. If s is not a null pointer, then, if s points to the null character, mbtowc() returns 0; if the next n or fewer bytes form a valid multibyte character, mbtowc() returns the number of bytes that comprise the converted multibyte character; otherwise, s does not point to a valid multibyte character and mbtowc() returns -1.

mblen() determines the number of bytes comprising the multibyte character pointed to by s. It is equivalent to

mbtowc((wchar_t *)0, s, n);

wctomb() determines the number of bytes needed to represent the multibyte character corresponding to the code whose value is wchar, and, if s is not a null pointer, stores the multibyte character representation in the array pointed to by s. At most MB_CUR_MAX bytes are stored.
If \( s \) is a null pointer, \texttt{wctomb()} simply returns 0. If \( s \) is not a null pointer, \texttt{wctomb()} returns \(-1\) if the value of \texttt{wchar} does not correspond to a valid multibyte character; otherwise it returns the number of bytes that comprise the multibyte character corresponding to the value of \texttt{wchar}.

\textbf{SEE ALSO} \texttt{chrtbl(1M)}, \texttt{mbstring(3C)}, \texttt{setlocale(3C)}, \texttt{environ(5)}

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

mbstring, mbstowcs, wcstombs - multibyte string functions

SYNOPSIS

#include <stdlib.h>

size_t mbstowcs(wchar_t *pwcs, const char *s, size_t n);
size_t wcstombs(char *s, const wchar_t *pwcs, size_t n);

MT-LEVEL

MT-Safe with exceptions

DESCRIPTION

mbstowcs() converts a sequence of multibyte characters from the array pointed to by s into a sequence of corresponding wide character codes and stores these codes into the array pointed to by pwcs, stopping after n codes are stored or a code with value zero (a converted null character) is stored. If an invalid multibyte character is encountered, mbstowcs() returns (size_t) -1; otherwise, mbstowcs() returns the number of array elements modified, not including the terminating zero code, if any.

wcstombs() converts a sequence of wide character codes from the array pointed to by pwcs into a sequence of multibyte characters and stores these multibyte characters into the array pointed to by s, stopping if a multibyte character would exceed the limit of n total bytes or if a null character is stored. If a wide character code is encountered that does not correspond to a valid multibyte character, wcstombs() returns (size_t) -1; otherwise, wcstombs() returns the number of bytes modified, not including a terminating null character, if any. If s is a null pointer, wcstombs() returns the number of bytes required for the character array.

SEE ALSO

chrtbl(1M), mbchar(3C), setlocale(3C), environ(5)

NOTES

mbstowcs and wcstombs can be used safely in a multi-thread application, as long as setlocale(3C) is not being called to change the locale.
NAME mctl – memory management control

SYNOPSIS /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
MS_ASYNC 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 supportmunlock(). 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   cc [ flag . . . ] file . . . -lvolmgt [ library . . . ]
            #include <volmgt.h>
            char *media_findname(char *start);

MT-LEVEL    MT-Unsafe

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:

3X-680      modified 31 May 1995
ENXIO  The specified absolute pathname was not a character special device, and it was not a directory with a character special device in it.

EXAMPLES  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(""fred" is at \\
           nm","n", nm);
} else {
    (void) printf(""fred" is at \\
           nm", nm);
}
```

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 \\
           nm", nm);
} else {
    (void) printf("nothing in floppy0\n");
}
```

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), volfs(7FS)

NOTES  If media_findname() cannot find a match for the supplied name, it performs a volmgt_check(3X) and tries again, so it can be more efficient to perform volmgt_check() before calling media_findname().

Upon success media_findname() returns a pointer to string which has been allocated; this should be freed when no longer in use (see free(3C)).
**NAME**
media_getattr, media_setattr – get and set media attributes

**SYNOPSIS**

```c
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);
```

**MT-LEVEL**
MT-Safe

**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>pathname</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.

3X-682 modified 23 Aug 1995
Both `media_getattr()` and `media_setattr()` can fail returning a null pointer if an `open(2)` of the specified `vol_path` fails, if an `fstat(2)` of that pathname fails, or if that pathname is not a block or character special device.

`media_getattr()` can also fail if the specified attribute was not found, and `media_setattr()` can also fail if the caller doesn’t have permission to set the attribute, either because it’s 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\n");
} else {
    (void) printf("software can eject "fred"\n");
}
```

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\n");
}
```

### SEE ALSO

`cc(1B)`, `vold(1M)`, `lstat(2)`, `open(2)`, `readlink(2)`, `stat(2)`, `free(3C)`, `malloc(3C)`, `media_findname(3X)`, `volmgt_check(3X)`, `volmgt_inuse(3X)`, `volmgt_root(3X)`, `volmgt_running(3X)`, `volmgt_symname(3X)`

### 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, memccpy, memchr, memcmp, memcpy, memmove, memset – memory operations

SYNOPSIS  
#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);

MT-LEVEL  MT-Safe

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.

memcpy() 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.

SEE ALSO  string(3C)
NAME
menu_attributes, set_menu_fore, menu_fore, set_menu_back, menu_back,
set_menu_grey, menu_grey, set_menu_pad, menu_pad — control menus display attributes

SYNOPSIS
c
#include <menu.h>
int set_menu_fore(MENU *menu, chtype attr);
chtpe menu_fore(MENU *menu);
int set_menu_back(MENU *menu, chtype attr);
chtpe menu_back(MENU *menu);
int set_menu_grey(MENU *menu, chtype attr);
chtpe menu_grey(MENU *menu);
int set_menu_pad(MENU *menu, int pad);
int menu_pad(MENU *menu);

MT-LEVEL
Unsafe

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.
memu_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.

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

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

modified 22 Jan 1993
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);

MT-LEVEL  Unsafe

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.

SEE ALSO  curses(3X), menus(3X), panel_update(3X), panels(3X)

NOTES  The header <menu.h> automatically includes the headers <eti.h> and <curses.h>.
NAME      menu_driver – command processor for the menus subsystem

SYNOPSIS  cc [ flag . . . ] file . . . -lmenu -lcurses [ library . . ]
#include <menu.h>
int menu_driver(MENU *menu, int c);

MT-LEVEL   Unsafe

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.

modified 22 Jan 1993
E_NOT_POSTED The menu has not been posted.
E_UNKNOWN_COMMAND An unknown request was passed to the menu driver.
E_NO_MATCH The character failed to match.
E_NOT_SELECTABLE The item cannot be selected.
E_REQUEST_DENIED The menu driver could not process the request.

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

NOTES Application defined commands should be defined relative to (greater than) MAX_COMMAND, the maximum value of a request listed above.
The header <menu.h> automatically includes the headers <eti.h> and <curses.h>. 
NAME

menu_format, set_menu_format – set and get maximum numbers of rows and columns in menus

SYNOPSIS

cc [ flag ... ] file ... -lmenu -l curses [ library ... ]
#include <menu.h>

int set_menu_format(MENU *menu, int rows, int cols);
void menu_format(MENU *menu, int *rows, int *cols);

MT-LEVEL

Unsafe

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.

SEE ALSO

curses(3X), menus(3X)

NOTES

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

modified 22 Jan 1993
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);

MT-LEVEL
Unsafe

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

NOTES  The header <menu.h> automatically includes the headers <eti.h> and <curses.h>.
NAME  menu_item_current, set_current_item, current_item, set_top_row, top_row, item_index – set and get current menus items

SYNOPSIS  cc [ flag ... ] file ... -lmenu -lcurses [ library ... ]

#include <menu.h>

int set_current_item(MENU *menu, ITEM *item);
ITEM *current_item(MENU *menu);
int set_top_row(MENU *menu, int row);
int top_row(MENU *menu);
int item_index(ITEM *item);

MT-LEVEL  Unsafe

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.

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.

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

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

3X-692 modified 22 Jan 1993
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);

MT-LEVEL
Unsafe

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.

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

NOTES
The header <menu.h> automatically includes the headers <eti.h> and <curses.h>.
NAME    menu_item_new, new_item, free_item – create and destroy menus items

SYNOPSIS cc [ flag ... ] file ... -lmenu -lcurses [ library ... ]
    #include <menu.h>
    ITEM *new_item(char *name, char *desc);
    int free_item(ITEM *item);

MT-LEVEL Unsafe

DESCRIPTION new_item() creates a new item from 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.

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

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

3X-694 modified 22 Jan 1993
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);

MT-LEVEL    Unsafe

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.

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

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

modified 22 Jan 1993
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);

MT-LEVEL Unsafe

DESCRIPTION Every item has an associated user pointer that can be used to store relevant information. set_item_userptr() sets the user pointer of 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.

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

NOTES The header <menu.h> automatically includes the headers <eti.h> and <curses.h>.
NAME
menu_item_value, set_item_value, item_value — set and get menus item values

SYNOPSIS
cc [ flag ... ] file ... -lmenu -lcurses [ library ... ]
#include <menu.h>
int set_item_value(ITEM *item, int bool);
int item_value(ITEM *item);

MT-LEVEL
Unsafe

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.

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

NOTES
The header <menu.h> automatically includes the headers <eti.h> and <curses.h>.
NAME    menu_item_visible, item_visible – tell if menus item is visible

SYNOPSIS cc [ flag ... ] file ... -lmenu -lcurses [ library ... ]
#include <menu.h>
int item_visible(ITEM *item);

MT-LEVEL Unsafe

DESCRIPTION A menu item is visible if it currently appears in the subwindow of a posted menu.
item_visible() returns TRUE if item is visible, otherwise it returns FALSE.

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

NOTES The header <menu.h> automatically includes the headers <eti.h> and <curses.h>.
NAME  menu_items, set_menu_items, item_count – connect and disconnect items to and from menus

SYNOPSIS  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);

MT-LEVEL  Unsafe

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.

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

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

modified 22 Jan 1993 3X-699
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);

MT-LEVEL  Unsafe

DESCRIPTION menus displays mark strings to distinguish selected items in a menu (or the current item in a single-valued menu). set_menu_mark() sets the mark string of 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.

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

NOTES The header <menu.h> automatically includes the headers <eti.h> and <curses.h>.
NAME  menu_new, new_menu, free_menu – create and destroy menus

SYNOPSIS  cc [ flag ... ] file ... -lmenu -lcurses [ library ... ]
#include <menu.h>

MENU *new_menu(ITEM **items);
int free_menu(MENU *menu);

MT-LEVEL  Unsafe

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.

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

NOTES  The header <menu.h> automatically includes the headers <eti.h> and <curses.h>.
NAME  menu_opts, set_menu_opts, menu_opts_on, menu_opts_off – menus option routines

SYNOPSIS  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);

MT-LEVEL  Unsafe

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.

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

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

3X-702  modified 22 Jan 1993
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);

MT-LEVEL    Unsafe

DESCRIPTION Every menu has a pattern buffer to match entered data with menu items.

set_menu_pattern() sets the pattern buffer to 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 follow-
ing:

E_OK          The routine returned successfully.
E_SYSTEM_ERROR System error.
E_BAD_ARGUMENT An incorrect argument was passed to
                 the routine.
E_NO_MATCH    The character failed to match.

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

NOTES       The header <menu.h> automatically includes the headers <eti.h> and <curses.h>.
NAME    menu_post, post_menu, unpost_menu – write or erase menus from associated subwindows

SYNOPSIS    cc [ flag ... ] file ... -lmenu -lcurses [ library ... ]
             #include <menu.h>
             int post_menu(MENU *menu);
             int unpost_menu(MENU *menu);

MT-LEVEL    Unsafe

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.

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

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

menu_userptr, set_menu_userptr – associate application data with menus

SYNOPSIS

cc [ flag ... ] file ... -lmenu -lcurses [ library ... ]
#include <menu.h>
char *menu_userptr(MENU *menu);
int set_menu_userptr(MENU *menu, char *userptr);

MT-LEVEL

Unsafe

DESCRIPTION

Every menu has an associated user pointer that can be used to store relevant information. set_menu_userptr() sets the user pointer of 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.

SEE ALSO

curses(3X), menus(3X)

NOTES

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

modified 22 Jan 1993
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);

MT-LEVEL

Unsafe

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.

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

NOTES

The header <menu.h> automatically includes the headers <eti.h> and <curses.h>.
NAME | menus – character based menus package

SYNOPSIS | #include <menu.h>

MT-LEVEL | Unsafe

DESCRIPTION | The menu library is built using the curses library, and any program using menus routines must call one of the curses initialization routines, such as initscr. A program using these routines must be compiled with −lmenu and −lcurses on the cc command line.

The menus package gives the applications programmer a terminal-independent method of creating and customizing menus for user interaction. The menus package includes: item routines, which are used to create and customize menu items; and menu routines, which are used to create and customize menus, assign pre- and post-processing routines, and display and interact with menus.

Current Default Values for Item Attributes | The menus package establishes initial current default values for item attributes. During item initialization, each item attribute is assigned the current default value for that attribute. An application can change or retrieve a current default attribute value by calling the appropriate set or retrieve routine with a NULL item pointer. If an application changes a current default item attribute value, subsequent items created using new_item() will have the new default attribute value. The attributes of previously created items are not changed if a current default attribute value is changed.

Routine Name Index | The following table lists each menus routine and the name of the manual page on which it is described.

<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>
</tbody>
</table>

modified 22 Jan 1993
RETURN VALUES

Routines that return pointers always return NULL on error. Routines that return an integer return one of the following:

- **E_OK**: The routine returned successfully.
- **E_SYSTEM_ERROR**: System error.
- **E_BAD_ARGUMENT**: An incorrect argument was passed to the routine.
- **E_POSTED**: The menu is already posted.
- **E_CONNECTED**: One or more items are already connected to another menu.
- **E_BAD_STATE**: The routine was called from an initialization or termination function.
- **E_NO_ROOM**: The menu does not fit within its subwindow.
- **E_NOT_POSTED**: The menu has not been posted.
- **E_UNKNOWN_COMMAND**: An unknown request was passed to the menu driver.
- **E_NO_MATCH**: The character failed to match.
- **E_NOT_SELECTABLE**: The item cannot be selected.
- **E_NOT_CONNECTED**: No items are connected to the menu.
- **E_REQUEST_DENIED**: The menu driver could not process the request.

SEE ALSO
curses(3X)

NOTES

The header `<menu.h>` automatically includes the headers `<eti.h>` and `<curses.h>`.
NAME  mkdirp, rmdirp – create, remove directories in a path

SYNOPSIS  
cc [ flag ... ] file ... -Igen [ library ... ]
#include <libgen.h>
#include <assert.h>
int mkdirp(const char *path, mode_t mode);
int rmdirp(char *dir, char *dir1);

MT-LEVEL  MT-Safe

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  /* 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.

SEE ALSO  mkdir(2), rmdir(2)

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);
```

MT-LEVEL  MT-Safe
Async-Signal-Safe

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.

SEE ALSO  `mkdir(1)`, `chmod(2)`, `exec(2)`, `mknod(2)`, `umask(2)`, `fs_ufs(4)`, `stat(5)`

NOTES  Bits other than the file permission bits in `mode` are ignored.
NAME  mktemp – make a unique file name

SYNOPSIS  
#include <stdlib.h>
char *mktemp(char *template);

MT-LEVEL  Safe

DESCRIPTION  mktemp() replaces the contents of the string pointed to by template with a unique file name, and returns template. The string in template should look like a file name with six trailing Xs; mktemp() will replace the Xs with a character string that can be used to create a unique file name.

RETURN VALUES  mktemp() will assign to template the empty string if it cannot create a unique name.

SEE ALSO  tmpfile(3S), tmpnam(3S)

NOTES  mktemp() can create only 26 unique file names per thread for each unique template.
NAME    mktime – converts a tm structure to a calendar time

SYNOPSIS #include <time.h>

time_t mktime(struct tm *timeptr);

MT-LEVEL Unsafe

DESCRIPTION mktime() 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:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tm_sec</td>
<td>seconds after the minute [0, 61]</td>
</tr>
<tr>
<td>tm_min</td>
<td>minutes after the hour [0, 59]</td>
</tr>
<tr>
<td>tm_hour</td>
<td>hour since midnight [0, 23]</td>
</tr>
<tr>
<td>tm_mday</td>
<td>day of the month [1, 31]</td>
</tr>
<tr>
<td>tm_mon</td>
<td>months since January [0, 11]</td>
</tr>
<tr>
<td>tm_year</td>
<td>years since 1900</td>
</tr>
<tr>
<td>tm_wday</td>
<td>days since Sunday [0, 6]</td>
</tr>
<tr>
<td>tm_yday</td>
<td>days since January 1 [0, 365]</td>
</tr>
<tr>
<td>tm_isdst</td>
<td>flag for daylight savings time</td>
</tr>
</tbody>
</table>

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 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, the correct timezone is determined and the components are not adjusted.

Local timezone information is used as if mktime() had called tzset() (see ctime(3C)).

mktime() returns the specified calendar time. If the calendar time cannot be represented, the function returns the value (time_t)−1.
EXAMPLES

What day of the week is July 4, 2001?

```c
#include <stdio.h>
#include <time.h>

static char *const wday[ ] = {
    "Sunday", "Monday", "Tuesday", "Wednesday",
    "Thursday", "Friday", "Saturday", "-unknown-
};

struct tm time_str;

/* ... */

  time_str.tm_year = 2001 - 1900;
  time_str.tm_mon = 7 - 1;
  time_str.tm_mday = 4;
  time_str.tm_hour = 0;
  time_str.tm_min = 0;
  time_str.tm_sec = 1;
  time_str.tm_isdst = -1;
  if (mktime(&time_str) == -1)
    time_str.tm_wday = 7;
  printf("%s\n", wday[time_str.tm_wday]);
```

SEE ALSO

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

NOTES

`tm_year` of the `tm` structure must be for year 1970 or later. Calendar times before 00:00:00 UTC, January 1, 1970 or after 03:14:07 UTC, January 19, 2038 cannot be represented.
NAME
mlock, munlock – lock (or unlock) pages in memory

SYNOPSIS
#include <sys/types.h>
int mlock(caddr_t addr, size_t len);
int munlock(caddr_t addr, size_t len);

MT-LEVEL
MT-Safe

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
EAGAIN mlock() only. Some or all of the memory identified by the range [addr, addr + len) could not be locked because of insufficient system resources.
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 super-user.

modified 22 Jan 1993
SEE ALSO  
fork(2), memcntl(2), mmap(2), plock(3C), mlockall(3C), sysconf(3C)

NOTES  
mlock and munlock require super-user privileges.
NAME  
mlockall, munlockall – lock or unlock address space

SYNOPSIS  
#include <sys/mman.h>

int mlockall(int flags);
int munlockall(void);

MT-LEVEL  
MT-Safe

DESCRIPTION  
The function mlockall() causes all pages mapped by an address space to be locked in memory.
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 to mlockall(), then as mappings are added to the address space (or existing mappings are replaced) they will also be locked, provided sufficient memory is available.

Mappings locked via mlockall() with any option may be explicitly unlocked with a munlock() call.

The function munlockall() removes address space locks and locks on mappings in the address space.

All conditions and constraints on the use of locked memory as exist for mlock() apply to mlockall().

Locks established with mlockall() are not inherited by a child process after a fork() 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     mlockall() only. Some or all of the memory in the address space could not be locked due to sufficient resources.
EINVAL     flags contains values other than MCL_CURRENT and MCL_FUTURE.
EPERM      The process’s effective user ID is not super-user.

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

NOTES  
mlockall() and munlockall() require super-user privileges.

modified 22 Jan 1993
NAME
monitor – prepare process execution profile

SYNOPSIS
#include <mon.h>

void monitor(int (*lowpc)(), int (*highpc)(), WORD *buffer, size_t bufsize,
size_t nfunc);

MT-LEVEL
Safe

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 distri-
bution 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:

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:

```
monitor (&eprol, &etext, wbuf, wbufsz, 600);
```

where:
- `eprol` is the beginning of the user’s program when linked with `cc −p` (see `end(3C)`);
- `etext` is the end of the user’s program (see `end(3C)`);
- `wbuf` is an array of WORD with `wbufsz` elements;
- `wbufsz` is computed using the bufsize formula shown above with BARSIZE of 8;
- `600` is the number of call count cells that have been reserved in buffer.

These parameter settings establish the computation of an execution-time distribution histogram that uses profil() for the entire program, initially reserves room for 600 call count cells in buffer, and provides for enough histogram cells to generate significant distribution-measurement results. For more information on the effects of bufsize on execution-distribution measurements, see profil(2).

**EXAMPLES**

To stop execution monitoring and write the results to a file, use the following:

```
monitor( int (*)( )0, (int (*)( )0, (WORD *)0, 0, 0);
```

Use prof to examine the results.

**FILES**

```
mon.out
```

**SEE ALSO**

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

**NOTE**

Additional calls to monitor() after main() has been called and before exit() has been called will add to the function-call count capacity, but such calls will also replace and restart the profil() histogram computation.
NAME
mq_close – close a message queue

SYNOPSIS
cc [ flag ...] file ... -lposix4 [ library ...]
#include <mqueue.h>
int mq_close(mqd_t mqdes);

MT-LEVEL
MT-Safe

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.

SEE ALSO
mq_notify(3R), mq_open(3R), mq_unlink(3R)

BUGS
In Solaris 2.5, these functions always return -1 and set errno to ENOSYS, because this release does not support the Message Passing option. It is our intention to provide support for these interfaces in future releases.
NAME
mq_notify – notify process (or thread) that a message is available on a queue

SYNOPSIS
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 */
};

MT-LEVEL
MT-Safe

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 mes-
sage 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 mes-
sage 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_receive(3R)`, and no notification will be sent to the registered process. The resulting behavior is as if the message queue remains empty, and this notification will not be sent until the next arrival of a message at this queue.

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

**RETURN VALUES**

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

**ERRORS**

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

**SEE ALSO**

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

**BUGS**

In Solaris 2.5, these functions always return -1 and set `errno` to ENOSYS, because this release does not support the Message Passing option. It is our intention to provide support for these interfaces in future releases.
NAME  mq_open – open a message queue

SYNOPSIS  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 */
    ...                
};

MT-LEVEL  MT-Safe

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), exit(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.

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.
mq_open (3R)  Realtime Library  SunOS 5.5

**O_RDONLY**
Open the queue for reading messages. The process can use any of the functions allowed for **O_RDWR** and **O_WRONLY**. A message queue may be open multiple times in the same or different processes for receiving messages.

Any combination of the remaining flags may additionally be specified in the value of **oflag**:

**O_CREAT**
This option is used to create a message queue, and it requires two additional arguments: **mode**, which is of type **mode_t**, and **attr**, which is pointer to a **mq_attr** structure. If the pathname, **name**, has already been used to create a message queue that still exists, then this flag has no effect, unless combined with **O_EXCL** (see below). Otherwise, a message queue is created without any messages in it. The message queue’s user ID is set to the process’s effective user ID, and the message queue’s group ID is set to the process’s effective group ID. The message queue’s permission bits will be set to the value of **mode**, and modified by clearing all bits set in the file mode creation mask of the process (see **umask**(2)). “AND-NOT”) those already set in the file mode creation mask of the process.

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

**O_EXCL**
If both **O_EXCL** and **O_CREAT** are set, **mq_open**( ) will fail if the message queue **name** exists. The check for the existence of the message queue and the creation of the message queue if it does not exist are atomic with respect to other processes executing **mq_open**( ) naming the same **name** with both **O_EXCL** and **O_CREAT** set.

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

**RETURN VALUES**
Upon successful completion, **mq_open**( ) returns a message queue descriptor; otherwise the function returns (**mqd_t**)(-1) and sets **errno** to indicate the error condition.

**ERRORS**

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

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

**EINTR**
The **mq_open**( ) operation was interrupted by a signal.

3R-724  modified 19 Dec 1994
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.

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

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.

BUGS  
In Solaris 2.5, these functions always return -1 and set errno to ENOSYS, because this 
release does not support the Message Passing option. It is our intention to provide sup- 
port for these interfaces in future releases.
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 */
    ...
};

MT-LEVEL  MT-Safe

DESCRIPTION  mq_receive( ) 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 attribute of the message queue, the function fails and returns an error. Otherwise, the selected message is removed from the queue and copied to the buffer pointed to by msg_ptr.

If msg_prio is not NULL, the priority of the selected message is stored in the location referenced 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  EAGAIN    O_NONBLOCK was set in the message description associated with mqdes, and the specified message queue is empty.
EBADF     mqdes is not a valid message queue descriptor open for reading.
EMSGSIZE  msg_len is less than the message size attribute of the message queue.
EINTR     mq_receive() operation was interrupted by a signal.

3R-726       modified 19 Aug 1993
ENOSYS mq_receive() is not supported by this implementation.

SEE ALSO mq_open(3R), mq_send(3R), mq_setattr(3R)

BUGS In Solaris 2.5, these functions always return −1 and set errno to ENOSYS, because this release does not support the Message Passing option. It is our intention to provide support for these interfaces in future releases.
NAME  mq_send – send a message to a message queue

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

MT-LEVEL  MT-Safe

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. |
| ENOSYS | mq_send() is not supported by this implementation. |

**SEE ALSO**
mq_open(3R), mq_receive(3R), mq_setattr(3R), sysconf(3C)

**BUGS**
In Solaris 2.5, these functions always return −1 and set errno to ENOSYS, because this release does not support the Message Passing option. It is our intention to provide support for these interfaces in future releases.

modified 19 Aug 1993
NAME
mq_setattr, mq_getattr – set/get message queue attributes

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

MT-LEVEL
MT-Safe

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.

3R-730 modified 19 Aug 1993
mq_setattr(), if successful, also changes the attributes of the message queue as specified.

**ERRORS**
- **EBADF**
  - *mqdes* is not a valid message queue descriptor.
- **ENOSYS**
  - *mq_setattr()* and *mq_getattr()* are not supported by this implementation.

**SEE ALSO**
- mq_open(3R), mq_receive(3R), mq_send(3R)

**BUGS**
In Solaris 2.5, these functions always return −1 and set *errno* to ENOSYS, because this release does not support the Message Passing option. It is our intention to provide support for these interfaces in future releases.

modified 19 Aug 1993
NAME  mq_unlink – remove a message queue

SYNOPSIS  cc [ flag ... ] file ... -lposix4 [ library ... ]
#include <mqueue.h>
int mq_unlink(const char *name);

MT-LEVEL  MT-Safe

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.

SEE ALSO  mq_close(3R), mq_open(3R)

BUGS  In Solaris 2.5, these functions always return -1 and set errno to ENOSYS, because this
release does not support the Message Passing option. It is our intention to provide sup-
port for these interfaces in future releases.
NAME
msync – synchronize memory with physical storage

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

int msync(caddr_t addr, size_t len, int flags);

DESCRIPTION
The function msync() writes all modified copies of pages over the range \([addr, addr + len)\)
to their backing storage locations. msync() optionally invalidates any copies so that
further references to the pages will be obtained by the system from their backing storage
locations. The backing storage for a modified MAP_SHARED mapping is the file the page
is mapped to; the backing storage for a modified MAP_PRIVATE mapping is its swap
area.

flags is a bit pattern built from the following values:

- MS_ASYNC: perform asynchronous writes
- MS_SYNC: perform synchronous writes
- MS_INVALIDATE: invalidate mappings

If MS_ASYNC is set, msync() returns immediately once all write operations are
scheduled; if MS_SYNC is set, msync() does not return until all write operations are com-
pleted.

MS_INVALIDATE invalidates all cached copies of data in memory, so that further refer-
ences to the pages will be obtained by the system from their backing storage locations.

RETURN VALUES
Upon successful completion, the function msync() returns 0; otherwise, it returns −1 and
sets errno to indicate the error.

ERRORS
- 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: addr is not a multiple of the page size as returned by sysconf(3C).
  flags 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.

SEE ALSO
mmap(2), memcntl(2), sysconf(3C)
NOTES

\texttt{msync()} should be used by programs that require a memory object to be in a known state, such as in building transaction facilities.
NAME
mutex, 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);

MT-LEVEL
MT-Safe

DESCRIPTION
Mutual exclusion locks (mutexes) prevent multiple threads from simultaneously executing
critical sections of code which access shared data (i.e., 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 initial-
ized with all zeros and its scope is set to be within the calling process. For POSIX porta-
bility of statically allocated mutexes, use the pthread_mutex_initializer macro (see
below).

modified 30 Jun 1995
3T-735
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 `pthread_mutex_init()` with the appropriate attribute that indicates inter-process use.

### 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).

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

```c
3T-736
```

modified 30 Jun 1995
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`.

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):

```c
mutex_t mp;
mutex_init(&mp, NULL, NULL);
OR
mutex_init(&mp, USYNC_THREAD, NULL);
```
mutex (3T) Threads Library SunOS 5.5

mutex_t mp = DEFAULTMUTEX;
OR
mutex_t mp;

mp = malloc(1, sizeof (mutex_t));
OR
mutex_t mp;

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.

`pthread_mutex_unlock()` or `mutex_unlock()` are called by the owner of the mutex object referenced by `mp` to release it. The mutex must be locked and the calling thread must be the one that last locked the mutex (the owner). If there are threads blocked on the mutex

3T-738 modified 30 Jun 1995
object referenced by *mp* when *pthread_mutex_unlock()* is called, the *mp* is freed, and the
scheduling policy will determine which thread gets the mutex. If the calling thread is not
the owner of the lock, no error status is returned, and the behavior of the program is
undefined.

**Destroy**

Either *pthread_mutex_destroy()* or *mutex_destroy()* destroys the mutex object refer-
enced 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.

*pthread_mutex_init()* or *mutex_init()* fails and returns the corresponding value if any of
the following conditions are detected:

**EINVAL**

The value specified by *mp* or *attr* is invalid.

*pthread_mutex_trylock()* or *mutex_trylock()* fails and returns the corresponding value
if any of the following conditions occur:

**EBUSY**

The mutex pointed to by *mp* was already locked.

**SEE ALSO**

mmap(2), pthread_create(3T), pthread_mutexattr_init(3T)

**EXAMPLES**

The following example uses one global mutex as a gate-keeper to permit each thread
exclusive sequential access to the code within the user-defined function
"change_global_data." This type of synchronization will protect the state of shared data,
but it also prohibits parallelism.

/* cc thisfile.c -lthread */

#include <stdio.h>
#include <thread.h>

#define NUM_THREADS 12

void *change_global_data(void *);

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 *null) {
    static mutex_t Global_mutex;
    static int Global_data = 0;
    mutex_lock(&Global_mutex);
    Global_data++;
    sleep(1);
    printf("%d is global data\n",Global_data);
    mutex_unlock(&Global_mutex);
    return NULL;
}

The previous example, the mutex, the code it owns, and the data it protects was enclosed in one function. The next example uses C++ features to accommodate many functions that use just one mutex to protect one data:

/* CC thisfile.c -lthread use C++ to compile*/
#define _REENTRANT
#include <stdlib.h>
#include <stdio.h>
#include <thread.h>
#include <errno.h>
#include <iostream.h>
#define NUM_THREADS 16
#define change_global_data(void *); /* 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;
}

int Mutected::subtract_from_global_data() {

3T-740 modified 30 Jun 1995
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 subtract */
    cout << Mutected::subtract_from_global_data() << endl;
else
    cout << Mutected::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, or USYNC_PROCESS, for
Solaris threads. (see mutex_init(3T) and pthread_mutexattr_init(3T)). One process ini-
tializes the process-shared mutex and writes it to a file to be mapped into memory by all
cooperating processes (see mmap(2)). Afterwards, other independent processes can run
the same program (whether concurrently or not) and share mutex-protected data.

/* cc thisfile.c -lthread */
/* To execute, run the command line "a.out 0 & a.out 1" */
#define _REENTRANT
#include <sys/types.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <thread.h>
#define INTERPROCESS_FILE "ipc-sharedfile"
#define NUM_ADDTHREADS 12
#define NUM_SUBTRACTTHREADS 10

modified 30 Jun 1995 3T-741
#define INCREMENT '0'
#define DECREMENT '1'
typedef struct {
    mutex_t Interprocess_mutex;
    int Interprocess_data;
} buffer_t;
buffer_t *buffer;

void *add_interprocess_data(), *subtract_interprocess_data();
void create_shared_memory(), test_argv();
int zeroed[sizeof(buffer_t)];
int ipc_fd, i=0;

void
main(int argc, char * argv[]) {
    test_argv(argv[1]);
    switch (argv[1]) {
    case INCREMENT:
        create_shared_memory();
        ipc_fd = open(INTERPROCESS_FILE, O_RDWR);
        buffer = (buffer_t *)mmap(NULL, sizeof(buffer_t),
            PROT_READ|PROT_WRITE, MAP_SHARED, ipc_fd, 0);
        buffer->Interprocess_data = 0;
        mutex_init(&buffer->Interprocess_mutex, USYNC_PROCESS, 0);
        for (i=0; i< NUM_ADDTHREADS; i++)
            thr_create(NULL, 0, add_interprocess_data, argv[1],
                0, NULL);
        break;
    case DECREMENT:
        while((ipc_fd = open(INTERPROCESS_FILE, O_RDWR)) == -1)
            sleep(1);
        buffer = (buffer_t *)mmap(NULL, sizeof(buffer_t),
            PROT_READ|PROT_WRITE, MAP_SHARED, ipc_fd, 0);
        for (i=0; i< NUM_SUBTRACTTHREADS; i++)
            thr_create(NULL, 0, subtract_interprocess_data, argv[1],
                0, NULL);
        break;
    } /* end switch */

    while ((thr_join(NULL,NULL,NULL) == 0));
} /* end main */
void *add_interprocess_data(char argv_1[]) {  
    mutex_lock(&buffer->Interprocess_mutex);  
    buffer->Interprocess_data++;  
    sleep(2);  
    printf("%d is add-interprocess data, and %c is argv1\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 commandline  
  
    a.out 0 & a.out 1  

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.
NAME    nanosleep – high resolution sleep

SYNOPSIS    cc [ flag ... ] file ... -lposix4 [ library ... ]
#include <time.h>
int nanosleep(const struct timespec *rqtp, struct timespec *rmtp);

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

MT-LEVEL    MT-Safe

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 multiple of the sleep resolution or because of the scheduling of other activity by the system. Except for the case of being interrupted by a signal, the suspension time will not be less than the time specified by 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 rmtp is non-NULL, the timespec structure referenced by rmtp 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.

SEE ALSO    sleep(3C)

modified 19 Aug 1993
NAME

ndbm, dbm_clearerr, dbm_close, dbm_delete, dbm_error, dbm_fetch, dbm_firstkey, dbm_nextkey, dbm_open, dbm_store – data base subroutines

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(char *file, int flags, int mode);
int dbm_store(DBM *db, datum key, datum content, int flags);

MT-LEVEL
Unsafe

DESCRIPTION

These functions maintain key/content pairs in a data base. The functions will handle very large (a billion blocks) data base and will access a keyed item in one or two file system accesses. This package replaces the earlier dbm(3B) library, which managed only a single data base.

keys and contents 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 data base can be accessed, it must be opened by dbm_open(). This will open and/or create the files file.dir and file.pag depending on the flags parameter (see open(2)). A data base is closed by calling dbm_close().

Once open, the data stored under a key is accessed by dbm_fetch() and data is placed under a key by dbm_store(). The flags field can be either DBM_INSERT or DBM_REPLACE. DBM_INSERT will only insert new entries into the data base and will not change an existing entry with the same key. DBM_REPLACE will replace an existing entry if it has the same key. A key (and its associated contents) is deleted by dbm_delete(). A linear pass through all keys in a data base may be made, in an (apparently) random order, by use of dbm_firstkey() and dbm_nextkey(). dbm_firstkey() will return the first key in the data base. dbm_nextkey() will return the next key in the data base. This code will traverse the data base:

for (key = dbm_firstkey(db); key.dptr != NULL; key = dbm_nextkey(db))

dbm_error() returns non-zero when an error has occurred reading or writing the data base. dbm_clearerr() resets the error condition on the named data base.
RETURN VALUES
All functions that return an `int` indicate errors with negative values. A return value of 0 indicates no error. Routines that return a `datum` indicate errors with a NULL (0) `dptr`. If `dbm_store()` is called with a `flags` value of `DBM_INSERT` and finds an existing entry with the same key, it returns 1.

EXAMPLES
The following example stores and retrieves a phone number, using the name as the key. Note that this example does not include error checking.

```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);
    (void) printf("Name: %s, Phone Number: %s0, name.dptr, get_phone_no.dptr);

    /* Close the database */
    dbm_close(db);
    return (0);
}
```

SEE ALSO
`ar(1)`, `cat(1)`, `cp(1)`, `tar(1)`, `open(2)`, `dbm(3B)`, `netconfig(4)`

NOTES
The .pag file will contain holes so that its apparent size may be larger than its actual content. Older versions of the UNIX operating system may create real file blocks for these holes when touched. These files cannot be copied by normal means (`cp(1)`, `cat(1)`, `tar(1)`, `ar(1)`) without filling in the holes.

dptr pointers returned by these subroutines point into static storage that is changed by subsequent calls.

modified 5 Apr 1994
The sum of the sizes of a key/content pair must not exceed the internal block size (currently 1024 bytes). Moreover all key/content pairs that hash together must fit on a single block. `dbm_store()` will return an error in the event that a disk block fills with inseparable data.

`dbm_delete()` does not physically reclaim file space, although it does make it available for reuse.

The order of keys presented by `dbm_firstkey()` and `dbm_nextkey()` depends on a hashing function.

There are no interlocks and no reliable cache flushing; thus concurrent updating and reading is risky.

The database files (`file.dir` and `file.pag`) are binary and are architecture-specific (for example, they depend on the architecture’s byte order.) These files are not guaranteed to be portable across architectures.
NAME
netdir, netdir_getbyname, netdir_getbyaddr, netdir_free, netdir_options, taddr2uaddr,
uaddr2taddr, netdir_pererror, netdir_spererror, netdir_mergeaddr – generic transport
name-to-address translation

SYNOPSIS
#include <netdir.h>
int netdir_getbyname(const struct netconfig *config,
const struct nd_hostserv *service, struct nd_addrlist **addrs);
int netdir_getbyaddr(const struct netconfig *config,
struct nd_hostservlist **service, const struct netbuf *netaddr);
void netdir_free(void *ptr, const int struct_type);
int netdir_options(const struct netconfig *config, const int option,
const int fildes, char *point_to_args);
char *taddr2uaddr(const struct netconfig *config, const struct netbuf *addr);
struct netbuf *uaddr2taddr(const struct netconfig *config, const char *uaddr);
void netdir_pererror(char *s);
char *netdir_spererror(void);

MT-LEVEL
MT-Safe

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_addrlist structure. The nd_hostserv structure contains the fol-
lowing members:

char *h_host; /* host name */
char *h_serv; /* service name */

The nd_addrlist structure contains the following members:

int n_cnt; /* number of addresses */
struct netbuf *n_addr;

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.

modified 28 Jun 1994
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. fd is the associated file descriptor. option, fd, 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. *fdles* 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 *fdles*.

**ND_SET_RESERVEDPORT**  
Allows the application to bind to a reserved port, if that concept exists for the transport provider. *fdles* is an unbound file descriptor into the transport. If *pointer_to_args* is NULL, *fdles* 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. *fdles* 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. *fdles* is not used. *pointer_to_args* is a pointer to a struct nd_mergearg, which has the following members:

- `char s_uaddr;` /* server’s universal address */  
- `char c_uaddr;` /* client’s universal address */  
- `char m_uaddr;` /* the result */

If *s_uaddr* is something like 0.0.0.1.12, and, if the call is successful, *m_uaddr* will be set to something like 192.11.109.89.1.12. For most transports, *m_uaddr* is exactly what *s_uaddr* is.

**RETURN VALUES**  
The netdir_perror() routine prints an error message on the standard output stating why one of the name-to-address mapping routines failed. The error message is preceded by the string given as an argument.

The netdir_sperror() routine returns a string containing an error message stating why one of the name-to-address mapping routines failed.

netdir_sperror() returns a pointer to a buffer which contains the error message string. This buffer is overwritten on each call. In multithreaded applications, this buffer is implemented as thread-specific data.
SEE ALSO  gethostname(3C), getnetconfig(3N), getnetpath(3N), netconfig(4)
NAME
nice – change priority of a process

SYNOPSIS
/usr/ucb/cc [ flag ...] file ...
int nice(increment)
int increment;

DESCRIPTION
The scheduling priority of the process is augmented by increment. 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 priority is limited to the range −20 (most urgent) to 20 (least). Requests for values above or below these limits result in the scheduling priority being set to the corresponding limit.

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 increment 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 platforms. Use of these interfaces with any of the system libraries or in multi-thread applications is unsupported.

modified 12 Feb 1993
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);

void db_free_result(db_result *);

MT-LEVEL

Unsafe

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+.

3N-754 modified 15 Jul 1993
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.
**db_checkpoint()** organizes the contents of the table in a more efficient manner. Checkpointing may mean different things to different types of databases. It does not affect the logical contents of the table — operations and queries should return the same result before and after a checkpoint. For example, in a log-based system, checkpointing may mean incorporating log entries of updates accumulated since the previous checkpoint into the table.

**db_free_result()** frees up the space allocated by various functions listed on this manual page that return a **db_result** structure.

**db_standby()** is an advisory call to the database manager. This call informs the database that activity has slowed down and it can free up unnecessary resources such as file descriptors.

**PROGRAMMING**

Most of the routines in this library use an NIS+ name to identify the object that the user desires. The name must be in canonical form before being passed to the database because one server may be serving several namespaces and discrimination of the requested objects is accomplished by comparing the domain names.

**DIAGNOSTICS**

**DB_SUCCESS** The query or operation completed successfully and returned status.

**DB_NOTFOUND** The name or entry that was named in the argument did not exist.

**DB_NOTUNIQUE** An attempt was made to remove an entry from a table that is not uniquely specified.

**DB_BADQUERY** The query that was submitted to the database was invalid (for example, it might name some nonexistent fields).

**DB_BADTABLE** The table was corrupted.

**DB_BADOBJECT** The fields of the object does not conform to the fields of the table to which it is being added.

**DB_MEMORY_LIMIT** There is insufficient memory to complete the operation requested.

**DB_STORAGE_LIMIT** There is insufficient file storage available to complete the operation requested.

**DB_INTERNAL_ERROR** An internal error was encountered during the execution of the operation requested (either a programming error or an unrecoverable exception).

**SEE ALSO** rpc.nisd(1M), nis_objects(3N), nisfiles(4)
NAME  nis_error, nis_sperrno, nis_perror, nis_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);
char *nis_sperror(const nis_error status, const char *label);

MT-LEVEL  Safe

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).
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 pre-
ferred interface, since it is suitable for single-threaded and multi-threaded programs.

SEE ALSO  niserror(1), string(3C), syslog(3)

NOTES  When compiling multithreaded applications, see Intro(3), Notes On Multithread Applica-
tions, for information about the use of the _REENTRANT flag.
NAME
nis_groups, nis_ismember, nis_addmember, nis_removemember, nis_creategroup,
nis_destroygroup, nis_verifygroup, nis_print_group_entry, nis_map_group,
__nis_map_group – NIS+ group manipulation functions

SYNOPSIS
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);

MT-LEVEL
MT-Safe

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 10 May 1993
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.

SEE ALSO

nisgrpadm(1), nis_objects(3N)

NOTES

These functions only accept fully-qualified NIS+ names.

A group is represented by a NIS+ object (see nis_objects(3N)) with a variant part that is defined in the group_obj structure. It contains the following fields:

```c
u_long gr_flags; /* Interpretation Flags
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);

MT-LEVEL  MT-Safe

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 qualified name for the host and is either the value returned by the gethostname(3C) function or, if the host name is only partially qualified, the concatenation of that value and the name of the NIS+ directory. Note that if a machine’s name and address cannot be found in the local NIS+ directory, its hostname must be fully qualified.
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 qualified, the value returned by nis_local_directory() will be concatenated to it.

SEE ALSO  nisdefaults(1), nisaddcred(1M), sysinfo(2), gethostname(3C), nis_names(3N), nis_objects(3N)
NAME  nis_names, nis_lookup, nis_add, nis_remove, nis_modify, nis_freeresult – NIS+
namespace functions

SYNOPSIS  cc [ flag . . . ] file. . . -l nis [ 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);

MT-LEVEL  MT-Safe

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, respec-
tively. 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() func-
tion 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 follow-
ing 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

modified 10 May 1993 3N-763
either a object is found that is not a LINK type object, or the library has followed 16 links.

**HARD_LOOKUP** When specified, the client library will retry the lookup until it is answered by a server. Using this flag will cause the library to block until at least one NIS+ server is available. If the network connectivity is impaired, this can be a relatively long time.

**NO_CACHE** When specified, the client library will bypass any object caches and will get the object from either the master NIS+ server or one of its replicas.

**MASTER_ONLY** When specified, the client library will bypass any object caches and any domain replicas and fetch the object from the NIS+ master server for the object’s domain. This insures that the object returned is up to date at the cost of a possible performance degradation and failure if the master server is unavailable or physically distant.

**EXPAND_NAME** When specified, the client library will attempt to expand a partially qualified name by calling the function `nis_getnames()` (see `nis_subr(3N)`) which uses the environment variable `NIS_PATH`.

The status value may be translated to ascii text using the function `nis_sperrno()` (see `nis_error(3N)`).

On return, the `objects` array in the result will contain one and possibly several objects that were resolved by the request. If the FOLLOW_LINKS flag was present, on success the function could return several entry objects if the link in question pointed within a table. If an error occurred when following a link, the objects array will contain a copy of the link object itself.

The function `nis_add()` will take the object `obj` and add it to the NIS+ namespace with the name `name`. This operation will fail if the client making the request does not have the create access right for the domain in which this object will be added. The parameter `name` must contain a fully qualified NIS+ name. The object members `zo_name` and `zo_domain` will be constructed from this name. This operation will fail if the object already exists. This feature prevents the accidental addition of objects over another object that has been added by another process.

The function `nis_remove()` will remove the object with name `name` from the NIS+ namespace. The client making this request must have the destroy access right for the domain in which this object resides. If the named object is a link, the link is removed and not the object that it points to. If the parameter `obj` is not NULL, it is assumed to point to a copy of the object being removed. In this case, if the object on the server does not have the same object identifier as the object being passed, the operation will fail with the NIS_NOTSAMEOBJ error. This feature allows the client to insure that it is removing the desired object. The parameter `name` must contain a fully qualified NIS+ name.

The function `nis_modify()` will modify the object named by `name` to the field values in the object pointed to by `obj`. This object should contain a copy of the object from the name space that is being modified. This operation will fail with the error NIS_NOTSAMEOBJ if the object identifier of the passed object does not match that of the object being modified.
in the namespace.

Note: Normally the contents of the member zo_name in the nis_object structure would be constructed from the name passed in the name parameter. However, if it is non-NULL the client library will use the name in the zo_name member to perform a rename operation on the object. This name must not contain any unquoted ‘.’(dot) characters. If these conditions are not met the operation will fail and return the NIS_BADNAME error code.

Results

These functions return a pointer to a structure of type nis_result:

```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 time.
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.

\textbf{RETURN VALUES} The client library can return a variety of error returns and diagnostics. The more salient ones are documented below.

\textbf{NIS\textunderscore SUCCESS} The request was successful.

\textbf{NIS\textunderscore S\textunderscore 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\textunderscore CACHE} when you call the lookup function.

\textbf{NIS\textunderscore NOTFOUND} The named object does not exist in the namespace.

\textbf{NIS\textunderscore CACHEEXPIRED} The object returned came from an object cache that has expired. The time to live value has gone to zero and the object may have changed. If the flag \texttt{NO\textunderscore CACHE} was passed to the lookup function then the lookup function will retry the operation to get an unexpired copy of the object.

\textbf{NIS\textunderscore 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\textunderscore LOOKUP} flag.

\textbf{NIS\textunderscore UNKNOWNOBJ} The object returned is of an unknown type.

\textbf{NIS\textunderscore 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 it’s internal state. And in the case of \texttt{nis\textunderscore list()} if the client specifies a callback and the server does not have enough resources to handle the callback.

\textbf{NIS\textunderscore 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.

\textbf{NIS\textunderscore NOT\textunderscore 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.

\textbf{NIS\textunderscore NOMEMORY} Generally a fatal result. It means that the service ran out of
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().

SEE ALSO
nis_error(3N), nis_objects(3N), nis_server(3N), nis_subr(3N), nis_tables(3N)

NOTES
You cannot modify the name of an object if that modification would cause the object to reside in a different domain.
You cannot modify the schema of a table object.
NAME
nis_objects – NIS+ object formats

SYNOPSIS
cc [ flag … ] file… -lnsl [ library… ]
/usr/include/rpcsvc/nis_objects.x

DESCRIPTION
Common Attributes

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 nis_object structure contains the following
members:

```c
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;
};
```

In this structure, the first member zo_oid, is a 64 bit number that uniquely identifies this
instance of the object on this server. This member is filled in by the server when the
object is created and changed by the server when the object is modified. When used in
conjunction with the object’s name and domain it uniquely identifies the object in the
entire NIS+ namespace.

The second member, 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.

zo_domain contains the name of the NIS+ domain to which this object belongs. This
information is useful when tracking the parentage of an object from a cache. When used in
conjunction with the members zo_name and zo_oid, it uniquely identifies an object.
This makes it possible to always reconstruct the name of an object by using the code fragment

```c
sprintf(buf,"%s.%s", obj→zo_name, obj→zo_domain);
```

The zo_owner and zo_group members contain the NIS+ names of the object’s principal
owner and group owner, respectively. Both names must be NIS+ fully qualified names.
However, neither name can be used directly to identify the object they represent. This
stems from the condition that NIS+ uses itself to store information that it exports.
The \texttt{zo\_owner} member contains a fully qualified NIS+ name of the form \texttt{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 \texttt{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 \texttt{netname} of the user making the request. This \texttt{netname} will be of the form \texttt{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

\texttt{group.domain}.

which the server then maps into a name of the form

\texttt{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 \texttt{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 \texttt{zo\_owner} name in the object. See \texttt{nis\_groups(3N)} for more details.

The \texttt{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 \texttt{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 \texttt{zo\_ttl}. Then each time the cached object is used, the time in \texttt{zo\_ttl} is compared with the current time. If the current time is later than the time in \texttt{zo\_ttl} the object is said to have expired and the cached copy should not be used.

modified 10 May 1993
Setting the TTL is somewhat of an art. You can think of it as the “half life” of the object, or half the amount of time you believe will pass before the object changes. The benefit of setting the ttl to a large number is that the object will stay in a cache for long periods of time. The problem with setting it to a large value is that when the object changes it will take a long time for the caches to flush out old copies of that object. The problems and benefits are reversed for setting the time to a small value. Generally setting the value to 43200 (12 hrs) is reasonable for things that change day to day, and 3024000 is good for things that change week to week. Setting the value to 0 will prevent the object from ever being cached since it would expire immediately.

The zo_data member is a discriminated union with the following members:

```c
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;
}
```

The union is discriminated based on the type value contained in zo_type. There are 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
};
```

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 {
```
typedef struct directory_obj directory_obj;

The main structure contains five primary members: do_name, do_type, do_servers, do_ttl, and do_armask. The information in the do_servers structure is sufficient for the client library to create a network connection with the named server for the directory.

The do_name member contains the name of the directory or domain represented in a format that is understandable by the type of nameservice serving that domain. In the case of NIS+ domains, this is the same as the name that can be composed using the zo_name and zo_domain members. For other name services, this name will be a name that they understand. For example, if this were a directory object describing an X.500 namespace that is “under” the NIS+ directory eng.sun.com., this name might contain “/C=US, /O=Sun Microsystems, /OU=Engineering/”. The type of nameservice that is being described is determined by the value of the member do_type.

The do_servers structure contains two members. do_servers_val is an array of nis_server structures; do_servers_len is the number of cells in the array. The nis_server structure is designed to contain enough information such that machines on the network providing name services can be contacted without having to use a name service. In the case of NIS+ servers, this information is the name of the machine in name, its public key for authentication in pkey, and a variable length array of endpoints, each of which describes the network endpoint for the rpcbind daemon on the named machine. The client library uses the addresses to contact the server using a transport that both the client and server can communicate on and then queries the rpcbind daemon to get the actual transport address that the server is using.

Note that the first server in the do_servers list is always the master server for the directory. The key_type field describes the type of key stored in the pkey netobj (see /usr/include/rpc/xdr.h for a definition of the network object structure). Currently supported types are NIS_PK_NONE for no public key and NIS_PK_DH for a Diffie-Hellman type public key.

The do_ttl member contains a copy of the zo_ttl member from the common attributes. This is 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 two-tiered. If the directory object itself grants a given access right (using the zo_access member in the nis_object structure representing the directory), then all objects within the directory are allowed that access. Otherwise, the do_armask structure is examined to see...
if the access is allowed specifically for that type of structure. This allows the administrator of a namespace to set separate policies for different object types, for example, one policy for the creation of tables and another policy for the creation of other directories. See nis+(1) for more details.

**Link Objects**

Link objects provide a means of providing aliases or symbolic links within the namespace. Their variant part is defined as follows.

```c
struct link_obj {
    zotypes   li_rtype;
    struct {
        u_int li_atrs_len;
        nis_attr *li_atrs_val;
    } li_atrs;
    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_atrs** 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:

modified 10 May 1993

3N-773
#define TA_BINARY 1
#define TA_CRYPT 2
#define TA_XDR 4
#define TA_SEARCHABLE 8
#define TA_CASE 16

struct table_col {
    char *tc_name;
    u_long tc_flags;
    u_long tc_rights;
}

typedef struct table_col table_col;

struct table_obj {
    char *ta_type;
    u_int ta_maxcol;
    u_char ta_sep;
    struct {
        u_int ta_cols_len;
        table_col *ta_cols_val;
    } ta_cols;
    char *ta_path;
}

The ta_type member contains a string that identifies the type of entries in this table. NIS+ does not enforce any policies as to the contents of this string. However, when entries are added to the table, the NIS+ service will check to see that they have the same “type” as the table as specified by this member.

The structure ta_cols contains two members. ta_cols_val is an array of table_col structures. The length of the array depends on the number of columns in the table; it is defined when the table is created and is stored in ta_cols_len. ta_maxcol also contains the number of columns in the table and always has the same value as ta_cols_len. Once the table is created, this length field cannot be changed.

The ta_sep character is used by client applications that wish to print out an entry from the table. Typically this is either space (’ ’) or colon (’:’).

The ta_path string defines a concatenation path for tables. This string contains an ordered list of fully qualified table names, separated by colons, that are to be searched if a search on this table fails to match any entries. This path is only used with the flag FOLLOW_PATH with a nis_list() call. See nis_tables(3N) for information on these flags.

In addition to checking the type, the service will check that the number of columns in an entry is the same as those in the table before allowing that entry to be added.

Each column has associated with it a name in tc_name, a set of flags in tc_flags, and a set of access rights in tc_rights. The name should be indicative of the contents of that column.
The TA_BINARY flag indicates that data in the column is binary (rather than text). Columns that are searchable cannot contain binary data. The TA_CRYPT flag specifies that the information in this column should be encrypted prior to sending it over the network. This flag has no effect in the export version of NIS+. The TA_XDR flag is used to tell the client application that the data in this column is encoded using the XDR protocol. The TA_BINARY flag must be specified with the XDR flag. Further, by convention, the name of a column that has the TA_XDR flag set is the name of the XDR function that will decode the data in that column.

The TA_SEARCHABLE flag specifies that values in this column can be searched. Searchable columns must contain textual data and must have a name associated with them. The flag TA_CASE specifies that searches involving this column ignore the case of the value in the column. At least one of the columns in the table should be searchable. Also, the combination of all searchable column values should uniquely select an entry within the table.

**Entry Objects**

Entry objects are stored in tables. The structure used to define the entry data is as follows.

```c
#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);

MT-LEVEL MT-Safe

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.

SEE ALSO nislog(1M), nis_names(3N), nis_tables(3N), nisfiles(4)
NAME

nis_server, nis_mkdir, nis_rmdir, nis_servstate, 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 **machines);

MT-LEVEL

MT-Safe

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.
nis_getservlist() returns a null terminated list of nis_server structures that represent the list of servers that serve the domain named dirname. Servers from this list can be used when calling functions that require the name of a NIS+ server. For a description of the nis_server structure, refer to nis_objects(3N). nis_freeservlist() frees the list of servers returned by nis_getservlist(). Note that this is the only legal way to free that list.

SEE ALSO nisstat(1M), nis_names(3N), nis_objects(3N), nis_subr(3N)
NAME
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);
name_pos nis_dir_cmp(const nis_name n1, const nis_name n2);
nis_object *nis_clone_object(const nis_object *src, nis_object *dest);
void nis_destroy_object(nis_object *obj);
void nis_print_object(const nis_object *obj);

MT-LEVEL
Safe

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().

3N-780 modified 10 May 1993
Though `nis_dir_cmp()` can be used to compare any two NIS+ names, it is used primarily to compare domain names. This comparison is done in a case independent fashion, and the results are an enum of type `name_pos`. When the names passed to this function are identical, the function returns a value of `SAME_NAME`. If the name `n1` is a direct ancestor of name `n2`, then this function returns the result `HIGHER_NAME`. Similarly, if the name `n1` is a direct descendant of name `n2`, then this function returns the result `LOWER_NAME`. When the name `n1` is neither a direct ancestor nor a direct descendant of `n2`, as it would be if the two names were siblings in separate portions of the namespace, then this function returns the result `NOT_SEQUENTIAL`. Finally, if either name cannot be parsed as a legitimate name then this function returns the value `BAD_NAME`.

The second set of functions, consisting of `nis_clone_object()` and `nis_destroy_object()`, are used for manipulating objects. `nis_clone_object()` creates an exact duplicate of the NIS+ object `src`. If the value of `dest` is non-null, it creates the clone of the object into this object structure and allocate the necessary memory for the variable length arrays. If this parameter is null, a pointer to the cloned object is returned. Refer to `nis_objects(3N)` for a description of the `nis_object` structure.

`nis_destroy_object()` can be used to destroy an object created by `nis_clone_object()`. This will free up all memory associated with the object and free the pointer passed. If the object was cloned into an array (using the `dest` parameter to `nis_clone_object()`) then the object cannot be freed with this function. Instead, the function `xdr_free(xdr_nis_object, dest)` must be used.

`nis_print_object()` prints out the contents of a NIS+ object structure on the standard output. Its primary use is for debugging NIS+ programs.

### ENVIRONMENT

**NIS_PATH**

This variable overrides the default NIS+ directory search path used by `nis_getnames()`. It contains an ordered list of directories separated by `:` (colon) characters. The `$` (dollar sign) character is treated specially. Directory names that end in `$` have the default domain appended to them, and a `$` by itself is replaced by the list of directories between the default domain and the global root that are at least two levels deep. The default NIS+ directory search path is `$`.

### SEE ALSO

`nis_names(3N), nis_objects(3N), nis_tables(3N)`

### NOTES

`nis_leaf_of()`, `nis_name_of()` and `nis_clone_object()` return their results as thread-specific data in multithreaded applications.
NAME
nis_tables, nis_list, nis_add_entry, nis_remove_entry, nis_modify_entry, nis_first_entry, nis_next_entry - NIS+ table functions

SYNOPSIS
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);

MT-LEVEL
MT-Safe

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.

MASTER_ONLY
This flag is even stronger than NO_CACHE in that it specifies that the client library should only get its information from the master server for a particular table. This guarantees that the information will be up to date. However, there may be severe performance penalties associated with contacting the master server directly on large networks. When used in conjunction with the HARD_LOOKUP flag, this will block the list operation until the master server is up and available.

EXPAND_NAME
When specified, the client library will attempt to expand a partially qualified name by calling nis_getnames() (see nis_local_names(3N)) which uses the environment variable NIS_PATH.

RETURN_RESULT
This flag is used to specify that a copy of the returning object be returned in the nis_result structure if the operation was successful.

The third parameter to nis_list(), callback, is an optional pointer to a function that will process the ENTRY type objects that are returned from the search. If this pointer is NULL, then all entries that match the search criteria are returned in the nis_result structure, otherwise this function will be called once for each entry returned. When called, this function should return 0 when additional objects are desired and 1 when it no longer wishes to see any more objects. The fourth parameter, userdata, is simply passed to callback function along with the returned entry object. The client can use this pointer to pass

modified 15 Feb 1995 3N-783
state information or other relevant data that the callback function might need to process the entries.

**nis_add_entry()** will add the NIS+ object to the NIS+ `table_name`. The `flags` parameter is used to specify the failure semantics for the add operation. The default (`flags` equal 0) is to fail if the entry being added already exists in the table. The `ADD_OVERWRITE` flag may be used to specify that existing object is to be overwritten if it exists, (a modify operation) or added if it does not exist. With the `ADD_OVERWRITE` flag, this function will fail with the error **NIS_PERMISSION** if the existing object does not allow modify privileges to the client.

If the flag `RETURN_RESULT` has been specified, the server will return a copy of the resulting object if the operation was successful.

**nis_remove_entry()** removes the identified entry from the table or a set of entries identified by `table_name`. If the parameter `object` is non-null, it is presumed to point to a cached copy of the entry. When the removal is attempted, and the object that would be removed is not the same as the cached object pointed to by `object` then the operation will fail with an **NIS_NOTSAMEOBJ** error. If an object is passed with this function, the search criteria in name is optional as it can be constructed from the values within the entry. However, if no object is present, the search criteria must be included in the `name` parameter. If the `flags` variable is null, and the search criteria does not uniquely identify an entry, the **NIS_NOTUNIQUE** error is returned and the operation is aborted. If the `flags` 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`:

```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 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 dticks from the value in zticks will yield the time spent in the service code itself. Subtracting the sum of the values in zticks and aticks from the value in cticks will yield the time spent in the client library itself. Note: all of the tick times are measured in microseconds.

ERRORS The client library can return a variety of error returns and diagnostics. The more salient ones are documented below.

NIS_BADATTRIBUTE The name of an attribute did not match up with a named column in the table, or the attribute did not have an associated value.

NIS_BADNAME The name passed to the function is not a legal NIS+ name.

NIS_BADREQUEST A problem was detected in the request structure passed to the client library.

NIS_CACHEEXPIRED The entry returned came from an object cache that has 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_CBERROE An RPC error occurred on the server while it was calling back to the client. The transaction was aborted at that time and any unsent data was discarded.

NIS_CBRESULTS Even though the request was successful, all of the entries have been sent to your callback function and are thus not included in this result.

NIS_FOREIGNNS The name could not be completely resolved. When the name passed to the function would resolve in a namespace that is outside the NIS+ name tree, this error is returned with a NIS+ object of type DIRECTORY. The returned object contains the type of namespace and contact information for a server within that namespace.

NIS_INVALIDOBJ The object pointed to by 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 LINK type object and the contents of the object pointed to an invalid name.

NIS_MODFAIL The attempted modification failed for some reason.
<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NIS_NAMEEXISTS</td>
<td>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.</td>
</tr>
<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().</td>
</tr>
<tr>
<td></td>
<td>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 action.</td>
</tr>
</tbody>
</table>
### NIS_RPCERROR
This fatal error indicates the RPC subsystem failed in some way. Generally there will be a `syslog(3)` message indicating why the RPC request failed.

### NIS_S_NOTFOUND
The named entry does not exist in the table, however not all tables in the path could be searched, so the entry may exist in one of those tables.

### NIS_S_SUCCESS
Even though the request was successful, a table in the search path was not able to be searched, so the result may not be the same as the one you would have received if that table had been accessible.

### NIS_SUCCESS
The request was successful.

### NIS_SYSTEMERROR
Some form of generic system error occurred while attempting the request. Check the `syslog(3)` record for error messages from the server.

### NIS_TOOMANYATTRS
The search criteria passed to the server had more attributes than the table had searchable columns.

### NIS_TRYAGAIN
The server connected to was too busy to handle your request. `add_entry()`, `remove_entry()`, and `modify_entry()` return this error when the master server is currently updating its internal state. It can be returned to `nis_list()` when the function specifies a callback and the server does not have the resources to handle callbacks.

### NIS_TYPENISMATCH
An attempt was made to add or modify an entry in a table, and the entry passed was of a different type than the table.

#### ENVIRONMENT
**NIS_PATH**
When set, this variable is the search path used by `nis_list()` if the flag `EXPAND_NAME` is set.

#### SEE ALSO
`niscat(1)`, `niserror(1)`, `nismatch(1)`, `nis_cachemgr(1M)`, `nis_error(3N)`, `nis_local_names(3N)`, `nis_names(3N)`, `nis_objects(3N)`, `syslog(3)`

#### WARNINGS
Use the flag `HARD_LOOKUP` carefully since it can cause the application to block indefinitely during a network partition.

#### NOTES
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 \texttt{nis\_first\_entry()} and \texttt{nis\_next\_entry()} to get their answers from only the master server.
NAME  nl_langinfo – language information

SYNOPSIS  #include <nl_types.h>
#include <langinfo.h>

char *nl_langinfo(nl_item item);

MT-LEVEL  Safe with exceptions

DESCRIPTION  nl_langinfo() returns a pointer to a null-terminated string containing information
relevant to a particular language or cultural area defined in the programs locale. The
manifest constant names and values of item are defined by <langinfo.h>.
Since yes and no strings are implemented using gettext(3I), link with the −lintl library.

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.

EXAMPLES  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.

SEE ALSO  gettext(3I), localeconv(3C), setlocale(3C), strftime(3C), langinfo(5), nl_types(5)

WARNINGS  The array pointed to by the return value should not be modified by the program. Subse-
quent calls to nl_langinfo() may overwrite the array.
This function is built upon the functions localeconv(), strftime(), and gettext(). Where
possible users are advised to use these interfaces to the required data instead of using
calls to nl_langinfo().

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.

3C-790  modified 22 Jan 1993
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)).

modified 3 May 1994  3B-791
NAME  nlist – get entries from name list

SYNOPSIS  cc [ flag ...] file ... -l elf [ library ...]
#include <nlist.h>
int nlist(const char *filename, struct nlist *nl);

MT-LEVEL  Safe

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.

SEE ALSO  cc(1B), elf(3E), kvm_nlist(3K), kvm_open(3K), a.out(4), ksysms(7D), mem(7D)
NAME
nlsgetcall – get client’s data passed via the listener

SYNOPSIS
#include <sys/tiuser.h>
struct t_call *nlsgetcall(int fildes);

MT-LEVEL
Unsafe

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 structure
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

SEE ALSO
nlsadmin(1M), getenv(3C), t_alloc(3N), t_connect(3N), t_error(3N), t_free(3N),
t_sync(3N)

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 22 Jan 1993

3N-793
NAME    nlsprovider – get name of transport provider

SYNOPSIS    char *nlsprovider(void);

MT-LEVEL    Unsafe

DESCRIPTION    nlsprovider() returns a pointer to a null terminated character string which contains the name of the transport provider as placed in the environment by the listener process. If the variable is not defined in the environment, a NULL pointer is returned. The environment variable is only available to server processes started by the listener process.

RETURN VALUES    If the variable is not defined in the environment, a NULL pointer is returned.

FILES    /usr/lib/libslan.a (7300)
          /usr/lib/libnls.a (3B2 Computer)
          /usr/lib/libnsl_s.a

SEE ALSO    nlsadmin(1M)

NOTES    This interface is unsafe in multithreaded applications. Unsafe interfaces should be called only from the main thread.
NAME  nlsrequest – format and send listener service request message

SYNOPSIS  
#include <listen.h>
int nlsrequest(int fildes, char *service_code);
extern int _nlslog, t_errno;
extern char *_nlsrmsg;

MT-LEVEL  Unsafe

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/libnsl_s.a

SEE ALSO  nlsadmin(1M), t_error(3N)

WARNINGS  nlsrequest() cannot always be certain that the remote server process has been successfully started. In this case, nlsrequest() returns with no indication of an error and the caller will receive notification of a disconnect event via a T_LOOK error before or during the first t_snd() or t_rcv() call.

modified 22 Jan 1993
NOTES

These interfaces are unsafe in multithreaded applications. Unsafe interfaces should be called only from the main thread.
NAME    offsetof – offset of structure member

SYNOPSIS  
#include <stddef.h>

size_t offsetof(type, member-designator);

MT-LEVEL  MT-Safe

DESCRIPTION  offsetof() is a macro defined in <stddef.h> which expands to an integral constant expression that has type size_t, the value of which is the offset in bytes, to the structure member (designated by member-designator), from the beginning of its structure (designated by type).
NAME  p2open, p2close – open, close pipes to and from a command

SYNOPSIS  cc [ flag … ] file … −Igen [ library … ]
           #include <libgen.h>
           int p2open(const char *cmd, FILE *fp[2]);
           int p2close(FILE *fp[2]);

MT-LEVEL  Unsafe

DESCRIPTION  p2open() forks and execs 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);
           }
## SEE ALSO
- `fclose(3S)`, `popen(3S)`, `setbuf(3S)`

## NOTES
Buffered writes on `fp[0]` can make it appear that the command is not listening. Judiciously placed `fflush()` calls or unbuffering `fp[0]` can be a big help; see `fclose(3S)`.

Many commands use buffered output when connected to a pipe. That, too, can make it appear as if things are not working.

Usage is not the same as for `popen()`, although it is closely related.
NAME
panel_above, panel_below – panels deck traversal primitives

SYNOPSIS
cc [ flag ... ] file ... -lpanel -lcurses [ library ... ]
#include <panel.h>
PANEL *panel_above(PANEL *panel);
PANEL *panel_below(PANEL *panel);

MT-LEVEL
Unsafe

DESCRIPTION
panel_above() returns a pointer to the panel just above 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.

SEE ALSO
curses(3X), panels(3X)

NOTES
These routines allow traversal of the deck of currently visible panels.
The header <panel.h> automatically includes the header <curses.h>. 
NAME
panel_move, move_panel – move a panels window on the virtual screen

SYNOPSIS
cc [ flag ... ] file ... -lpanel -lcurses [ library ... ]
#include <panel.h>
int move_panel(PANEL *panel, int starty, int startx);

MT-LEVEL
Unsafe

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.

SEE ALSO
curses(3X), panel_update(3X), panels(3X)

NOTES
For panels windows, use move_panel() instead of the mvwin() curses routine. Otherwise, update_panels() will not properly update the virtual screen.
The header <panel.h> automatically includes the header <curses.h>.

modified 22 Jan 1993
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);

MT-LEVEL | Unsafe

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.

SEE ALSO | curses(3X), panel_update(3X), panels(3X)

NOTES | The header <panel.h> automatically includes the header <curses.h>.
NAME panel_show, show_panel, hide_panel, panel_hidden – panels deck manipulation routines

SYNOPSIS cc [flag ...] file ... -lpanel -lcurses [library ...]
#include <panel.h>
int show_panel(PANEL *panel);
int hide_panel(PANEL *panel);
int panel_hidden(PANEL *panel);

MT-LEVEL Unsafe

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.

SEE ALSO curses(3X), panel_update(3X), panels(3X)

NOTES The header <panel.h> automatically includes the header <curses.h>.
NAME  panel_top, top_panel, bottom_panel – panels deck manipulation routines

SYNOPSIS  cc [ flag . . . ] file . . . -lpanel -lcurses [ library . . ]
           #include <panel.h>
           int top_panel(PANEL *panel);
           int bottom_panel(PANEL *panel);

MT-LEVEL  Unsafe

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 deck 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.

SEE ALSO  curses(3X), panel_update(3X), panels(3X)

NOTES  The header <panel.h> automatically includes the header <curses.h>. 
NAME  panel_update, update_panels – panels virtual screen refresh routine

SYNOPSIS  cc [ flag ... ] file ... -o panel -lcurses [ library ... ]
#include <panel.h>
void update_panels(void);

MT-LEVEL  Unsafe

DESCRIPTION  update_panels() refreshes the virtual screen to reflect the depth relationships between the panels in the deck. The user must use the curses library call doupdate() (see curs_refresh(3X)) to refresh the physical screen.

SEE ALSO  curs_refresh(3X), curses(3X), panels(3X)

NOTES  The header <panel.h> automatically includes the header <curses.h>.
NAME  panel_userptr, set_panel_userptr – associate application data with a panels panel

SYNOPSIS  cc [ flag ... ] file ... -lpanel -lcurses [ library ... ]
#include <panel.h>
int set_panel_userptr(PANEL *panel, char *ptr);
char *panel_userptr(PANEL *panel);

MT-LEVEL  Unsafe

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.

SEE ALSO  curses(3X), panels(3X)

NOTES  The header <panel.h> automatically includes the header <curses.h>.
NAME
panel_window, replace_panel – get or set the current window of a panels panel

SYNOPSIS
cc [ flag ... ] file ... -loppanel -lcurses [ library ... ]
#include <panel.h>
WINDOW *panel_window(PANEL *panel);
int replace_panel(PANEL *panel, WINDOW *win);

MT-LEVEL
Unsafe

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.

SEE ALSO
curses(3X), panels(3X)

NOTES
The header <panel.h> automatically includes the header <curses.h>.
NAME  panels – character based panels package

SYNOPSIS  

```c
#include <panel.h>
```

MT-LEVEL  Unsafe

DESCRIPTION  The panel library is built using the curses library, and any program using panels routines must call one of the curses initialization routines such as `initscr`. A program using these routines must be compiled with `−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.

SEE ALSO  curses(3X), and 3X pages whose names begin “panel_” for detailed routine descriptions.

3X-808  modified 22 Jan 1993
NOTES | The header `<panel.h>` automatically includes the header `<curses.h>`.
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);

MT-LEVEL MT-Safe

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 directories 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`.

SEE ALSO
`sh(1), test(1), access(2), mknod(2), stat(2), getenv(3C)`

NOTES
The string pointed to by the returned pointer is stored in an area that is reused on subsequent calls to `pathfind()`. The string should not be deallocated by the caller.

When compiling multi-thread applications, the `_REENTRANT` flag must be defined on the compile line. This flag should only be used in multi-thread applications.
NAME  perror, errno – print system error messages

SYNOPSIS  #include <stdio.h>
            void perror(const char *s);
            #include <errno.h>
            int errno;

MT-LEVEL  MT-Safe

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. (How-
ever, 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.

SEE ALSO  intro(2), fmtmsg(3C), gettext(3I), setlocale(3C), strerror(3C)

NOTES  If the application is linked with −lintl, then messages printed from this function are in the
native language specified by the LC_MESSAGES locale category; see setlocale(3C).
NAME
pfmt – display error message in standard format

SYNOPSIS
#include <pfmt.h>
int pfmt(FILE *stream, long flags, char *format, .../* arg */);

MT-LEVEL
MT-safe

DESCRIPTION
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 argu-
ment 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 cata-
log name, if no catalog is specified (either explicitely 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.
- **MM_STD**: Output using the standard message format (default, value 0).

Catalog access control

- **MM_NOGET**: Do not retrieve a localized version of `format`. In this case, only the `<defmsg>` part of the `format` is specified.
- **MM_GET**: Retrieve a localized version of `format`, from the `<catalog>`, using `<msgid>` as the index and `<defmsg>` as the default message (default, value 0).

Severity (standard message format only)

- **MM_HALT**: generates a localized version of HALT, but does not halt the machine.
- **MM_ERROR**: generates a localized version of ERROR (default, value 0).
- **MM_WARNING**: generates a localized version of WARNING.
- **MM_INFO**: generates a localized version of INFO.

Additional severities can be defined. Add-on severities can be defined with number-string pairs with numeric values from the range [5-255], using `addsev()`. The numeric value ORed with other flags will generate the specified severity.

If the severity is not defined, `pfmt()` used the string `SEV=N` where `N` is replaced by the integer severity value passed in `flags`.

Multiple severities passed inf `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:

\[
\begin{align*}
\text{label:} & \quad \text{severity:} & \quad \text{text} \\
\text{label:} & \quad \text{TO FIX:} & \quad \text{text}
\end{align*}
\]

**RETURN VALUE**

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

-1 write error to stream.

**EXAMPLES**

Example 1:

```c
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:

```c
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.

**SEE ALSO**

`addsev(3C)`, `gettext(3C)`, `lfmt(3C)`, `printf(3S)`, `setcat(3C)`, `setLabel(3C)`, `setlocale(3C)`, `environ(5)`

modified 08 December 1993
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.

3C-816 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);

MT-LEVEL

Safe

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. It may contain one of the following: dotted, solid, longdashed, shortdashed, or 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/liblvto.a

**SEE ALSO**

graph(1), ld(1), plot(4B)
**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);
```

**MT-LEVEL**
Unsafe

**DESCRIPTION**
popen() 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), and the child invoked the shell using the call:

Solaris  
```
execl("/usr/bin/sh", "sh", \
"-c", command, (char *)0);
```

XPG4  
```
execl("/usr/bin/ksh", "ksh", \
"-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**
popen() returns a null pointer if files or processes cannot be created.
pclose() returns the termination status of the command. pclose() 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:
```
#include <stdio.h>
#include <stdlib.h>
main()
{
    char *cmd = "/usr/bin/ls *.c";
    char buf[BUFSIZ];
    FILE *ptr;

    if ((ptr = popen(cmd, "r")) != NULL)
        while (fgets(buf, BUFSIZ, ptr) != NULL)
            (void) printf("%s", buf);

    return 0;
}
```
This program will print on the standard output (see stdio(3S)) all the file names in the current directory that have a .c suffix.

SEE ALSO ksh(1), pipe(2), wait(2), waitpid(2), fclose(3), fopen(3), stdio(3), system(3), wstat(5), xpg4(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(3)). 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
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
(\0), in consecutive bytes starting at *s; it is the user’s responsibility to ensure that
enough storage is available.

vprintf(), vfprintf(), and vsprintf() are the same as printf(), fprintf(), and sprintf()
respectively, except that instead of being called with a variable number of arguments,
they are called with an argument list as defined by varargs(5).

Each of these functions converts, formats, and prints its 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 abcedf are used for x conversion and the letters ABCDEF for X conversion. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it will
be expanded with leading zeroes. (For compatibility with older versions, padding with leading zeroes may alternatively be specified by prepending a zero to the field width. This does not imply an octal value for the field width.) The default precision is 1. The result of converting a zero value with a precision of zero is a NULL string.

f

The float or double arg is converted to decimal notation in the style \[−−ddd.ddd\] where the number of digits after the decimal point is equal to the precision specification. If the precision is missing, 6 digits are given; if the precision is explicitly 0, no digits and no decimal point are printed.

e,E

The float or double arg is converted in the style \[−−d.ddde±ddd\], where there is one digit before the decimal point and the number of digits after it is equal to the precision; when the precision is missing, 6 digits are produced; if the precision is zero, no decimal point appears. The E format code will produce a number with E instead of e introducing the exponent. The exponent always contains at least two digits.

g,G

The float or double arg is printed in style f or e (or in style E in the case of a G format code), with the precision specifying the number of significant digits. The style used depends on the value converted: style e or E will be used only if the exponent resulting from the conversion is less than \(-4\) or greater than the precision. Trailing zeroes are removed from the result; a decimal point appears only if it is followed by a digit.

The e, E f, g, and G formats print IEEE indeterminate values (infinity or not-a-number) as "Infinity" or "NaN" respectively.

c

The character arg is printed.

s

The arg is taken to be a string (character pointer) and characters from the string are printed until a NULL character \(\0\) is encountered or until the number of characters indicated by the precision specification is reached. If the precision is missing, it is taken to be infinite, so all characters up to the first NULL character are printed. A NULL value for arg will yield undefined results.

% 

Print a %; no argument is converted.

In no case does a non-existent or small field width cause truncation of a field; if the result of a conversion is wider than the field width, the field is simply expanded to contain the conversion result. Padding takes place only if the specified field width exceeds the actual width. Characters generated by printf() and fprintf() are printed as if putc(3S) had been called.

RETURN VALUES

Upon success, printf() and fprintf() return the number of characters transmitted, excluding the null character. vprintf() and vfprintf() return the number of characters transmitted. sprintf() and vsprintf() always return s. If an output error is encountered, printf(), 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
printf("%s, %s %i, %d:%.2d", weekday, month, day, hour, min);

To print π to 5 decimal places:

printf("pi = %.5f", 4 * atan(1.0));

SEE ALSO econvert(3), putc(3S), scanf(3S), vprintf(3S), varargs(5)

NOTES Use of these interfaces should be restricted to only applications written on BSD platforms. Use of these interfaces with any of the system libraries or in multi-thread applications is unsupported.

Very wide fields (>128 characters) fail.
NAME
printf, fprintf, sprintf – print formatted output

SYNOPSIS
#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 */);

MT-LEVEL
See the NOTES section of this page.

DESCRIPTION
printf() places output on the standard output stream stdout.

fprintf() places output on strm.

sprintf() places output, followed by the null character (\0), in consecutive bytes starting at s. It is the user’s responsibility to ensure that enough storage is available. Each function returns the number of characters transmitted (not including the \0 in the case of sprintf()) or a negative value if an output error was encountered.

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 10 Oct 1990
Each conversion specification is introduced by the character %%. After the character %%, the following appear in sequence:

- An optional field, consisting of a decimal digit string followed by a $, specifying the next args to be converted. If this field is not provided, the args following the last args converted will be used.

- Zero or more flags, which modify the meaning of the conversion specification.

- An optional string of decimal digits to specify a minimum field width. If the converted value has fewer characters than the field width, it will be padded on the left (or right, if the left-adjustment flag (−), described below, has been given) to the field width. If the format is %s or %ws (wide-character string), then the field width should be interpreted as the minimum columns of screen display. E.g. %10s means if the converted value has a screen width of 7 columns, then 3 spaces would be padded on the right.

- An optional precision that gives the minimum number of digits to appear for the d, i, o, u, x, or X conversions (the field is padded with leading zeros), the number of digits to appear after the decimal-point character for the e, E, and f conversions, the maximum number of significant digits for the g and G conversions, or the maximum number of characters to be printed from a string in s or ws conversions. The precision takes the form of a period (.) followed by a decimal digit string; a null digit string is treated as zero. Padding specified by the precision overrides the padding specified by the field width.

- An optional h specifies that a following d, i, o, u, x, or X conversion specifier applies to a short int or unsigned short int argument (the argument will be promoted according to the integral promotions and its value converted to short int or unsigned short int before printing); an optional h specifies that a following n conversion specifier applies to a pointer to a short int argument. An optional l (ell) specifies that a following d, i, o, u, x, or X conversion specifier applies to a long int or unsigned long int argument; an optional l (ell) specifies that a following n conversion specifier applies to a pointer to 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 int or unsigned long long int argument; an optional ll (ell ell) specifies that a following n conversion specifier applies to a pointer to a long long int argument. An optional L specifies that a following e, E, f, g, or G conversion specifier applies to a long double argument. If an h, l, or L appears before any other conversion specifier, the behavior is undefined.

- A conversion character (see below) that indicates the type of conversion to be applied.

A field width or precision may be indicated by an asterisk (∗) instead of a digit string. In this case, an integer args supplies the field width or precision. The args that is actually converted is not fetched until the conversion letter is seen, so the args specifying field width or precision must appear before the args (if any) to be converted. If the precision argument is negative, it will be changed to zero. A negative field width argument is taken as a – flag, followed by a positive field width.
In format strings containing the ∗digits$ form of a conversion specification, a field width or precision may also be indicated by the sequence ∗digits$, giving the position in the argument list of an integer args containing the field width or precision.

When numbered argument specifications are used, specifying the 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 [–]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 $[−]jd.ddde±dd$, 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.

**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 the precision is specified, no more than that many characters are written. If the precision is not specified, it is taken to be infinite, so all characters up to the first null character are printed. A null value for *args* will yield undefined results.

**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, no more than that many wide characters are written. If the precision is not specified, it is taken to be infinite, so all wide characters up to the first null character are printed. A null value for *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 %digits$ sequence is not a valid conversion character, the results of the conversion are undefined.

If a floating-point value is the internal representation for infinity, the output is [±]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.

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

RETURN VALUES  printf(), fprintf(), and sprintf() return the number of characters transmitted, or return a negative value if an error was encountered.

EXAMPLES  
To print a date and time in the form Sunday, July 3, 10:02, where weekday and month are pointers to null-terminated strings:

printf("%s, %s %i, %d:%.2d", weekday, month, day, hour, min);

To print π to 5 decimal places:

printf("π = %.5f", 4 * atan(1.0));

To print a list of names in columns which are 20 characters wide:

printf("%20s%20s%20s", lastname, firstname, middlename );

FILES  
/usr/lib/locale/locale/LC_NUMERIC/numeric  
LC_NUMERIC database for locale

SEE ALSO  
exit(2), lseek(2), write(2), abort(3C), ecvt(3C), putc(3S), scanf(3S), setlocale(3C), stdio(3S)

NOTES  
sprintf() is MT-Safe in multi-thread applications. printf and fprintf can be used safely in a multi-thread application, as long as setlocale(3C) is not being called to change the locale.

modified 10 Oct 1990 3S-829
NAME

proc_service, ps_pstop, ps_pcontinue, ps_lstop, ps_lcontinue, ps_pglobal_lookup,
ps_pdread, ps_pdwrite, ps_ptread, ps_ptwrite, ps_lgetregs, ps_lsetregs, ps_plog,
ps_lgetxregsize, ps_lgetxregs, ps_lsetxregs, ps_lgetfregs, ps_lsetfregs – process service
interface

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_pglobal_lookup(const struct ps_prochandle *ph,
                           const char *ld_object_name, const char *ld_symbol_name,
                           paddr_t *ld_symbol_addr);
ps_err_e ps_pdread(const struct ps_prochandle *ph, paddr_t addr,
                    char *buf, int size);
ps_err_e ps_pdwrite(const struct ps_prochandle *ph, paddr_t addr,
                    char *buf, int size);
ps_err_e ps_ptread(const struct ps_prochandle *ph, paddr_t addr,
                    char *buf, int size);
ps_err_e ps_ptwrite(const struct ps_prochandle *ph, paddr_t addr,
                    char *buf, int size);
ps_err_e ps_lgetregs(const struct ps_prochandle *ph, lwpid_t lwpid,
                     prgregset_t gregset);
ps_err_e ps_lsetregs(const struct ps_prochandle *ph, lwpid_t lwpid,
                     const prgregset_t gregset);
void ps_plog(const char *fmt, ...);
ps_err_e ps_lgetxregsize( const struct ps_prochandle *ph, lwpid_t lwpid,
                          int *xregsize);
ps_err_e ps_lgetxregs( const struct ps_prochandle *ph, lwpid_t lwpid,
                        prxregset_t *xregset);
ps_err_e ps_lsetxregs( const struct ps_prochandle *ph, lwpid_t lwpid,
                       prxregset_t *xregset);
ps_err_e ps_lgetfregs(const struct ps_prochandle *ph, lwpid_t lwpid,
                       prfpregset_t *fregset);
ps_err_e ps_lsetfregs(const struct ps_prochandle *ph, lwpid_t lwpid,
                       const prfpregset_t *fregset);

3T-830 modified 30 Aug 1995
DESCRIPTION
proc_service is a set of interfaces that provide operations on a process. These services are utilized by libraries such as libthread_db to examine a process under inspection (PUI). These functions are provided by the user of such libraries.

FUNCTIONS
ps_pstop() stops the process with the given process handle and returns after the process has stopped.

ps_continue() continues a process with the given process handle that was stopped by ps_pstop().

ps_lstop() stops an LWP on the process with the given lwp identifier and process handle and returns after the LWP has stopped.

ps_lcontinue() continues an LWP on the process with the given lwp identifier and process handle that was stopped by ps_lstop().

ps_pglobal_lookup() finds the absolute address of a symbol given a process handle, symbol name and load object name (e.g., "libthread.so"). Only the global name space is searched.

ps_pdread() reads from a process's data segment at the given address, addr, for size bytes. Contents are returned in buf.

ps_pdwrite() writes contents of buf for size bytes to a process’s data segment at the given address, addr.

ps_ptread() reads from a process’s text segment at the given address, addr, for the size bytes. Contents are returned in buf.

ps_ptwrite() writes contents of buf for size bytes to a process’s text segment at the address, addr.

ps_lgetregs() reads an LWP’s (process with process handle *ph and lwp with identifier lwpid) general register set into gregset.

ps_lsetregs() writes an LWP’s (process with process handle *ph and lwp with identifier lwpid) general register set from gregset.

ps_plog() logs a message. ps_plog() may be used to write diagnostic messages. Input is in printf style format.

ps_lgetxregs() reads an LWP’s (process with process handle *ph and lwp with identifier lwpid) extra register set into xregset.

ps_lsetxregs() writes an LWP’s (process with process handle *ph and lwp with identifier lwpid) extra register set from xregset.

ps_lgetfpregs() reads an LWP’s (process with process handle *ph and lwp with identifier lwpid) floating point register set into fpregset.
**ps_lsetfregs()** writes an LWP’s (process with process handle *ph and lwp with identifier lwpid) floating point register set from fpregset.

**x86 Only** **ps_lgetLDT()** reads an LWP’s (process with process handle *ph and lwp with identifier lwpid) Local Descriptor Table.

### RETURN VALUES

- **PS_OK** The call succeeded.
- **PS_ERR** The call failed without a specific reason.
- **PS_BADPID** This return value indicates that a bad process handle was passed to the function.
- **PS_BADLID** This return value indicates that a bad lwp identifier was passed to the function.
- **PS_BADADDR** This return value indicates that a bad lwp identifier was passed to the function.
- **PS_NOSYM** The symbol could not be found.
- **PS_NOFREGS** This return value indicates that an FPU register set was not available for the lwp.

### SEE ALSO

libthread_db(3T)
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**

```c
#include <siginfo.h>

void psignal(int sig, const char *s);
void psiginfo(siginfo_t *pinfo, char *s);
```

**MT-LEVEL**  Safe

**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.

**SEE ALSO**  `sigaction(2)`, `gettext(3I)`, `perror(3C)`, `setlocale(3C)`, `siginfo(5)`, `signal(5)`

**NOTES**  If the application is linked with `-lintl`, then messages printed from these functions are in the native language specified by the `LC_MESSAGES` locale category; see `setlocale(3C)`.
## NAME

`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.

## SEE ALSO

`fork(2)`, `atexit(3C)`

## 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.

## 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()`.

modified 30 Jun 1995
The following is a brief and simple description of how to make a library safe with respect to \texttt{fork1()} by using \texttt{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 \texttt{pthread_atfork(f1, f2, f3)} in the library’s \texttt{.init} section. f1, f2, f3 are defined as follows:

\begin{verbatim}
f1()
{
    pthread_mutex_lock(L1);
    pthread_mutex_lock(...); --> ordered in lock order
    pthread_mutex_lock(Ln);
}

f2()
{
    pthread_mutex_unlock(L1);
    pthread_mutex_unlock(...);
    pthread_mutex_unlock(Ln);
}

f3()
{
    pthread_mutex_unlock(L1);
    pthread_mutex_unlock(...);
    pthread_mutex_unlock(Ln);
}
\end{verbatim}
NAME

pthread_attr_init, pthread_attr_destroy, 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 contentionscope);
int pthread_attr_getscope(const pthread_attr_t *attr, int *contentionscope);
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);

MT-LEVEL

MT-Safe

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
thread.

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>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>stack allocated by system</td>
</tr>
<tr>
<td>stacksize</td>
<td>NULL</td>
<td>1 megabyte</td>
</tr>
</tbody>
</table>

modified 30 Jun 1995

3T-837
### pthread_attr_init(3T)

**priority**

- priority of parent (calling) thread

**policy**

- SCED_OTHER
  
- determined by system

**inheritsched**

- PTHREAD_EXPLICIT_SCHED
  
- scheduling policy and parameters not inherited but explicitly defined by the attribute object

**NOTE:** 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

**contentionscope**

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)`.)

**schedparam (priority)**

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 `attr` 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.

**schedpolicy**

The `pthread_attr_setschedpolicy()` and `pthread_attr_getschedpolicy()` functions set and get the `schedpolicy` thread attribute in the `attr` argument.

Values for the `policy` attribute are SCHED_FIFO, SCHED_RR, or the default value SCHED_OTHER (see NOTES section below).

**RETURN VALUES**

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()`, and `pthread_attr_getschedpolicy()`.

**ERRORS**

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_setinheritsched()`, `pthread_attr_getinheritsched()`, `pthread_attr_setschedpolicy()`, `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.

SEE ALSO  `pthread_create(3T)`, `pthread_join(3T)`, `thr_create(3T)`. 
Currenty, the only policy supported is `SCHED_OTHER`. Attempting to set policy as `SCHED_FIFO` or `SCHED_RR` will result in the error `ENOSUP`.

The attribute object is part of the POSIX threads interface. There is no Solaris threads counterpart to the POSIX threads attribute object.
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);

MT-LEVEL
MT-Safe

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.

3T-842
modified 30 Jun 1995
**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* parameter; otherwise, an error number is returned.

**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.

pthread_condattr_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.

**SEE ALSO** cond_init(3T), pthread_create(3T), pthread_cond_init(3T), pthread_mutex_init(3T).
NAME
pthread_create, thr_create – thread creation

SYNOPSIS
POSIX
cc [ flag ... ] file ... -lpthread [ library ... ]
#include <pthread.h>
int pthread_create(pthread_t *new_thread_ID, const pthread_attr_t *attr,
    void * (*)(void *)(void *), void *arg);

Solaris
cc [ flag ... ] file ... -lthread [ library ... ]
#include <thread.h>
int thr_create(void *stack_base, size_t stack_size, void *(void *)(void *),
    void *arg, long flags, thread_t *new_thread_ID);

MT-LEVEL
MT-Safe

DESCRIPTION
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 argu-
ment, 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.

3T-844 modified 30 Jun 1995
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>stacksize</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(3T)` 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>stacksize</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:

```
pthread_t tid;
void *start_func(void *), *arg;

pthread_create(&tid, NULL, start_func, arg);
```

This would have the same effect as:

```
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:

```
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(3T)` 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></td>
</tr>
<tr>
<td>concurrency</td>
<td>NULL</td>
<td>Determined by system</td>
<td></td>
</tr>
</tbody>
</table>

3T-846 modified 30 Jun 1995
suspended NULL Runnable, not suspended flags
daemon NULL Not a daemon flags

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 contention scope 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 contention scope to the PTHREAD_SCOPE_SYSTEM in POSIX.

**THR_DETACHED**

This flag affects the detach state 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) (nor pthread_join(3T)) will not wait for a detached thread. This is equivalent to PTHREAD_CREATE_DETACHED in POSIX, which is the default for POSIX.

**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.

\[ \text{thr_create(NULL, NULL, start_func, arg, THR_BOUND | THR_NEW_LWP, \\ &tid);} \]

With \text{thr_create()}, the new thread will use the stack starting at the address specified by \text{stack_base} and continuing for \text{stack_size} bytes. \text{stack_size} must be greater than the value returned by \text{thr_min_stack}(3T). If \text{stack_base} is NULL then \text{thr_create()} allocates a stack for the new thread with at least \text{stack_size} bytes. If \text{stack_size} is zero then a default size is used. If \text{stack_size} is not zero then it must be greater than the value returned by \text{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 \text{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, \text{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, \text{thr_create()} fails and returns the corresponding value:

- **EINVAL**
  
  - \text{stack_base} is not NULL and \text{stack_size} is less than the value returned by \text{thr_min_stack}(3T).
  
  - \text{stack_base} is NULL and \text{stack_size} is not zero and is less than the value returned by \text{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 \text{pthread_create}(3T) if you execute \text{a.out 0}, or \text{thr_create}(3T) if you execute \text{a.out 1}.

Five threads are created that simultaneously perform a time-consuming function, \text{sleep}(10). If the execution of this process is timed, the results will show that all five individual calls to \text{sleep} for ten-seconds completed in about ten seconds, even on a uniprocessor. If a single-threaded process calls \text{sleep}(10) five times, the execution time will be about 50-seconds.

3T-848

modified 30 Jun 1995
The command-line to time this process is:
/usr/bin/time a.out 0 (for POSIX threading)
or
/usr/bin/time a.out 1 (for Solaris threading)

/* cc thisfile.c -lthread -lpthread */
#define _REENTRANT /* basic 3-lines for threads */
#include <pthread.h>
#include <thread.h>

#define NUM_THREADS 5
#define SLEEP_TIME 10

void *sleeping(void *); /* thread routine */
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,0,&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 ...\n", 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);

SEE ALSO _lwp_create(2), exit(2), exit(3C), pthread_attr_init(3T),
pthread_cancel(3T), pthread_exit(3T), pthread_join(3T), thr_suspend(3T),
thr_min_stack(3T), thr_setconcurrency(3T), threads(3T)

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 | cc [ flag ... ] file ... -lpthread [ library ... ]
#include <pthread.h>
int pthread_detach(pthread_t threadID);

MT-LEVEL | MT-Safe

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.

SEE ALSO | pthread_create(3T), pthread_join(3T)
NAME      pthread_equal – compare thread IDs

SYNOPSIS  
#include <pthread.h>
int pthread_equal(pthread_t t1, pthread_t t2);

MT-LEVEL  MT-Safe

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.

SEE ALSO  pthread_create(3T), pthread_self(3T)

NOTES     Solaris thread IDs do not require an equal function because the thread_t structure is really
an unsigned int.

modified 30 Jun 1995
### NAME

pthread_exit, thr_exit – thread termination

### SYNOPSIS

**POSIX**

```c
#include <pthread.h>
void pthread_exit(void *status);
```

**Solaris**

```c
#include <thread.h>
void thr_exit(void *status);
```

### MT-LEVEL

MT-Safe

### 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.

### SEE ALSO

`exit(3C), pthread_cancel(3T), pthread_create(3T), pthread_join(3T), pthread_key_create(3T), pthread_cancel(3T)`.

---

3T-854  modified 30 Jun 1995
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);

MT-LEVEL

MT-Safe

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.

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.)

SEE ALSO `wait(2)`, `pthread_create(3T)`, `pthread_exit(3T)`, `pthread_join(3T)`

NOTES
Using `thr_join(3T)` in the following syntax,

```c
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

cc [ 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

cc [ 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);

MT-LEVEL

MT-Safe

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).
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 cleanup 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) {
        pthread_mutex_lock(&keylock);
        if (!once_per_keyname++)
            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 */
```
void free(void *v) {
    /* application-specific clean-up function */
}

SEE ALSO  pthread_exit(3T)

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**

```c
#include <signal.h>
#include <pthread.h>
int pthread_kill(pthread_t thread, int sig);
```

**Solaris**

```c
#include <signal.h>
#include <thread.h>
int thr_kill(thread_t thread, int sig);
```

**MT-LEVEL**  
Async-Signal-Safe

**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.

**SEE ALSO**

`kill(2), sigaction(2), pthread_self(3T), pthread_sigmask(3T), raise(3C), signal(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.
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);

MT-LEVEL  MT-Safe

DESCRIPTION  In the current implementation, {_POSIX_THREAD_PRIO_PROTECT} is undefined and the functions
              pthread_mutex_setprioceiling() and pthread_mutex_getprioceiling() return
              ENOSYS.

SEE ALSO  pthread_mutex_init(3T)
NAME

pthread_mutexattr_init, pthread_mutexattr_destroy, pthread_mutexattr_getpshared,
 pthread_mutexattr_setpshared, 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);

MT-LEVEL

MT-Safe

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-
stroyed 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_setpshared(), pthread_mutexattr_getprotocol(),
 pthread_mutexattr_setprioceiling(), pthread_mutexattr_getprioceiling() , and
 pthread_mutexattr_setpshared() return 0; otherwise, an error number is returned.

3T-864 modified 30 Jun 1995
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.

**ERRORS**

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.

**SEE ALSO**

`pthread_cond_init(3T)`, `pthread_create(3T)`, `pthread_mutex_init(3T)`

**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_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));
```

**MT-LEVEL**

MT-Safe

**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.

**NOTES**

Solaris threads do not offer this functionality.
NAME  pthread_self, thr_self – get calling thread’s ID

SYNOPSIS

POSIX  cc [ flag ... ] file ... -lpthread [ library ... ]
#include <pthread.h>
pthread_t pthread_self(void);
typedef unsigned int pthread_t;

Solaris  cc [ flag ... ] file ... -lthread [ library ... ]
#include <thread.h>
thread_t thr_self(void)
typedef unsigned int thread_t;

MT-LEVEL  MT-Safe

DESCRIPTION  thr_self() returns the thread ID of the calling thread.
pthread_self() performs the same function as thr_self().

SEE ALSO  pthread_create(3T), pthread_equal(3T)
<table>
<thead>
<tr>
<th>Name</th>
<th>pthread_setschedparam, pthread_getschedparam, thr_setprio, thr_getprio – dynamic access to thread scheduling</th>
</tr>
</thead>
</table>
| Synopsis | **POSIX**

```c
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**

```c
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);
```

<table>
<thead>
<tr>
<th>MT-Level</th>
<th>MT-Safe</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>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.</td>
</tr>
</tbody>
</table>
| Contention scope | Bound threads, which are inter-process, compete system-wide for scheduling resources and must be set at creation, for example:

```c
pthread_attr_setscope(&attr, PTHREAD_SCOPE_SYSTEM);
pthread_create(NULL, &attr, thread_routine, arg);
```

OR

```c
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
POSIX can modify priority at creation time (see `pthread_attr_setschedparam(3T)`). Equivalently, a Solaris thread can be created suspended and its priority can be modified.

`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.

NOTE: 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(2)`. See `pthread_create(3T)` and `priocntl(2)`.

### 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()`).

**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.
- **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.
- **EINVAL** The value of `priority` makes no sense for the scheduling class associated with the `target_thread`.

**SEE ALSO**
`priocntl(2), sched_setparam(3R), sched_setscheduler(3R), pthread_attr_init(3T), pthread_create(3T), thr_suspend(3T), thr_yield(3T)`

**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`.

3T-870 modified 30 Jun 1995
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
cc [ flag ...] file ... -lthread [ library ...]
#include <thread.h>
#include <signal.h>
int thr_sigsetmask(int how, const sigset_t *set, sigset_t *oset);

MT-LEVEL
MT-Safe
Async-Signal-Safe

DESCRIPTION
pthread_sigmask() and thr_sigsetmask() changes and/or examines a calling thread’s signal mask. Each thread has its own signal mask. A new thread inherits the calling thread’s signal mask and priority, however, pending signals are not inherited. Signals pending for a new thread will be empty.

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
Zero is returned upon successful completion; otherwise, a non-zero value indicates an error.

ERRORS
If any of the following conditions occur, pthread_sigmask() or thr_sigsetmask() fails and returns the corresponding value:

EINVAL set is not NULL and the 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:

```c
EFAULT
```

`set` or `oset` are not valid addresses.

**EXAMPLES**

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 than 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_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();
         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 pro-
cess, 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, SIG-
  INT, 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()`.
- 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()`. 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()` should be such that all threads block the signals passed to `sigwait()`, at all times. Only the thread that calls `sigwait()` will get the signals. Note that the call to `sigwait()` takes two arguments. See `sigwait()`.

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()`.

SEE ALSO `sigaction()`, `sigprocmask()`, `sigwait()`, `sigsetops()`, `pthread_cancel()`, `pthread_create()`, `pthread_exit()`, `pthread_join()`, `pthread_kill()`, `pthread_self()`.

NOTES

It is not possible to block signals that cannot be ignored (see `sigaction()`). If using the threads library, it is not possible to block the signals `SIGLWP` or `SIGCANCEL`, which are reserved by the threads library. Additionally, it is impossible to unblock the signal `SIGWAITING`, which is always blocked on all threads. This restriction is quietly enforced by the threads library.

Using `sigwait()` in a dedicated thread allows asynchronous signals to be managed synchronously; however, `sigwait()` should never be used to manage synchronous signals. Synchronous signals (i.e., exceptions or traps) are sent by the process itself, such as `SIGFPE`, `pthread_kill()`, `pthread_exit()`, `pthread_cancel()`, `thr_kill()`.
thr_exit(3T), rather than device interrupts or signals sent by other processes.
Synchronous 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 will not generate any synchronous signals.
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 pro-
grams 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 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 environ-
ment, this safeness is not guaranteed to be portable to other POSIX domains.
## NAME
ptsname – get name of the slave pseudo-terminal device

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

char *ptsname(int fildes);
```

## MT-LEVEL
Safe

## DESCRIPTION
The `ptsname()` function returns the name of the slave pseudo-terminal device associated with a master pseudo-terminal device. `fildes` 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 `fildes` is an invalid file descriptor or if the slave device name does not exist in the file system.

## SEE ALSO
- `open(2)`, `grantpt(3C)`, `ttyname(3C)`, `unlockpt(3C)`
- 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);

MT-LEVEL  See the NOTES section of this page.

DESCRIPTION  putc() 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.

putc_unlocked() and putchar_unlocked() are respectively variants of putc() and putchar() that do not lock the stream. It is the caller’s responsibility to acquire the stream lock before calling these functions and releasing the lock afterwards; see flockfile(3S) and stdio(3S).

fputc() behaves like putc(), but is a function rather than a macro. fputc() runs more slowly than putc(), but it takes less space per invocation and its name can be passed as an argument to a function.

putw() 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. putw() 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.

SEE ALSO  write(2), intro(3), fclose(3S), ferror(3S), flockfile(3S), fopen(3S), printf(3S), puts(3S), setbuf(3S), stdio(3S)

NOTES  Because it is implemented as a macro, putc() evaluates a stream argument more than once. In particular, putc(c, *f++); does not work sensibly. fputc() should be used instead.

modified 22 Jan 1993

3S-877
Because of possible differences in word length and byte ordering, files written using `putw()` are machine-dependent, and may not be read using `getw()` on a different processor.

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

`fputc()`, `putc()`, `putchar()`, and `putw()` are MT-Safe in multi-thread applications.

`putc_unlocked()` and `putchar_unlocked()` are unsafe in multi-thread applications.
NAME
putenv – change or add value to environment

SYNOPSIS
#include <stdlib.h>
int putenv(const char *string);

MT-LEVEL
Safe

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.

SEE ALSO
exec(2), getenv(3C), malloc(3C), environ(5)

NOTES
This routine uses malloc(3C) to enlarge the environment.
After putenv() is called, environment variables are not in alphabetical order.
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  putpwent – write password file entry

SYNOPSIS  #include <pwd.h>
           int putpwent(const struct passwd *p, FILE *f);

MT-LEVEL  Unsafe

DESCRIPTION  putpwent() is the inverse of getpwent(), (see getpwnam(3C)). Given a pointer to a
              passwd structure created by getpwent() (or getpwuid() or getpwnam()), putpwent()
              writes a line on the stream f, which matches the format of /etc/passwd.

RETURN VALUES  putpwent() returns non-zero if an error was detected during its operation, otherwise
                zero.

SEE ALSO  getpwnam(3C), putspent(3C)

NOTES  Do not use without also using putspent() to update the shadow file.
       The use of this function is discouraged.

BUGS  This routine is of limited utility, since most password files are maintained as Network
       Information Service (NIS) files, and cannot be updated with this routine.
NAME  puts, fputs – put a string on a stream

SYNOPSIS  #include <stdio.h>
           int puts(const char *s);
           int fputs(const char *s, FILE *stream);

MT-LEVEL  MT-Safe

DESCRIPTION  puts() writes the string pointed to by s, followed by a new-line character, to the standard output stream stdout (see intro(3)).
             fputs() writes the null-terminated string pointed to by s to the named output stream. Neither function writes the terminating null character.

RETURN VALUES  On success both routines return the number of characters written; otherwise they return EOF.

SEE ALSO  write(2), intro(3)), fclose(3S), ferror(3S), fopen(3S), printf(3S), putc(3S), stdio(3S)

NOTES  puts() appends a new-line character while fputs() does not.
NAME
putspent – write shadow password file entry

SYNOPSIS
#include <shadow.h>
int putspent(const struct spwd *p, FILE *fp);

MT-LEVEL
Unsafe

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 structure 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.

SEE ALSO
getpwnam(3C), getspnam(3C), putpwent(3C)

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, putwchar, fputwc – convert Process Code character to EUC and put on a stream

SYNOPSIS
cc [flag ...] file ... -lw [library ...]
#include <stdio.h>
#include <widec.h>
wchar_t putwc(int c, FILE *stream);
wchar_t putwchar(int c);
wchar_t fputwc(int c, FILE *stream);

MT-LEVEL
MT-Safe

DESCRIPTION
putwc() and fputwc() convert the Process Code (wchar_t) character c to Extended Unix Code (EUC) and write it onto the named output stream (at the position where the file pointer, if defined, is pointing). It returns the character written.

putwchar(c) is defined as fputwc(c, stdout). putwc() and putwchar() are macros.
See stdio(3S) for a discussion of output streams.

RETURN VALUES
On success, these functions each return the value passed. On error, these functions return the constant EOF.

SEE ALSO
fclose(3S), ferror(3S), fopen(3S), fread(3S), getwc(3I), printf(3S), putc(3S), putws(3I), setbuf(3S), stdio(3S)
NAME
putws, fputws – convert a string of Process Code characters to EUC characters and put it on a stream

SYNOPSIS
cc [flag...] file... -lw [library...]
#include <stdio.h>
#include <widec.h>
int putws(wchar_t *s);
int fputws(wchar_t *s, FILE *stream);

MT-LEVEL
MT-Safe

DESCRIPTION
putws() 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.
fputws() writes the (wchar_t)NULL-terminated string pointed to by s to the named output stream, and does not append a NEWLINE.
Neither function writes the terminal NULL character.

RETURN VALUES
Both routines return the number of Process Code characters transformed and written. Both routines return EOF on error. This will happen if the routines try to write on a file that has not been opened for writing.

SEE ALSO ferror(3S), fopen(3S), fread(3S), getws(3I), printf(3S), putwc(3I)
NAME  qsort – quick sort

SYNOPSIS  

#include <stdlib.h>

void qsort(void *base, size_t nel, size_t width,
            int (*compar) (const void *, const void *));

MT-LEVEL  Safe

DESCRIPTION  

qsort() is an implementation of the quick-sort algorithm. It sorts a table of data in place. The contents of the table are sorted in ascending order according to the user-supplied comparison function.

base points to the element at the base of the table. nel is the number of elements in the table. width specifies the size of each element in bytes. compar 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");
}

SEE ALSO  sort(1), bsearch(3C), lsearch(3C), string(3C)

NOTES The comparison function need not compare every byte, so arbitrary data may be con-
tained in the elements in addition to the values being compared.
The relative order in the output of two items that compare as equal is unpredictable.
NAME  
raise – send signal to program

SYNOPSIS  
#include <signal.h>
int raise(int sig);

MT-LEVEL  
MT-Safe

DESCRIPTION  
raise() sends the signal 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.

SEE ALSO  
getpid(2), kill(2), signal(3C)
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  #include <stdlib.h>
            int rand(void);
            void srand(unsigned int seed);
            int rand_r(unsigned int *seed);

MT-LEVEL  See the NOTES section of this page.

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 RAND_MAX (defined in <stdlib.h>).

The function srand() uses the argument 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 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 seed value of 1.

rand_r() has the same functionality as rand() except that a pointer to a seed seed must be supplied by the caller. The seed to be supplied is not the same seed as in srand().

SEE ALSO  drand48(3C)

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.
NAME  random, srandom, initstate, setstate – better random number generator; routines for changing generators

SYNOPSIS  long random( );
            int srandom( unsigned seed );
            char *initstate( unsigned seed, char *state, int n );
            char *setstate( char *state );

MT-LEVEL  See the NOTES section of this page.

DESCRIPTION  random( ) uses a non-linear additive feedback random number generator employing a default table of size 31 long integers to return successive pseudo-random numbers in the range from 0 to \( 2^{31} - 1 \). The period of this random number generator is very large, approximately \( 16 \times (2^{31} - 1) \).

random( ) and srandom( ) have (almost) the same calling sequence and initialization properties as rand( ) and srand( ) (see rand(3C)). The difference is that rand(3C) produces a much less random sequence—in fact, the low dozen bits generated by rand go through a cyclic pattern. All the bits generated by random( ) are usable. For example,

\[
\text{random()} \& 01
\]

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. Like rand(3C), however, random( ) will, by default, produce a sequence of numbers that can be duplicated by calling srand( ) with 1 as the seed.

The initstate( ) routine allows a state array, passed in as an argument, to be initialized for future use. \( n \) specifies the size of state in bytes. initstate( ) uses \( n \) to decide how sophisticated a random number generator it should use—the more state, the better the random numbers will be. Current “optimal” values for the amount of state information are 32, 64, 128, and 256 bytes. If the amount of state information is less than 32 bytes, a simple linear congruential random number generator is used. Using less than 8 bytes causes an error.

The seed for the initialization (which specifies a starting point for the random number sequence, and provides for restarting at the same point) is also an argument. initstate( ) returns a pointer to the previous state information array.

Once a state has been initialized, the setstate( ) routine provides for rapid switching between states. setstate( ) returns a pointer to the previous state array; its argument state array is used for further random number generation until the next call to initstate( ) or setstate( ).

Once a state array has been initialized, it may be restarted at a different point either by calling initstate( ) (with the desired seed, the state array, and its size) or by calling both setstate( ) (with the state array) and srandom( ) (with the desired seed). The advantage of calling both setstate( ) and srandom( ) is that the size of the state array does not have to

3C-890  modified 30 Sep 1994
be remembered after it is initialized.
With 256 bytes of state information, the period of the random number generator is
greater than $2^{69}$, which should be sufficient for most purposes.

**RETURN VALUES**
If `initstate()` is called with less than 8 bytes of state information, or if `setstate()` detects
that the state information has been garbled, error messages are printed on the standard
error output.

**EXAMPLES**

```c
/* Initialize an array and pass it in to initstate. */
static long state1[32] = {
    3,
    0x9a319039, 0x32d9c024, 0x9b663182, 0x5da1f342,
    0x7449e56b, 0xbeb1dbb0, 0xab5c5918, 0x946554fd,
    0x8c2e680f, 0xeb3d799f, 0xb11ee0b7, 0x2d436b86,
    0xda672e2a, 0x1588ca88, 0xe369735d, 0x904f35f7,
    0xd7158fd6, 0x6fa6f051, 0x616e6b96, 0xac94efdc,
    0xde3b81e0, 0xdf0a6fb5, 0xf103bc02, 0x48f340fb,
    0x36413f93, 0xc622c298, 0xf5a42ab8, 0x8a88d77b,
    0xf5ad9d0e, 0x8999220b, 0x27fb47b9
};
main() {
    unsigned seed;
    int n;
    seed = 1;
    n = 128;
    initstate(seed, state1, n);
    setstate(state1);
    printf("%d,random()");
}
```

**SEE ALSO**
`drand48(3C), rand(3C)`

**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)`.

modified 30 Sep 1994
NAME
rcmd, rresvport, ruserok – routines for returning a stream to a remote command

SYNOPSIS
cc [ flag ... ] file ... -lsocket -lnsl [ 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);

MT-LEVEL
Unsafe

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 for-
warded 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 possi-
bly .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

**SEE ALSO**

rlogin(1), rsh(1), in.rexecd(1M), in.rshd(1M), intro(2), gethostbyname(3N), rexec(3N)

**NOTES**

The error code **EAGAIN** is overloaded to mean “All network ports in use.”

These interfaces are unsafe in multithreaded applications. Unsafe interfaces should be called only from the main thread.
read_vtoc (3X)  Miscellaneous Library Functions  SunOS 5.5

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);

MT-LEVEL  Unsafe

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.

SEE ALSO  format(1M), fmthard(1M), prtvtoc(1M), ioctl(2), dkio(7I)

BUGS  write_vtoc cannot write a VTOC on an unlabeled disk. Use format(1M) for this purpose.

3X-894  modified 22 Jan 1993
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 readdir() 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 filesystems. It returns a NULL pointer upon reaching the end of the directory stream, or upon detecting an invalid location in the directory.

readdir() 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. readdir() 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.

RETURN VALUES readdir() returns NULL on failure and sets errno to indicate the error.

ERRORS readdir() 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.

EINTR  A signal was caught during the read() or readv() function.

EINVAL  Attempted to read from a stream linked to a multiplexor.

EIO  A physical I/O error has occurred, or the process is in a background process group and is attempting to read from its controlling terminal, and either the process is ignoring or blocking the SIGTTIN signal or the

modified 03 Aug 1994
<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENOENT</td>
<td>The current file pointer for the directory is not located at a valid entry.</td>
</tr>
<tr>
<td>ENOLCK</td>
<td>The system record lock table was full, so the <code>read()</code> or <code>readv()</code> could not go to sleep until the blocking record lock was removed.</td>
</tr>
<tr>
<td>ENOLINK</td>
<td><code>fildes</code> is on a remote machine and the link to that machine is no longer active.</td>
</tr>
<tr>
<td>ENXIO</td>
<td>The device associated with <code>fildes</code> is a block special or character special file and the value of the file pointer is out of range.</td>
</tr>
</tbody>
</table>

**SEE ALSO**
- `getdents(2)`, `scandir(3B)`, `directory(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  realpath – returns the real file name

SYNOPSIS  

```
#include <stdlib.h>
#include <sys/param.h>
char *realpath(char *file_name, char *resolved_name);
```

MT-LEVEL  MT-Safe

DESCRIPTION  realpath() resolves all links and references to “." and “..” in file_name and stores it in resolved_name. It can handle both relative and absolute path names. For absolute path names and the relative names whose resolved name cannot be expressed relatively (for example, ../../rel-dir), it returns the resolved absolute name. For the other relative path names, it returns the resolved relative name.

resolved_name must be big enough (MAXPATHLEN) to contain the fully resolved path name.

RETURN VALUES  If there is no error, realpath() returns a pointer to the resolved_name. Otherwise it returns a null pointer and places the name of the offending file in resolved_name. The global variable errno is set to indicate the error.

SEE ALSO  getcwd(3C), sysconf(3C)

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.

modified 22 Jan 1993  
3C-897
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
recv, recvfrom, recvmsg – receive a message from a socket

SYNOPSIS
cc [ flag ...] file ... -lsocket -lnsl [ 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);

MT-LEVEL
Safe

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_iov_len; /* # elements in msg_iov */
caddr_t msg_accrights; /* access rights sent/received */
int msg_accrights_len;

modified 30 Mar 1993

3N-899
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_iov_len** describe the scatter-gather locations, as described in *read*(2). A buffer to receive any access rights sent along with the message is specified in **msg_accrights**, which has length **msg_accrights_len**.

**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.

**SEE ALSO**  
NAME
regcmp, regex – compile and execute regular expression

SYNOPSIS
cc [flag ...] file ... -lgen [library ...]
#include <libgen.h>

char *regcmp(const char *string1, char *string2 ... , int *(char *)0);
char *regex(const char *re, const char *subject, char *ret0 ...);
extern char *__loc1;

MT-LEVEL
MT-Safe

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.

[] *.: This group of symbols retains its meaning as described on the regexp(5) manual page.

$ Matches 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,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} 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`

**SEE ALSO**

`ed(1), regcmp(1), malloc(3C), 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.
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 cflags);
int regexec(const regex_t *preg, const char *string, size_t nmatch, regmatch_t pmatch[],
int eflags);
size_t regerror(int errcode, const regex_t *preg, char *errbuf, size_t errbuf_size);
void regfree(regex_t *preg);

MT-LEVEL
MT-Safe

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 cflags 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 appli-
cation can specify Extended Regular Expressions using the REG_EXTENDED cflags flag.
If the REG_NOSUB flag was not set in cflags, 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 eflags
argument is the bitwise inclusive OR of zero or more of the following flags, which are
defined in the header <regex.h>:
REG_NOTBOL

The first character of the string pointed to by string is not the beginning of the line. Therefore, the circumflex character (^), when taken as a special character, will not match the beginning of string.

REG_NOTEOL

The last character of the string pointed to by string is not the end of the line. Therefore, the dollar sign ($), when taken as a special character, will not match the end of string.

If nmatch is zero or REG_NOSUB was set in the cflags argument to regcomp(), then 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 \ \ appar...
5. If subexpression $i$ matched a zero-length string, then both byte offsets in $pmatch[i]$ will be the byte offset of the character or NULL terminator immediately following the zero-length string.

If, when `regexec()` is called, the locale is different from when the regular expression was compiled, the result is undefined.

If `REG_NEWLINE` is not set in `flags`, then a NEWLINE character in `pattern` or `string` will be treated as an ordinary character. If `REG_NEWLINE` is set, then newline will be treated as an ordinary character except as follows:

1. A NEWLINE character in `string` will not be matched by a period outside a bracket expression or by any form of a non-matching list.
2. A circumflex (^) in `pattern`, when used to specify expression anchoring will match the zero-length string immediately after a newline in `string`, regardless of the setting of `REG_NOTBOL`.
3. A dollar-sign ($) in `pattern`, when used to specify expression anchoring, will match the zero-length string immediately before a newline in `string`, regardless of the setting of `REG_NOTEOL`.

`regfree()` The `regfree()` function frees any memory allocated by `regcomp()` associated with `preg`.

The following constants are defined as error return values:

- `REG_NOMATCH` `regexec()` failed to match.
- `REG_BADPAT` Invalid regular expression.
- `REG_ECOLLATE` Invalid collating element referenced.
- `REG_ECTYPE` Invalid character class type referenced.
- `REG_EESCAPE` Trailing \ in pattern.
- `REG_ESUBREG` Number in \digit invalid or in error.
- `REG_EBRACK` [ ] imbalance.
- `REG_ENOSYS` The function is not supported.
- `REG_EPAREN` \( \) or ( ) imbalance.
- `REG_EBRACE` \{ \} imbalance.
- `REG_BADBR` Content of \{ \} invalid: not a number, number too large, more than two numbers, first larger than second.
- `REG_ERANGE` Invalid endpoint in range expression.
- `REG_ESPACE` Out of memory.
- `REG_BADRPT` ?, * or + not preceded by valid regular expression.

`regerror()` The `regerror()` function provides a mapping from error codes returned by `regcomp()` and `regexec()` to unspecified printable strings. It generates a string corresponding to the value of the `errcode` argument, which must be the last non-zero value returned by `regcomp()` or `regexec()` with the given value of `preg`. If `errcode` is not such a value, an error message indicating that the error code is invalid is returned.
If *preg* is a NULL pointer, but *errcode* is a value returned by a previous call to *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()*.

**RETURN VALUES**

The following values are returned by *regcomp()*:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>successful completion</td>
</tr>
<tr>
<td>non-zero</td>
<td>an error has occurred. The value returned is described in <code>&lt;regex.h&gt;</code>, and the content of <em>preg</em> is undefined.</td>
</tr>
</tbody>
</table>

The following values are returned by *regexec()*:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>successful completion.</td>
</tr>
<tr>
<td>REG_NOMATCH</td>
<td>no match</td>
</tr>
<tr>
<td>REG_ENOSYS</td>
<td>the function is not supported.</td>
</tr>
</tbody>
</table>

The following values are returned by *regerror()*:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>the function is not implemented.</td>
</tr>
</tbody>
</table>

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:

```c
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
table, 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);
}

SEE ALSO fnmatch(3C), glob(3C), malloc(3C), regex(5)

modified 26 Jan 1995
NAME regex, re_comp, re_exec – regular expression handler

SYNOPSIS #include <re_comp.h>
char *re_comp(const char *sp);
int re_exec(const char *p1);

DESCRIPTION re_comp() compiles a string into an internal form suitable for pattern matching.
re_exec() checks the argument string against the last string passed to re_comp().
re_comp() returns a NULL pointer if the string sp was compiled successfully; otherwise a string containing an error message is returned. If re_comp() is passed 0 or a NULL string, it returns without changing the currently compiled regular expression.
re_exec() returns 1 if the string p1 matches the last compiled regular expression, 0 if the string p1 failed to match the last compiled regular expression, and −1 if the compiled regular expression was invalid (indicating an internal error).
The strings passed to both re_comp() and re_exec() may have trailing or embedded NEWLINE characters; they are terminated by NULL characters. The regular expressions are described on the regexp(5) manual page.

RETURN VALUES re_exec() returns −1 for an internal error.
re_comp() returns one of the following strings if an error occurs:
   - No previous regular expression
   - Regular expression too long
   - unmatched \(
     - missing ]
   - too many \(\) pairs
   - unmatched \)

SEE ALSO grep(1), regcmp(1), regcmp(3G), regexpr(3G), regexp(5)

NOTES The regular expressions of the form \{m\}, \{m,\}, or \{m,n\} are not supported.
NAME
regexpr, compile, step, advance – regular expression compile and match routines

SYNOPSIS
cc [ flag ...] file ... -lgen [ library ...]
#include <regexpr.h>
char *compile(const char *instring, char *expbuf, char *endbuf);
int step(const char *string, char *expbuf);
int advance(const char *string, char *expbuf);
extern char **loc1, **loc2, **locs;
extern int nbra, regerrno, reglength;
extern char **braslist[], **braelist[];

MT-LEVEL
MT-Safe

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 <regexpr.h>

... 
if(compile(*argv, (char **)0, (char **)0) == (char **)0)
  regerr(regerrno);
... 
if (step(linebuf, expbuf))
  succeed();
```

**RETURN VALUES**

If `compile()` succeeds, it returns a non-`NULL` pointer whose value depends on `expbuf`. If `expbuf` is non-`NULL`, `compile()` returns a pointer to the byte after the last byte in the compiled regular expression. The length of the compiled regular expression is stored in `reglength`. Otherwise, `compile()` returns a pointer to the space allocated by `malloc`.

The functions `step()` and `advance()` return non-zero if the given string matches the regular expression, and zero if the expressions do not match.

**ERRORS**

If an error is detected when compiling the regular expression, a `NULL` pointer is returned from `compile()` and `regerrno` is set to one of the non-zero error numbers indicated below:

<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><code>&quot;\digit&quot;</code> 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>`(^) imbalance.</td>
</tr>
<tr>
<td>43</td>
<td>Too many <code>\(</code></td>
</tr>
</tbody>
</table>

3G-910 modified 21 Jan 1994
ENVIRONMENT

If any of the LC_* variables (LC_CTYPE, LC_MESSAGES, LC_TIME, LC_COLLATE, LC_NUMERIC, and LC_MONETARY) (see environ(5)) are not set in the environment, the operational behavior of tar for each corresponding locale category is determined by the value of the LANG environment variable. If LC_ALL is set, its contents are used to override both the LANG and the other LC_* variables. If none of the above variables is set in the environment, the “C” (U.S. style) locale determines how tar behaves.

LC_CTYPE

Determines how tar handles characters. When LC_CTYPE is set to a valid value, tar can display and handle text and filenames containing valid characters for that locale. tar can display and handle Extended Unix code (EUC) characters where any individual character can be 1, 2, or 3 bytes wide. tar can also handle EUC characters of 1, 2, or more column widths. In the “C” locale, only characters from ISO 8859-1 are valid.

SEE ALSO

ed(1), grep(1), sed(1), malloc(3C), 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.
<table>
<thead>
<tr>
<th><strong>NAME</strong></th>
<th>remove – remove file</th>
</tr>
</thead>
</table>
| **SYNOPSIS** | #include <stdio.h>  
int remove(const char *path); |
| **MT-LEVEL** | MT-Safe |
| **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. |
| **SEE ALSO** | rmdir(2), unlink(2) |
NAME
resolver, res_init, res_mkquery, res_send, res_search, 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, char *dname, int class, int type, char *data, int datalen,
    struct rrec *newrr, char *buf, int buflen);
int res_send(char *msg, int msglen, char *answer, int anslen);
int res_search(char *dname, int class, int type, uchar *answer, int anslen);
int dn_comp(char *exp_dn, char *comp_dn, int length, char **dnptrs, char **lastdnptr);
int dn_expand(char *exp_dn, char *comp_dn, int length, char *msg, char *buf, int buflen);

MT-LEVEL
Unsafe

DESCRIPTION
These routines are used for making, sending and interpreting packets to Internet domain name servers. Global information that is used by the resolver routines is kept in the variable _res. Most of the values have reasonable defaults and can be ignored. Options are a simple bit mask and are OR-ed in to enable. Options stored in _res.options are defined in <resolv.h> and are as follows.

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_AAONLY Accept authoritative answers only. res_send() will continue until it finds an authoritative answer or finds an error. Currently this is not implemented.
RES_USEVC Use TCP connections for queries instead of UDP.
RES_PRIMARY Query primary server only.
RES_IGNTC Unused currently (ignore truncation errors, that is, do not retry with TCP).
RES_RECURSE Set the recursion desired bit in queries. This is the default. res_send() does not do iterative queries and expects the name server to handle recursion.
RES_DEFNAMES Append the default domain name to single label queries. This is the default.

modified 7 Jan 1994 3N-913
<table>
<thead>
<tr>
<th><strong>RES_STAYOPEN</strong></th>
<th>Used with <strong>RES_USEVC</strong> to keep the TCP connection open between queries. This is useful only in programs that regularly do many queries. <strong>UDP</strong> should be the normal mode used.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RES_DNSRCH</strong></td>
<td>Search up local domain tree.</td>
</tr>
<tr>
<td><strong>res_init()</strong></td>
<td>reads the initialization file to get the default domain name and the Internet address of the initial hosts running the name server. If this line does not exist, the host running the resolver is tried.</td>
</tr>
<tr>
<td><strong>res_mkquery()</strong></td>
<td>makes a standard query message and places it in <code>buf</code>. <code>res_mkquery()</code> will return the size of the query or −1 if the query is larger than <code>buflen</code>. <code>op</code> is usually <strong>QUERY</strong> but can be any of the query types defined in <code>&lt;arpa/nameser.h&gt;</code>. <code>dname</code> is the domain name. If <code>dname</code> consists of a single label and the <strong>RES_DEFNAMES</strong> flag is enabled (the default), <code>dname</code> will be appended with the current domain name. The current domain name is defined in a system file and can be overridden by the environment variable <strong>LOCALDOMAIN</strong>. <code>newrr</code> is currently unused but is intended for making update messages. <code>class</code> and <code>type</code> define the class and type of query. <code>*data</code> is the resource record; and <code>datalen</code> is the length of the record.</td>
</tr>
<tr>
<td><strong>res_send()</strong></td>
<td>sends a query to name servers and returns an answer. It will call <strong>res_init()</strong> if <strong>RES_INIT</strong> is not set, send the query to the local name server, and handle timeouts and retries. <code>msg</code> is the query sent; <code>msglen</code> is its length. <code>answer</code> is the response returned. The length of the response is stored in <code>anslen</code>. <strong>res_send()</strong> returns the length of the response or −1 if there were errors.</td>
</tr>
<tr>
<td><strong>res_search()</strong></td>
<td>formulates and sends a normal query (<strong>QUERY</strong>) message, and stores the response in a buffer supplied by the caller. <code>dname</code> is the domain name. <code>class</code> and <code>type</code> define the class and type of query (see <code>&lt;arpa/nameser.h&gt;</code>). The response is returned in the user-supplied buffer <code>answer</code>. <strong>res_search</strong> returns the length of <code>answer</code> in <code>anslen</code>. <strong>res_search()</strong> will call <strong>res_init()</strong> if the <strong>RES_INIT</strong> flag is not enabled. If <code>dname</code> consists of a single label and the <strong>RES_DEFNAMES</strong> flag is enabled (the default), <code>dname</code> will be appended with the current domain name. If the <strong>RES_DNSRCH</strong> flag is enabled, <strong>res_search()</strong> will search up the local domain tree until an answer has been retrieved or an unrecoverable error has been encountered. <strong>res_search()</strong> returns the length of <code>answer</code> on success and −1 on error. Note that <strong>res_search()</strong> is only useful for queries in the same name hierarchy as the local host.</td>
</tr>
<tr>
<td><strong>dn_expand()</strong></td>
<td>expands the compressed domain name <code>comp_dn</code> to a full domain name. Expanded names are converted to upper case. <code>msg</code> is a pointer to the beginning of the message, <code>exp_dn</code> is a pointer to a buffer of size <code>length</code> for the result. The size of compressed name is returned or −1 if there was an error.</td>
</tr>
</tbody>
</table>
| **dn_comp()**    | compresses the domain name `exp_dn` and stores it in `comp_dn`. The size of the compressed name is returned or −1 if there were errors. `length` is the size of the array pointed to by `comp_dn`. `dnptrs` is a list of pointers to previously compressed names in the current message. The first pointer points to the beginning of the message and the list ends with **NULL**. `lastdnptr` is a pointer to the end of the array pointed to `dnptrs`. A side effect is to update the list of pointers for labels inserted into the message by **dn_comp()** as the name is compressed. If `dnptrs` is **NULL**, do not try to compress names. If `lastdnptr`
is NULL, do not update the list.

**FILES**
/etc/resolv.conf

**SEE ALSO**
in.named(1M), nstest(1M), resolv.conf(4)

**NOTES**
These interfaces are unsafe in multithreaded applications. Unsafe interfaces should be called only from the main thread.
NAME
rexec – return stream to a remote command

SYNOPSIS
cc [flag ...] file ... -lsocket -linsl [ library ... ]
int rexec(char **ahost, unsigned short inport, const char *user, const char *passwd,
        const char *cmd, int *fd2p);

MT-LEVEL
Unsafe

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.

SEE ALSO
in.rexecd(1M), gethostbyname(3N), getservbyname(3N)

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-916  modified 11 Feb 1994
NAME  
rpc – library routines for remote procedure calls

SYNOPSIS  
cc [ flag ...] file ... -lnsl [ library ...]
#include <rpc/rpc.h>
#include <netconfig.h>

MT-LEVEL  
MT-Safe with exceptions

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 argu-
ments 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 param-
ters (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

modified 4 Feb 1994
datagram transports (semantics **tpi_clts**) from the entries in the
/etc/netconfig file.

**circuit_n**
This is same as **netpath** except that it chooses only the connection
oriented datagram transports (semantics **tpi_cots** or **tpi_cots_ord**).

**datagram_n**
This is same as **netpath** except that it chooses only the connectionless
datagram transports (semantics **tpi_clts**).

**udp**
This refers to Internet UDP.

**tcp**
This refers to Internet TCP.

If **nettype** is **NULL**, it defaults to **netpath**. The transports are tried in left to right order in
the **NETPATH** variable or in top to down order in the **/etc/netconfig** file.

---

### Data Structures

Some of the data structures used by the RPC package are shown below.

#### The AUTH Structure

```c
union des_block {
    struct {
        u_int32 high;
        u_int32 low;
    } key;
    char c[8];
};
typedef union des_block des_block;
extern bool_t xdr_des_block();

/*
 * Authentication info. Opaque to client.
 */
struct opaque_auth {
    enum_t oa_flavor; /* flavor of auth */
    caddr_t oa_base; /* address of more auth stuff */
    u_int oa_length; /* not to exceed MAX_AUTH_BYTES */
};

/*
 * Auth handle, interface to client authenticators.
 */
typedef struct {
    struct opaque_auth ah_cred;
    struct opaque_auth ah_verf;
    union des_block ah_key;
    struct auth_ops {
        void (*ah_nextverf)();
        int (*ah_marshall)(); /* nextverf & serialize */
        int (*ah_validate)(); /* validate verifier */
        int (*ah_refresh)(); /* refresh credentials */
    } ah_ops;
} auth_cookie;
```
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
*/
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 xpert_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 xpert_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 */
}
```c
void (*xp_destroy)( ); /* destroy this struct */
} *xp_ops;
int xp_addrlen; /* length of remote addr. 
     Obsolete */
char *xp_tp; /* transport provider device 
     name */
char *xp_netid; /* network identifier */
struct netbuf xp_ltaddr; /* local transport address */
struct netbuf xp_rtaddr; /* remote transport address */
char xp_raddr[16]; /* remote address. Obsolete */
struct opaque_auth xp_verf; /* raw response verifier */
caddr_t xp_p1; /* private: for use 
     by svc ops */
caddr_t xp_p2; /* private: for use 
     by svc ops */
caddr_t xp_p3; /* private: for use 
     by svc lib */
int xp_type /* transport type */
}; SVCXPRT;

The svc_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 4 Feb 1994
## Index to Routines

The following table lists RPC routines and the manual reference pages on which they are described:

<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_sperro</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>Module</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_svc_create(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_svc_reg(3N)</td>
</tr>
<tr>
<td>svc_getreq</td>
<td>rpc_svc_reg(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_calls(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_unregister</td>
<td>rpc_svc_calls(3N)</td>
</tr>
<tr>
<td>svc_unregister</td>
<td>rpc_svc_calls(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>
svecerr_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/etconfig

SEE ALSO
getnetconfg(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),
etconfig(4), rpc(4), environ(5)
NAME
rpc_clnt_auth, auth_destroy, authnone_create, authsys_create, authsys_create_default—
library routines for client side remote procedure call authentication

MT-LEVEL
MT-Safe

DESCRIPTION
These routines are part of the RPC library that allows C language programs to make pro-
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 authen-
tication information with each remote procedure call. This is the default authenti-
cation 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 authen-
tication 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.
| SEE ALSO | `kerberos_rpc(3N), rpc(3N), rpc_clnt_calls(3N), rpc_clnt_create(3N), secure_rpc(3N)` |
RPC library routines allow C language programs to make procedure calls on other machines across the network. First, the client calls a procedure to send a request to the server. Upon receipt of the request, the server calls a dispatch routine to perform the requested service, and then sends back a reply.

The `clnt_call()`, `rpc_call()`, and `rpc_broadcast()` routines handle the client side of the procedure call. The remaining routines deal with error handling in the case of errors.

Some of the routines take a `CLIENT` handle as one of the parameters. A `CLIENT` handle can be created by an RPC creation routine such as `clnt_create()` (see `rpc_clnt_create(3N)`). These routines are safe for use in multithreaded applications. `CLIENT` handles can be shared between threads, however, in this implementation requests by different threads are serialized (that is, the first request will receive its results before the second request is sent).

Routines

See `rpc(3N)` for the definition of the `CLIENT` data structure.

```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.

```c
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_errno(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_errno(), 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_errno() 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_errno() is to be used. Note: unlike clnt_sperrno() and clnt_spcreateerror()
    (see rpc_clnt_create(3N)), clnt_sperrno() does not return pointer to static data so
    the result will not get overwritten on each call.

char *clnt_sperror(const CLIENT *clnt, const char *s);
    Like clnt_perror(), except that (like clnt_sperrno()) it returns a string instead of
    printing to standard error. However, clnt_sperror() does not append a newline
    at the end of the message.

    Warning: returns pointer to a buffer that is overwritten on each call. In multi-
    thread applications, this buffer is implemented as thread-specific data.

eenum 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 trans-
    ports specified by nettype. If nettype is NULL, it defaults to "netpath. Each time it
    receives a response, this routine calls eachresult(), whose form is:
bool_t eachresult(caddr_t out, const struct netbuf *addr,  
    const struct netconfig *netconf);

where out is the same as out passed to rpc_broadcast(), except that the remote  
procedure’s output is decoded there; addr points to the address of the machine  
that sent the results, and netconf is the netconfig structure of the transport  
on which the remote server responded. If eachresult() returns 0, rpc_broadcast()  
waits for more replies; otherwise it returns with appropriate status.  

Warning: broadcast file descriptors are limited in size to the maximum transfer  
size of that transport. For Ethernet, this value is 1500 bytes. rpc_broadcast()  
uses AUTH_SYS credentials by default (see rpc_clnt_auth(3N)).

define 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);

Like rpc_broadcast(), except that the initial timeout, inittime and the maximum  
timeout, waittime are specified in milliseconds.  

inittime is the initial time that rpc_broadcast_exp() waits before resending the  
request. After the first resend, the re-transmission interval increases exponen-
tially until it exceeds waittime.

define clnt_stat rpc_call(const char *host, const u_long prognum,  
    const u_long versnum, const u_long procnum, const xdrproc_t inproc,  
    const char *in, const xdrproc_t outproc, char *out, const char *nettype);

Call the remote procedure associated with prognum, versnum, and procnum on the  
machine, host. The parameter inproc is used to encode the procedure’s parame-
ters, and outproc is used to decode the procedure’s results; in is the address of the  
procedure’s argument(s), and out is the address of where to place the result(s).  
nettype can be any of the values listed on rpc(3N). This routine returns  
RPC_SUCCESS if it succeeds, or an appropriate status is returned. Use the  
clnt_perrno() routine to translate failure status into error messages.  

Warning: rpc_call() uses the first available transport belonging to the class net-
type, on which it can create a connection. You do not have control of timeouts or  
authentication using this routine.

SEE ALSO printf(3S), rpc(3N), rpc_clnt_auth(3N), rpc_clnt_create(3N)
NAME

rpc_clnt_create, clnt_control, clnt_create, clnt_create_timed, clnt_create_vers,
clnt_destroy, clnt_dg_create, clnt_pcreateerror, clnt_raw_create, clnt_spcreateerror,
clnt_tli_create, clnt_tp_create, clnt_tp_create_timed, clnt VC_create, rpc_createerr –
library routines for dealing with creation and manipulation of CLIENT handles

MT-LEVEL

MT-Safe

DESCRIPTION

RPC library routines allow C language programs to make procedure calls on other
machines across the network. First a CLIENT handle is created and then the client calls a
procedure to send a request to the server. On receipt of the request, the server calls a
dispatch routine to perform the requested service, and then sends a reply.

These routines are MT-Safe. In the case of multithreaded applications, the _REENTRANT
flag must be defined on the command line at compilation time
(−D_REENTRANT). When the _REENTRANT flag is defined, rpc_createerr becomes a
macro which enables each thread to have its own rpc_createerr.

Routines

See rpc(3N) for the definition of the CLIENT data structure.

#include <rpc/rpc.h>

bool_t clnt_control(CLIENT ∗clnt, const 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
desciptor when destroying
the client handle

CLGET_VERS unsigned long ∗ get the RPC program’s version
number associated with the
client handle

3N-930  modified 26 Jan 1995
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 clnt_control(). This routine returns NULL if it fails. The clnt_pcreateerror() routine can be used to print the reason for failure.

Note: clnt_create() returns a valid client handle even if the particular version number supplied to clnt_create() is not registered with the rpcbind service. This mismatch will be discovered by a clnt_call later (see rpc_clnt_calls(3N)).

CLIENT *clnt_create_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);

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)). How-
ever, clnt_create_vers() does this for you and returns a valid handle only if a ver-
sion within the range supplied is supported by the server.

void clnt_destroy(CLIENT *clnt);

A function macro that destroys the client’s RPC handle. Destruction usually
involves deallocation of private data structures, including clnt itself. Use of clnt
is undefined after calling clnt_destroy(). If the RPC library opened the associated
file descriptor, or CLSET_FD_CLOSE was set using clnt_control(), the file descrip-
tor will be closed.

The caller should call auth_destroy(clnt→cl_auth) (before calling clnt_destroy())
to destroy the associated AUTH structure (see rpc_clnt_auth(3N)).

CLIENT *clnt_dg_create(const int filedes, 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 filedes 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.

modified 26 Jan 1995
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.

SEE ALSO rpc(3N), rpc_clnt_auth(3N), rpc_clnt_calls(3N), rpcbind(1M)
NAME  rpc_control – library routine for manipulating global RPC attributes for client and server applications

SYNOPSIS  bool_t rpc_control(int op, void *info);

MT-LEVEL  MT-Safe

DESCRIPTION  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

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.

modified 22 Oct 1993 3N-935
RETURN VALUES

This routine returns **TRUE** if the operation was successful, and **FALSE** otherwise.

SEE ALSO

rpc(3N), rpc_svc_calls(3N), rpcbind(1M)
NAME rpc_rac, rac_drop, rac_poll, rac_recv, rac_send – remote asynchronous calls

SYNOPSIS cc [ flag ...] file ... -lrac -lnsl [ library ... ]

#include <rpc/rpc.h>
#include <rpc/rac.h>

MT-LEVEL Unsafe

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.

#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.

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.

```
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(... *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
clint_geterr() immediately after a rac_send() failure (see rpc(3N)). Possible
ergors include both transient problems (such as transport failures) and per-
manent ones (such as XDR encoding failures).

Multiple rac Sends on the same client handle are permitted, but may introduce
unpredictable perturbations to the current timeout and retry model used by the
RPC library.

The interface imposes a limit on the amount of time a call may be in progress
before it is considered to have failed. This method was chosen over limitations
on the number of retries because of a desire for transport independence.

SEE ALSO poll(2), rpc(3N), rpc_clnt_create(3N), rpc_clnt_calls(3N), xdr(3N)

WARNINGS The RAC interface is not the recommended interface for having multiple RPC requests
outstanding. The preferred method of accomplishing this in the Solaris environment is to
use synchronous RPC calls with threads. The RAC interface is provided as a service to
developers interested in porting RPC applications to Solaris 2.0. Use of this interface will
degrade the performance of normal synchronous RPC calls (see rpc_clnt_calls(3N)). For
these reasons, use of this interface is disparaged.

The library librac must be linked before libnsl to use RAC. If the libraries are not linked
in the correct order, then the results are indeterminate.

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

```c
#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 keyserver 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 credential, given in seconds. A small window is more secure than a large one, but
choosing too small of a window will increase the frequency of resynchronizations because of clock drift. The third parameter syncaddr is optional. If it is NULL, then the authentication system will assume that the local clock is always in sync with the server’s clock, and will not attempt resynchronizations. If an address is supplied, however, then the system will use the address for consulting the remote time service whenever resynchronization is required. This parameter is usually the address of the RPC server itself. The final parameter ckey is also optional. If it is NULL, then the authentication system will generate a random DES key to be used for the encryption of credentials. If it is supplied, however, then it will be used instead.

Warning: this routine exists for backward compatibility only, and is obsoleted by authdes_seccreate() (see secure_rpc(3N)).

AUTH * authunix_create(char *host, int uid, int gid, int grouplen, int gidlistp);

Create and return an RPC authentication handle that contains authentication information. The parameter host is the name of the machine on which the information was created; uid is the user’s user ID; gid is the user’s current group ID; grouplen and gidlistp refer to a counted array of groups to which the user belongs.

Warning: it is not very difficult to impersonate a user.

Warning: this routine exists for backward compatibility only, and is obsoleted by authsys_create() (see rpc_clnt_auth(3N)).

AUTH * authunix_create_default(void)

Call authunix_create() with the appropriate parameters.

Warning: this routine exists for backward compatibility only, and is obsoleted by authsys_create_default() (see rpc_clnt_auth(3N)).

callrpc(char *host, u_long prognum, u_long versnum, u_long procnum, xdrproc_t inproc, char *in, xdrproc_t outproc, char *out);

Call the remote procedure associated with prognum, versnum, and procnum on the machine, host. The parameter inproc is used to encode the procedure’s parameters, and outproc is used to decode the procedure’s results; in is the address of the procedure’s argument, and out is the address of where to place the result(s). This routine returns 0 if it succeeds, or the value of enum clnt_stat cast to an integer if it fails. The routine clnt_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 procnun,
    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.

```c
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.

```c
CLIENT * clntudp_create(struct sockaddr_in *addr, u_long prognum, u_long versnum,
                         struct timeval wait, int *fdp);
```

This routine creates an RPC client handle for the remote program `prognum`, version `versnum`; the client uses UDP/IP as a transport. The remote program is located at Internet address `addr`. If `addr->sin_port` is 0, then it is set to actual port that the remote program is listening on (the remote `rpcbind` service is consulted for this information). The parameter `*fdp` is a file descriptor, which may be open and bound; if it is `RPC_ANYSOCK`, then this routine opens a new one and sets `*fdp`. Refer to the File Descriptor section for more information. The UDP transport resends the call message in intervals of `wait` time until a response is received or until the call times out. The total time for the call to time out is specified by `clnt_call()` (see `rpc_clnt_calls(3N)`). `clntudp_create()` returns a client handle on success, otherwise it returns NULL. The error can be printed using the `clnt_pcreateerror()` (see `rpc_clnt_create(3N)`) routine.
Warning: since UDP-based RPC messages can only hold up to 8 Kbytes of encoded data, this transport cannot be used for procedures that take large arguments or return huge results.

Warning: this routine exists for backward compatibility only. clnt_create(), clnt_tli_create(), or clnt_dg_create() (see rpc_clnt_create(3N)) should be used instead.

```c
void get_myaddress(struct sockaddr_in *addr);
```

Places the local system’s IP address into *addr, without consulting the library routines that deal with /etc/hosts. The port number is always set to htons(PMAPPORT).

Warning: this routine is only intended for use with the RPC library. It returns the local system’s address in a form compatible with the RPC library, and should not be taken as the system’s actual IP address. In fact, the *addr buffer’s host address part is actually zeroed. This address may have only local significance and should NOT be assumed to be an address that can be used to connect to the local system by remote systems or processes.

Warning: this routine remains for backward compatibility only. The routine netdir_getbyname() (see netdir(3N)) should be used with the name HOST_SELF to retrieve the local system’s network address as a netbuf structure.

```c
void getrpcport(char *host, int prognum, int versnum, int proto)
```

getrpcport() returns the port number for the version versnum of the RPC program prognum running on host and using protocol proto. getrpcport() returns 0 if the RPC system failed to contact the remote portmap service, the program associated with prognum is not registered, or there is no mapping between the program and a port.

Warning: This routine exists for backward compatibility only. Enhanced functionality is provided by rpcb_getaddr() (see rpcbind(3N)).

```c
struct pmaplist * pmap_getmaps(struct sockaddr_in *addr);
```

A user interface to the portmap service, which returns a list of the current RPC program-to-port mappings on the host located at IP address addr. This routine can return NULL. The command ‘rpcinfo -p’ uses this routine.

Warning: this routine exists for backward compatibility only, enhanced functionality is provided by rpcb_getmaps() (see rpcbind(3N)).

```c
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 \textit{protocol}. The value of \textit{protocol} is most likely \texttt{IPPROTO_UDP} or \texttt{IPPROTO_TCP}. A return value of \texttt{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 \texttt{rpc_createerr} contains the RPC status.

Warning: this routine exists for backward compatibility only, enhanced functionality is provided by \texttt{rpcb_getaddr()} (see \texttt{rpcbind(3N)}).

\begin{verbatim}
enum clnt_stat pmap_rmtcall(struct sockaddr_in *addr, u_long prognum,
   u_long versnum, u_long protocol, char *in, xdrproct_t inproc,
   char *out, xdrproct_t outproc, struct timeval tout, u_long *portp);
\end{verbatim}

Request that the portmap on the host at IP address \texttt{*addr} make an RPC on the behalf of the caller to a procedure on that host. \texttt{*portp} is modified to the program’s port number if the procedure succeeds. The definitions of other parameters are discussed in \texttt{callrpc()} and \texttt{clnt_call()} (see \texttt{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 \texttt{rpcb_rmtcall()} (see \texttt{rpcbind(3N)}).

\begin{verbatim}
bool_t pmap_set(u_long prognum, u_long versnum, u_long protocol, u_short port);
\end{verbatim}

A user interface to the portmap service, that establishes a mapping between the triple \texttt{[prognum, versnum, protocol]} and \texttt{port} on the machine’s portmap service. The value of \textit{protocol} may be \texttt{IPPROTO_UDP} or \texttt{IPPROTO_TCP}. Formerly, the routine failed if the requested \textit{port} was found to be in use. Now, the routine only fails if it finds that \textit{port} is still bound. If \textit{port} is not bound, the routine completes the requested registration. This routine returns \texttt{1} if it succeeds, \texttt{0} otherwise.

Automatically done by \texttt{svc_register()}. 

Warning: this routine exists for backward compatibility only, enhanced functionality is provided by \texttt{rpcb_set()} (see \texttt{rpcbind(3N)}).

\begin{verbatim}
bool_t pmap_unset(u_long prognum, u_long versnum);
\end{verbatim}

A user interface to the portmap service, which destroys all mapping between the triple \texttt{[prognum, versnum, all-protocols]} and \texttt{port} on the machine’s portmap service. This routine returns one if it succeeds, \texttt{0} otherwise.

Warning: this routine exists for backward compatibility only, enhanced functionality is provided by \texttt{rpcb_unset()} (see \texttt{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().

SVCXPRT * svcfd_create(int fd, u_int sendsz, u_int recvsz);
Create a service on top of any open and bound descriptor. Typically, this
descriptor is a connected file descriptor for a stream protocol. Refer to the File
Descriptor section for more information. sendsz and recvsz indicate sizes for the
send and receive buffers. If they are 0, a reasonable default is chosen.
Warning: this interface is obsoleted by svc_fd_create() (see rpc_svc_create(3N)).

SVCXPRT * 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.

`SVCXPRT * svctcp_create(int fd, u_int sendsz, u_int recvsz);`

This routine creates a TCP/IP-based RPC service transport, to which it returns a pointer. The transport is associated with the file descriptor `fd`, which may be `RPC_ANYSOCK`, in which case a new file descriptor is created. If the file descriptor is not bound to a local TCP port, then this routine binds it to an arbitrary port. Refer to the `File Descriptor` section for more information. Upon completion, `xprt→xp_fd` is the transport’s file descriptor, and `xprt→xp_port` is the transport’s port number. This routine returns `NULL` if it fails. Since TCP-based RPC uses buffered I/O, users may specify the size of buffers; values of 0 choose suitable defaults.

Warning: this routine exists for backward compatibility only. `svc_create()`, `svc_tli_create()`, or `svc_vc_create()` (see `rpc_svc_create`(3N)) should be used instead.

`SVCXPRT * svcudp_bufcreate(int fd, u_int sendsz, u_int recvsz);`

This routine creates a UDP/IP-based RPC service transport, to which it returns a pointer. The transport is associated with the file descriptor `fd`. If `fd` is `RPC_ANYSOCK`, then a new file descriptor is created. If the file descriptor is not bound to a local UDP port, then this routine binds it to an arbitrary port. Upon completion, `xprt→xp_fd` is the transport’s file descriptor, and `xprt→xp_port` is the transport’s port number. Refer to the `File Descriptor` section for more information. This routine returns `NULL` if it fails.

The user specifies the maximum packet size for sending and receiving UDP-based RPC messages by using the `sendsz` and `recvsz` parameters.

Warning: this routine exists for backward compatibility only. `svc_tli_create()`, or `svc_dg_create()` (see `rpc_svc_create`(3N)) should be used instead.

`SVCXPRT * svcudp_create(int fd);`

This routine creates a UDP/IP-based RPC service transport, to which it returns a pointer. The transport is associated with the file descriptor `fd`, which may be `RPC_ANYSOCK`, in which case a new file descriptor is created. If the file descriptor is not bound to a local UDP port, then this routine binds it to an arbitrary port. Upon completion, `xprt→xp_fd` is the transport’s file descriptor, and `xprt→xp_port` is the transport’s port number. This routine returns `NULL` if it fails.

Warning: since UDP-based RPC messages can only hold up to 8 Kbytes of encoded data, this transport cannot be used for procedures that take large arguments or return huge results.

Warning: this routine exists for backward compatibility only. `svc_create()`, `svc_tli_create()`, or `svc_dg_create()` (see `rpc_svc_create`(3N)) should be used instead.
registerrpc(u_long prognum, u_long versnum, u_long procnun, char *(procname)(, xdrproc_t inproc, xdrproc_t outproc);

Register program prognum, procedure procname, and version versnum with the RPC service package. If a request arrives for program prognum, version versnum, and procedure procnun, procname is called with a pointer to its parameter(s); procname should return a pointer to its static result(s); inproc is used to decode the parameters while outproc is used to encode the results. This routine returns 0 if the registration succeeded, −1 otherwise.

svc_run() must be called after all the services are registered.
Warning: this routine exists for backward compatibility only, and is obsoleted by rpc_reg().

svc_register(SVCXPRT *xprt, u_long prognum, u_long versnum, void (*dispatch)(, u_long protocol);

Associates prognum and versnum with the service dispatch procedure, dispatch. If protocol is 0, the service is not registered with the portmap service. If protocol is non-zero, then a mapping of the triple [prognum, versnum, protocol] to xprt→xp_port is established with the local portmap service (generally protocol is 0, IPPROTO_UDP or IPPROTO_TCP). The procedure dispatch has the following form:

dispatch(struct svc_req *request, SVCXPRT *xprt);

The svc_register() routine returns one if it succeeds, and 0 otherwise.
Warning: this routine exists for backward compatibility only; enhanced functionality is provided by svc_reg().

void svc_unregister(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().

xdr_authunix_parms(XDR *xdrs, struct authunix_parms *aupp);

Used for describing UNIX credentials. This routine is useful for users who wish to generate these credentials without using the RPC authentication package.
Warning: this routine exists for backward compatibility only, and is obsoleted by xdr_authsys_parms() (see rpc_xdr(3N)).

SEE ALSO keyserv(1M), rpcbind(1M), rpcinfo(1M), rpc(3N), rpc_clnt_auth(3N), rpc_clnt_calls(3N), rpc_clnt_create(3N), rpc_svc_calls(3N), rpc_svc_create(3N), rpc_svc_err(3N), rpc_svc_reg(3N), rpcbind(3N), secure_rpc(3N), select(3C)
NOTES

These interfaces are unsafe in multithreaded applications. Unsafe interfaces should be called only from the main thread.
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_pollset, svc_run, svc_sendreply – library routines for RPC servers

MT-LEVEL
See the NOTES section of this page.

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.

bool_t svc_freeargs(const SVCXPRT *xprt, const xdrproc_t inproc, caddr_t in);
    
    A function macro that frees any data allocated by the RPC/XDR system when it decoded the arguments to a service procedure using svc_getargs(). This routine returns TRUE if the results were successfully freed, and FALSE otherwise.

    This function macro is safe in MT applications utilizing the Automatic or User MT modes.

bool_t svc_getargs(const SVCXPRT *xprt, const xdrproc_t inproc, caddr_t in);
    
    A function macro that decodes the arguments of an RPC request associated with the RPC service transport handle xprt. The parameter in is the address where the arguments will be placed; inproc is the XDR routine used to decode the arguments. This routine returns TRUE if decoding succeeds, and FALSE otherwise.

    This function macro is safe in MT applications utilizing the Automatic or User MT modes.

void svc_getreq_common(const int fd);
    
    This routine is called to handle a request on the given file descriptor.
void svc_getreq_poll(struct pollfd *pfdp, const int pollretval);

This routine is only of interest if a service implementor does not call svc_run(), but instead implements custom asynchronous event processing. It is called when poll(2) has determined that an RPC request has arrived on some RPC file descriptors; pollretval is the return value from poll(2) and pfdp is the array of pollfd structures on which the poll(2) was done. It is assumed to be an array large enough to contain the maximal number of descriptors allowed.

This function macro is unsafe in MT applications.

void svc_getreqset(fd_set *rdfs);

This routine is only of interest if a service implementor does not call svc_run(), but instead implements custom asynchronous event processing. It is called when select(3C) has determined that an RPC request has arrived on some RPC file descriptors; rdfs is the resultant read file descriptor bit mask. The routine returns when all file descriptors associated with the value of rdfs have been serviced.

This function macro is unsafe in MT applications.

struct netbuf *svc_getrpccaller(const SVCXPRT *xprt);

The approved way of getting the network address of the caller of a procedure associated with the RPC service transport handle xprt.

This function macro is safe in MT applications.

void svc_run(void);

This routine never returns. In single threaded mode, it waits for RPC requests to arrive, and calls the appropriate service procedure using svc_getreq_poll() when one arrives. This procedure is usually waiting for the poll(2) library call to return.

Applications executing in the Automatic or User MT modes should invoke this function exactly once. In the Automatic MT mode, it will create threads to service client requests. In the User MT mode, it will provide a framework for service developers to create and manage their own threads for servicing client requests.
bool_t svc_sendreply(const SVCXPRT *xprt, const xdrproc_t outproc, const caddr_t out);

Called by an RPC service’s dispatch routine to send the results of a remote procedure call. The parameter xprt is the request’s associated transport handle; outproc is the XDR routine which is used to encode the results; and out is the address of the results. This routine returns TRUE if it succeeds, FALSE otherwise. This function macro is safe in MT applications utilizing the Automatic or User MT modes.

SEE ALSO
rpcgen(1), poll(2), rpc(3N), rpc_control(3N), rpc_svc_create(3N), rpc_svc_err(3N), rpc_svc_reg(3N), select(3C), xprt_register(3N)

NOTES
svc_dg_enablecache() and svc_getrpccaller() are safe in multithreaded applications. svc_freeargs(), svc_getargs(), and svc_sendreply() are safe in MT applications utilizing the Automatic or User MT modes. svc_getreq_common(), svc_getreqset(), and svc_getreq_poll() are unsafe in multithreaded applications and should be called only from the main thread.
NAME
rpc_svc_create, svc_control, svc_create, svc_destroy, svc_dg_create, svc_fd_create,
svc_raw_create, svc_tli_create, svc_tp_create, svc_vc_create – library routines for the
creation of server handles

MT-LEVEL
MT-Safe

DESCRIPTION
These routines are part of the RPC library which allows C language programs to make
procedure calls on servers across the network. These routines deal with the creation of
service handles. Once the handle is created, the server can be invoked by calling
svc_run().

Routines
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 normally 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 indi-
cates that the out of range request will be silently ignored.

SVCSET_VERSQUIET If a request is received for a program number served by
this server but the version number is outside the range
registered with the server, an RPC_PROGVERSMISMATCH
error will normally be returned. It is sometimes desirable
to change this behavior. info should be a pointer to an
integer which is either 0 (indicating normal server
behavior – an RPC_PROGVERSMISMATCH error will be
returned), or 1 (indicating that the out of range request
should be silently ignored).

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.

void svc_destroy(SVCXPRT *xprt);
A function macro that destroys the RPC service handle xprt. Destruction usually involves deallocation of private data structures, including xprt itself. Use of xprt is undefined after calling this routine.

SVCXPRT *svc_dg_create(const int fildes, const u_int sendsz, const u_int recvsz);
This routine creates a connectionless RPC service handle, and returns a pointer to it. This routine returns NULL if it fails, and an error message is logged. sendsz and recvsz are parameters used to specify the size of the buffers. If they are 0, suitable defaults are chosen. The file descriptor fildes should be open and bound. The server is not registered with rpcbind(1M).

Warning: since connectionless-based RPC messages can only hold limited amount of encoded data, this transport cannot be used for procedures that take large arguments or return huge results.

SVCXPRT *svc_fd_create(const int fildes, const u_int sendsz, const u_int recvsz);
This routine creates a service on top of an open and bound file descriptor, and returns the handle to it. Typically, this descriptor is a connected file descriptor for a connection-oriented transport. sendsz and recvsz indicate sizes for the send and receive buffers. If they are 0, reasonable defaults are chosen. This routine returns NULL if it fails, and an error message is logged.

SVCXPRT *svc_raw_create(void);
This routine creates an RPC service handle and returns a pointer to it. The transport is really a buffer within the process’s address space, so the corresponding RPC client should live in the same address space; (see clnt_raw_create() in rpc_clnt_create(3N)). This routine allows simulation of RPC and acquisition of RPC overheads (such as round trip times), without any kernel and networking interference. This routine returns NULL if it fails, and an error message is logged.

Note: svc_run() should not be called when the raw interface is being used.
**svc_tli_create()**  

```c
SVCXPRT *svc_tli_create(const int fildes, const struct netconfig *netconf,  
const struct t_bind *bindaddr, const u_int sendsz, const u_int recvsz);
```

This routine creates an RPC server handle, and returns a pointer to it. `fildes` is the file descriptor on which the service is listening. If `fildes` is `RPC_ANYFD`, it opens a file descriptor on the transport specified by `netconf`. If the file descriptor is unbound and `bindaddr` is non-null, `fildes` is bound to the address specified by `bindaddr`, otherwise `fildes` is bound to a default address chosen by the transport. In the case where the default address is chosen, the number of outstanding connect requests is set to 8 for connection-oriented transports. The user may specify the size of the send and receive buffers with the parameters `sendsz` and `recvsz`; values of 0 choose suitable defaults. This routine returns `NULL` if it fails, and an error message is logged. The server is not registered with the `rpcbind(1M)` service.

**svc_tp_create()**  

```c
SVCXPRT *svc_tp_create(const void (*dispatch)(const struct svc_req *,  
const SVCXPRT *), const u_long prognum, const u_long versnum,  
const struct netconfig *netconf);
```

`svc_tp_create()` creates a server handle for the network specified by `netconf`, and registers itself with the `rpcbind` service. `dispatch` is called when there is a remote procedure call for the given `prognum` and `versnum`; this requires calling `svc_run()`. `svc_tp_create()` returns the service handle if it succeeds, otherwise a `NULL` is returned and an error message is logged.

**svc_vc_create()**  

```c
SVCXPRT *svc_vc_create(const int fildes, const u_int sendsz, const u_int recvsz);
```

This routine creates a connection-oriented RPC service and returns a pointer to it. This routine returns `NULL` if it fails, and an error message is logged. The users may specify the size of the send and receive buffers with the parameters `sendsz` and `recvsz`; values of 0 choose suitable defaults. The file descriptor `fildes` should be open and bound. The server is not registered with the `rpcbind(1M)` service.

**SEE ALSO**  

`rpcbind(1M)`, `rpc(3N)`, `rpc_svc_calls(3N)`, `rpc_svc_err(3N)`, `rpc_svc_reg(3N)`
NAME rpc_svc_err, svcerr_auth, svcerr_decode, svcerr_noproc, svcerr_noprog, svcerr_progvers, svcerr_systemerr, svcerr_weakauth – library routines for server side remote procedure call errors

MT-LEVEL MT-Safe

DESCRIPTION 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.

```c
#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).

SEE ALSO rpc(3N), rpc_svc_calls(3N), rpc_svc_create(3N), rpc_svc_reg(3N)
NAME    rpc_svc_reg, rpc_reg, svc_reg, svc_unreg, svc_auth_reg, xprt_register, xprt_unregister — library routines for registering servers

MT-LEVEL MT-Safe

DESCRIPTION These routines are a part of the RPC library which allows the RPC servers to register themselves with rpcbind() (see rpcbind(1M)), and associate the given program and version number with the dispatch function. When the RPC server receives a RPC request, the library invokes the dispatch routine with the appropriate arguments.

Routines See rpc(3N) for the definition of the SVCXPRT data structure.

#include <rpc/rpc.h>

bool_t rpc_reg(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 prognum, procedure procname, and version versnum with the RPC service package. If a request arrives for program prognum, version versnum, and procedure procnum, procname is called with a pointer to its parameter(s); procname should return a pointer to its static result(s). The arg parameter to procname is a pointer to the (decoded) procedure argument. inproc is the XDR function used to decode the parameters while outproc is the XDR function used to encode the results. Procedures are registered on all available transports of the class nettype. See rpc(3N). This routine returns 0 if the registration succeeded, −1 otherwise.

int svc_reg(const SVCXPRT *xprt, const u_long prognum, const u_long versnum,  
            const void (*dispatch), const struct netconfig *netconf);

Associates prognum and versnum with the service dispatch procedure, dispatch. If netconf is NULL, the service is not registered with the rpcbind service. For example, if a service has already been registered using some other means, such as inetd (see inetd(1M)), it will not need to be registered again. If netconf is non-zero, then a mapping of the triple [prognum, versnum, netconf→nc_netid] to -xprt→xp_ltaddr is established with the local rpcbind service.

The svc_reg() routine returns 1 if it succeeds, and 0 otherwise.

void svc_unreg(const u_long prognum, const u_long versnum);

Remove from the rpcbind service, all mappings of the triple [prognum, versnum, all-transports] to network address and all mappings within the RPC service package of the double [prognum, versnum] to dispatch routines.

modified 23 May 1995
int svc_auth_reg(const int cred_flavor, const enum auth_stat (*handler));

Registers the service authentication routine handler with the dispatch mechanism so that it can be invoked to authenticate RPC requests received with authentication type cred_flavor. This interface allows developers to add new authentication types to their RPC applications without needing to modify the libraries. Service implementors usually do not need this routine.

Typical service application would call svc_auth_reg() after registering the service and prior to calling svc_run(). When needed to process an RPC credential of type cred_flavor, the handler procedure will be called with two parameters (struct svc_req *rqst, struct rpc_msg *msg) and is expected to return a valid enum auth_stat value. There is no provision to change or delete an authentication handler once registered.

The svc_auth_reg() routine returns 0 if the registration is successful, 1 if cred_flavor already has an authentication handler registered for it, and −1 otherwise.

void xprt_register(const SVCXPRT *xprt);

After RPC service transport handle xprt is created, it is registered with the RPC service package. This routine modifies the global variable svc_fdset (see rpc_svc_calls(3N)). Service implementors usually do not need this routine.

void xprt_unregister(const SVCXPRT *xprt);

Before an RPC service transport handle xprt is destroyed, it unregisters itself with the RPC service package. This routine modifies the global variable svc_fdset (see rpc_svc_calls(3N)). Service implementors usually do not need this routine.

SEE ALSO inetd(1M), rpcbind(1M), rpc(3N), rpc_svc_calls(3N), rpc_svc_create(3N), rpc_svc_err(3N), rpcbind(3N), select(3C)
## NAME
rpc_xdr, xdr_accepted_reply, xdr_authsys_parms, xdr_callhdr, xdr_callmsg,
xdr_opaque_auth, xdr_rejected_reply, xdr_replymsg – XDR library routines for remote
procedure calls

## MT-LEVEL
Safe

## DESCRIPTION
These routines are used for describing the RPC messages in XDR language. They should
normally be used by those who do not want to use the RPC package directly. These rou-
tines 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 ver-
sion number mis-match or because of authentication errors.
```
bool_t xdr_replymsg(XDR *xdrs, const struct rpc_msg *rmsg);

Used for describing RPC reply messages. It translates between the RPC reply message and its external representation. This reply could be either an acceptance, rejection or NULL.

SEE ALSO rpc(3N), xdr(3N)
NAME rpcbind, rpcb_getmaps, rpcb_getaddr, rpcb_gettime, rpcb_rmtcall, rpcb_set, rpcb_unset — library routines for RPC bind service

MT-LEVEL MT-Safe

DESCRIPTION These routines allow client C programs to make procedure calls to the RPC binder service. rpcbind (see rpcbind(1M)) maintains a list of mappings between programs and their universal addresses.

Routines

#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 prognum, const u_long versnum, const struct netconfig *netconf, struct netbuf *svcaddr, const char *host);

An interface to the rpcbind service, which finds the address of the service on host that is registered with program number prognum, 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.

does not contain any additional content that cannot be directly transcribed from the page.
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-ports] and the addresses on the machine’s rpcbind service. This routine returns TRUE if it succeeds, FALSE otherwise. Only the owner of the service or the super-user can destroy the mapping. (See also svc_unreg() in rpc_svc_calls(3N)).

SEE ALSO  rpcbind(1M), rpcinfo(1M), rpc_clnt_calls(3N), rpc_svc_calls(3N)
NAME          rstat, havedisk – get performance data from remote kernel

PROTOCOL      /usr/include/rpcsvc/rstat.x

SYNOPSIS      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);

MT-LEVEL       MT-Safe

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

SEE ALSO      rpc.rstatd(1M), rup(1), rpc_clnt_calls(3N)
NAME  rusers, rnusers – return information about users on remote machines

PROTOCOL  /usr/include/rpcs svc/rusers.x

SYNOPSIS  cc [ flag  ... ] file  ... -lrpcs vc [ library  ... ]
#include <rpc/rpc.h>
#include <rpcsvc/rusers.h>
enum clnt_stat rusers(char *host, struct utmpidlearr *up);
int rnusers(char *host);

MT-LEVEL  MT-Safe

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_sperrno(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 librpcs vc:

    xdr_utmpidlearr.

SEE ALSO  rusers(1), rpc.rusersd(1M), rpc_clnt_calls(3N), xdr_free(3N)
NAME         rwall – write to specified remote machines

PROTOCOL     /usr/include/rpcsrv/rwall.x

SYNOPSIS     cc [ flag ... ] file ... -lrpcsrv [ library ... ]
             #include <rpc/rpc.h>
             #include <rpcsvc/rwall.h>
             enum clnt_stat rwall(char *host, char *msg);

MT-LEVEL      MT-Safe

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_perror(3N).

SEE ALSO      rpc.rwalld(1M), wall(1M), rpc_clnt_calls(3N)
NAME  rwlock, rwlock_init, rwlock_destroy, rw_rdlock, rw_wrlock, rw_tryrdlock, rw_trywrlock, rw_unlock – multiple readers, single writer locks

SYNOPSIS  cc [ flag ... ] file ... - pthreads -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);

MT-LEVEL  MT-Safe

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.

stype 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 lock. arg is ignored.

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. A readers/writer lock must not be reinitialized while in use by other threads.

The following are default readers/writer lock initialization (intra-process):

rwlock_t rwlp;

rwlock_init(&rwlp, NULL, NULL);
The following is a customized readers/writer lock initialization (inter-process):

```c
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.
SEE ALSO  mmap(2)

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 depriv- ing writers. The current implementation favors writers over readers.
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  scandir() 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. alphasort() is a routine that sorts the array alphabetically.

scandir() returns the number of entries in the array and a pointer to the array through the parameter namelist.

RETURN VALUES  Returns −1 if the directory cannot be opened for reading or if malloc(3C) cannot allocate enough memory to hold all the data structures.

SEE ALSO  getdents(2), readdir(3B), directory(3C), malloc(3C), qsort(3C)

NOTES  Use of these interfaces should be restricted to only applications written on BSD platforms. Use of these interfaces with any of the system libraries or in multi-thread applications is unsupported.
NAME
scanf, fscanf, sscanf – convert formatted input

SYNOPSIS
#include <stdio.h>
int scanf(const char *format, . . .);
int fscanf(FILE *strm, const char *format, . . .);
int sscanf(const char *s, const char *format, . . .);

MT-LEVEL
MT-Safe

DESCRIPTION
scanf() reads from the standard input stream, stdin.
fscanf() reads from the stream strm.
sscanf() 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 %digits$, an optional assignment suppression character *, a decimal
digit string that specifies an optional numerical maximum field width, an
optional letter l (ell), L, or h indicating the size of the receiving object, and a
conversion code:
% or digit, *, decimal digit string, h or l or L, conversion code

The following defines which size indicators can used with which conversion
codes, and the size they indicate.
### Conversion Code

<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>int</td>
</tr>
<tr>
<td>h</td>
<td></td>
<td>short int</td>
</tr>
<tr>
<td>l</td>
<td></td>
<td>long int</td>
</tr>
<tr>
<td>o, u, x</td>
<td>none</td>
<td>unsigned int</td>
</tr>
<tr>
<td>h</td>
<td></td>
<td>unsigned short int</td>
</tr>
<tr>
<td>l</td>
<td></td>
<td>unsigned long int</td>
</tr>
<tr>
<td>e, f, g</td>
<td>none</td>
<td>float</td>
</tr>
<tr>
<td>l</td>
<td></td>
<td>double</td>
</tr>
<tr>
<td>L</td>
<td></td>
<td>long double</td>
</tr>
</tbody>
</table>

The h, l, or L modifier is ignored with any other conversion codes.

A conversion specification directs the conversion of the next input field; the result is placed in the variable pointed to by the corresponding argument unless assignment suppression was indicated by the character *.

The suppression of assignment provides a way of describing an input field that is to be skipped. An input field is defined as a string of non-space characters; it extends to the next inappropriate character or until the maximum field width, if one is specified, is exhausted. For all descriptors except the character [ and the character c, white space leading an input field is ignored.

Conversions can be applied to the 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 (see `strtol(3C)`) with the value 10 for the base argument. The corresponding argument should be a pointer to unsigned integer.
- `o`: Matches an optionally signed octal integer, whose format is the same as expected for the subject sequence of the `strtoul()` function with the value 8 for the base argument. The corresponding argument should be a pointer to unsigned integer.
- `x`: Matches an optionally signed hexadecimal integer, whose format is the
same as expected for the subject sequence of the `strtoul()` function with the value 16 for the base argument. The corresponding argument should be a pointer to unsigned integer.

i Matches an optionally signed integer, whose format is the same as expected for the subject sequence of the `strtol()` function with the value 0 for the base argument. The corresponding argument should be a pointer to integer.

n No input is consumed. The corresponding argument should be a pointer to integer into which is to be written the number of characters read from the input stream so far by the call to the function. Execution of a %n directive does not increment the assignment count returned at the completion of execution of the function.

e.f.g Matches an optionally signed floating point number, whose format is the same as expected for the subject string of the `strtod` function. The corresponding argument should be a pointer to floating.

s A character string is expected; the corresponding argument should be a character pointer pointing to an array of characters large enough to accept the string and a terminating \0, which will be added automatically. The input field is terminated by a white-space character.

ws A wide character string is expected; the corresponding argument should be a wide character pointer pointing to an array of wide characters large enough to accept the wide character string and a terminating \0, which will be added automatically. The input field is terminated by a white-space character.

c Matches a sequence of characters of the number specified by the field width (1 if no field width is present in the directive). The corresponding argument should be a pointer to the initial character of an array large enough to accept the sequence. No null character is added. The normal skip over white space is suppressed.

wc Matches a sequence of wide characters of the number specified by the field width (1 if no field width is present in the directive). The corresponding argument should be a pointer to the initial character of an array large enough to accept the sequence. No null character is added. The normal skip over white space is suppressed.

[ Matches a nonempty sequence of characters from a set of expected characters (the scanset). The corresponding argument should be a pointer to the initial character of an array large enough to accept the sequence and a terminating null character, which will be added automatically. The conversion specifier includes all subsequent characters in the format string, up to and including the matching right bracket (]). The characters between the brackets (the scanlist) comprise the scanset, unless the character after the left bracket is a circumflex (^), in which case the scanset contains all characters that do not appear in the scanlist between the circumflex and the right bracket. If the conversion specifier begins with [] or [^], the right bracket
character is in the scanlist and the next right bracket character is the matching right bracket that ends the specification; otherwise the first right bracket character is the one that ends the specification.

A range of characters in the scanset may be represented by the construct first – last; thus [0123456789] may be expressed [0–9]. Using this convention, first must be lexically less than or equal to last, or else the dash will stand for itself. The character – will also stand for itself whenever it is the first or the last character in the scanlist. To include the right bracket as an element of the scanset, it must appear as the first character (possibly preceded by a circumflex) of the scanlist and in this case it will not be syntactically interpreted as the closing bracket. At least one character must match for this conversion to be considered successful.

p Matches the set of implementation-defined sequences produced as output by the %p conversion of the printf(3S) function. The corresponding argument should be a pointer to void. If the input item is a value converted earlier during the same program execution, the pointer that results compares equal to that value; otherwise, the behavior of the %p conversion is undefined.

If an invalid conversion character follows the %, the results of the operation may not be predictable.

The conversion specifiers E, G, and X are also valid and, under the -Xa and -Xc compilation modes (see cc(1B)), behave the same as e, g, and x, respectively. Under the -Xt compilation mode, E, G, and X behave the same as le, lg, and lx, respectively.

Each function allows for detection of a language-dependent decimal point character in the input string. The decimal point character is defined by the program's locale (category LC_NUMERIC). In the "C" locale, or in a locale where the decimal point character is not defined, the decimal point character defaults to a period (.)

The scanf() conversion terminates at end of file, at the end of the control string, or when an input character conflicts with the control string.

If end-of-file is encountered during input, conversion is terminated. If end-of-file occurs before any characters matching the current directive have been read (other than leading white space, where permitted), execution of the current directive terminates with an input failure; otherwise, unless execution of the current directive is terminated with a matching failure, execution of the following directive (if any) is terminated with an input failure.

If conversion terminates on a conflicting input character, the offending input character is left unread in the input stream. Trailing white space (including new-line characters) is left unread unless matched by a directive. The success of literal matches and suppressed assignments is not directly determinable other than via the %n directive.

LC_NUMERIC

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

RETURN VALUES
These routines return the number of successfully matched and assigned input items; this number can be 0 in the event of an early matching failure between an input character and the control string. If the input ends before the first matching failure or conversion, EOF is returned.

EXAMPLES
The call to the function `scanf()`:

```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.

FILES
`/usr/lib/locale/locale/LC_NUMERIC/numeric`

`LC_NUMERIC` database for `locale`

SEE ALSO
`cc(1B), strtod(3C), strtol(3C), printf(3S)`
NAME sched_get_priority_max, sched_get_priority_min, sched_rr_get_interval – get scheduling parameter limits

SYNOPSIS

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 */
};

MT-LEVEL MT-Safe

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.

SEE ALSO sched_setparam(3R), sched_setscheduler(3R)

BUGS

In Solaris 2.5, these functions always return −1 and set errno to ENOSYS, because this release does not support the Priority Scheduling option. It is our intention to provide support for these interfaces in future releases.

modified 20 Aug 1993
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 */
    ...
}

MT-LEVEL  MT-Safe

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. Other-
wise, 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 run-
n ing 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.

SEE ALSO

intro(2), exec(2), sched_setscheduler(3R)

BUGS

In Solaris 2.5, these functions always return −1 and set errno to ENOSYS, because this release does not support the Priority Scheduling option. It is our intention to provide support for these interfaces in future releases.
NAME sched_setscheduler, sched_getscheduler — set/get scheduling policy and scheduling parameters

SYNOPSIS cc [flag ...] file ... -lposix4 [ library ... ]
#include <sched.h>

int sched_setscheduler(int pid, int policy, const struct sched_param *param);
int sched_getscheduler(int pid);

struct sched_param {
    int sched_priority;    /* process execution scheduling priority */
    ...
}

MT-LEVEL MT-Safe

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()) user ID of the target process unless the effective user ID of the calling process is super-user. 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`.

**SEE ALSO**

`priocntl(1), intro(2), exec(2), priocntl(2), sched_get_priority_max(3R), sched_setparam(3R)`

**BUGS**

In Solaris 2.5, these functions always return -1 and set `errno` to ENOSYS, because this release does not support the Priority Scheduling option. It is our intention to provide support for these interfaces in future releases.
NAME  sched_yield – yield processor

SYNOPSIS  cc [ flag ... ] file ... -lposix4 [ library ... ]
           #include <sched.h>
           int sched_yield(void);

MT-LEVEL  MT-Safe

DESCRIPTION  sched_yield() forces the running process to relinquish the processor until the process
             again becomes the head of its process list.

RETURN VALUES  If successful, sched_yield() returns 0, otherwise, it returns -1, and sets errno to indicate
                 the error condition.

ERRORS  ENOSYS  sched_yield() is not supported by this implementation.

BUGS  In Solaris 2.5, these functions always return –1 and set errno to ENOSYS, because this
      release does not support the Priority Scheduling option. It is our intention to provide
      support for these interfaces in future releases.
### NAME

secure_rpc, authdes_getucred, authdes_seccreate, getnetname, host2netname, key_decryptsession, key_encryptsession, key_gendes, key_setsecret, key_secretkey_is_set, netname2host, netname2user, user2netname – library routines for secure remote procedure calls

### MT-LEVEL

MT-Safe

### DESCRIPTION

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 keyserver daemon `keyserv` (see `keyserv(1M)`) must be running for the **AUTH_DES** authentication system to work, and `keylogin(1)` must have been run. Only the **AUTH_DES** style of authentication is discussed here. For information about the **AUTH_NONE** and **AUTH_SYS** styles of authentication, refer to `rpc_clnt_auth(3N)`. For information about the **AUTH_KERB** style of authentication, refer to `kerberos_rpc(3N)`. 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.

```c
#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).

---

modified 20 Apr 1995

3N-983
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
  key_encryptsession(), key_gendes(), and key_setsecret().

  key_decryptsession() takes a server netname remotename and a DES key deskey,
  and decrypts the key by using the the public key of the the server and the secret
  key associated with the effective UID of the calling process. It is the inverse of
  key_encryptsession().

int key_encryptsession(const char *remotename, des_block *deskey);

  key_encryptsession() is a keyserver interface routine. It takes a server netname
  remotename and a DES key deskey, and encrypts it using the public key of the the
  server and the secret key associated with the effective UID of the calling process.
  It is the inverse of key_decryptsession(). This routine returns 0 if it succeeds, −1
  if it fails.

int key_gendes(des_block *deskey);

  key_gendes() is a keyserver interface routine. It is used to ask the keyserver for a
  secure conversation key. Choosing one at random is usually not good enough,
  because the common ways of choosing random numbers, such as using the
  current time, are very easy to guess. This routine returns 0 if it succeeds, −1 if it
  fails.

int key_setsecret(const char *key);

  key_setsecret() is a keyserver interface routine. It is used to set the key for the
  effective UID of the calling process. This routine returns 0 if it succeeds, −1 if it
  fails.

int key_secretkey_is_set(void);

  key_secretkey_is_set() is a keyserver interface routine that may be used to 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.

int netname2host(const char *name, char *host, const int hostlen);

  Convert from an operating system independent netname name to a domain-
  specific hostname host. hostlen is the maximum size of host. Returns 1 if it
  succeeds, and 0 if it fails. Inverse of host2netname().

int netname2user(const char *name, uid_t *uidp, gid_t *gidp,
int *gidlenp, gid_t gidlist[NGROUPS]);

  Convert from an operating system independent netname to a domain-specific
  user ID. Returns 1 if it succeeds, and 0 if it fails. Inverse of user2netname().

modified 20 Apr 1995
*uidp is set to the user's numerical ID associated with name. *gidp is set to the numerical ID of the user's group. gidlist contains the numerical IDs of the other groups to which the user belongs. *gidlenp is set to the number of valid group ID entries in gidlist.

```c
int user2netname(char name[MAXNETNAMELEN+1], const uid_t uid,
                const char *domain);
```

Convert from a domain-specific username to an operating system independent netname. Returns 1 if it succeeds, and 0 if it fails. Inverse of `netname2user()`.

**SEE ALSO** chkey(1), keyserv(1M), newkey(1M), kerberos_rpc(3N), rpc(3N), rpc_clnt_auth(3N)
NAME  select – synchronous I/O multiplexing

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

int select(int nfds, fd_set *readfds, fd_set *writefds, fd_set *exceptfds,
           struct timeval *timeout);

void FD_SET(int fd, fd_set &fdset);
void FD_CLR(int fd, fd_set &fdset);
int FD_ISSET(int fd, fd_set &fdset);
void FD_ZERO(fd_set &fdset);

MT-LEVEL  MT-Safe

DESCRIPTION  
select() examines the I/O file descriptor sets whose addresses are passed in readfds, writefds, and exceptfds to see if any of their file descriptors are ready for reading, are ready for writing, or have an exceptional condition pending, respectively. Out-of-band data is the only exceptional condition. nfds is the number of bits to be checked in each bit mask that represents a file descriptor; the file descriptors from 0 to nfds −1 in the file descriptor sets are examined. On return, select() replaces the given file descriptor sets with subsets consisting of those file descriptors that are ready for the requested operation. The return value from the call to select() is the number of ready file descriptors.

The file descriptor sets are stored as bit fields in arrays of integers. The following macros are provided for manipulating such file descriptor sets: FD_ZERO() initializes a file descriptor set fdset to the null set. FD_SET() includes a particular file descriptor fd in fdset. FD_CLR() removes fd from fdset. FD_ISSET() is nonzero if fd is a member of fdset, zero otherwise. The behavior of these macros is undefined if a file descriptor value is less than zero or greater than or equal to FD_SETSIZE. FD_SETSIZE is a constant defined in <sys/select.h>.

If timeout is not a NULL pointer, it specifies a maximum interval to wait for the selection to complete. If timeout is a NULL pointer, the select() blocks indefinitely. To effect a poll, the timeout argument should be a non-NULL pointer, pointing to a zero-valued timeval structure.

Any of readfds, writefds, and exceptfds may be given as NULL pointers if no file descriptors are of interest.

RETURN VALUES  
select() returns the number of ready file descriptors contained in the file descriptor sets or −1 if an error occurred. If the time limit expires, then select() returns 0.

ERRORS  
The call fails if:
EBADF  One of the I/O file descriptor sets specified an invalid I/O file descriptor.

modified 22 Jan 1993
EINTR  A signal was delivered before any of the selected events occurred, or the
time limit expired.

EINVAL  A component of the pointed-to time limit is outside the acceptable
range: t_sec must be between 0 and 10^8, inclusive. t_usec must be
greater than or equal to 0, and less than 10^6.

SEE ALSO  poll(2), read(2), write(2)

NOTES  The default value for FD_SETSIZE (currently 1024) is larger than the default limit on the
number of open files. In order to accommodate programs that may use a larger number
of open files with select(), it is possible to increase this size within a program by provid-
ing a larger definition of FD_SETSIZE before the inclusion of <sys/types.h>.
The file descriptor sets are always modified on return, even if the call returns as the result
of a timeout.
NAME     sem_close – close a named semaphore

SYNOPSIS  cc [ flag ... ] file ... -lpix4 [ library ... ]
#include <semaphore.h>
int sem_close(sem_t *sem);
typedef struct {
    ...
} sem_t;  /* opaque POSIX.4 semaphore*/

MT-LEVEL  MT-Safe

DESCRIPTION  sem_close() is used to indicate that the calling process is finished using the named semaphore sem. sem_close() deallocates any system resources for use by this process for this semaphore. If the semaphore has not been removed with a successful call to sem_unlink(3R), then sem_close() has no effect on the state of the semaphore. If sem_close(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.

sem_close() 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  sem is not a valid semaphore descriptor.
ENOSYS    sem_close() is not supported by this implementation.

SEE ALSO   sem_init(3R), sem_open(3R), sem_unlink(3R)

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.

modified 9 Aug 1993
NAME  sem_destroy – destroy an unnamed semaphore

SYNOPSIS  cc [ flag ... ] file ... -lposix4 [ library ... ]
#include <semaphore.h>
int sem_destroy(sem_t *sem);
typedef struct {
   ...}
sem_t; /*opaque POSIX.4 semaphore*/

MT-LEVEL  MT-Safe

DESCRIPTION  sem_destroy( ) 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  sem is not a valid semaphore.
ENOSYS  sem_destroy( ) is not supported by this implementation.
EBUSY  Other processes (or LWPs or threads) are currently blocked on the semaphore.

SEE ALSO  sem_init(3R), sem_open(3R)

BUGS  In Solaris 2.5, 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_getvalue – get the value of a semaphore

SYNOPSIS cc [ flag ... ] file ... -lposix4 [ library ... ]
#include <semaphore.h>
int sem_getvalue(sem_t *sem, int *sval);
typedef struct {
    ...
} sem_t; /*opaque POSIX.4 semaphore*/

MT-LEVEL MT-Safe

DESCRIPTION sem_getvalue() 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 zero or positive. If sval is zero, 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 sem does not refer to a valid semaphore.
ENOSYS sem_getvalue() is not supported by this implementation.

SEE ALSO sem_post(3R), sem_wait(3R)

BUGS In Solaris 2.5, 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.

modified 12 Aug 1993 3R-991
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);

typedef struct {

} sem_t; /*opaque POSIX.4 semaphore*/

MT-LEVEL MT-Safe

DESCRIPTION sem_init() 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 value exceeds SEM_VALUE_MAX.
ENOSPC A resource required to initialize the semaphore has been exhausted.

The resources have reached the limit on semaphores, SEM_NSEMS_MAX.
ENOSYS sem_init() is not supported by this implementation.
EPERM The calling process lacks the appropriate privileges to initialize the semaphore.

SEE ALSO sem_destroy(3R), sem_post(3R), sem_wait(3R)

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_open – initialize/open a named semaphore

SYNOPSIS   cc [ flag …] file … -lpthread4 [ library …]
#include <semaphore.h>

sem_t *sem_open(const char *name, int oflag,
                 /∗ unsigned long mode, unsigned int value */ …);

typedef struct {
    …
} sem_t;       /∗opaque POSIX.4 semaphore*/

MT-LEVEL    MT-Safe

DESCRIPTION sem_open() 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.

name 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.

oflag 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 sema-
phore 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.

modified 12 Aug 1993 3R-993
EEXIST  O_CREAT and O_EXCL are set and the named semaphore already exists.
EINTR  sem_open() was interrupted by a signal.
EINVAL  name is not a valid name.
EMFILE  The number of open semaphore descriptors in this process exceeds
        {SEM_NSEMS_MAX}.
        The number of open file descriptors in this process exceeds {OPEN_MAX}.
ENAMETOOLONG  The string-length of name exceeds {PATH_MAX}, or a pathname component is longer than
                {NAME_MAX} while _POSIX_NO_TRUNC is in effect.
ENFILE  The system file table is full.
ENOENT  O_CREAT is not set and the named semaphore does not exist.
ENOSPC  There is insufficient space for the creation of the new named semaphore.
ENOSYS  sem_open() is not supported by this implementation.
SEE ALSO  exec(2), exit(2), umask(2), sysconf(3C), sem_close(3R), sem_post(3R), sem_unlink(3R),
            sem_wait(3R)
BUGS  In Solaris 2.5, 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_post – increment the count of a semaphore

SYNOPSIS  cc [ flag ...] file ... -lposix4 [ library ...]
#include <semaphore.h>
int sem_post(sem_t *sem);
typedef struct {
...
} sem_t /*opaque POSIX.4 semaphore*/

MT-LEVEL  Async-Signal-Safe

DESCRIPTION  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 threads) were blocked for the semaphore, then its value is incremented by one.

sem_post() is reentrant with respect to signals (ASYNC-SAFE), and may be invoked from a signal-catch function. The semaphore functionality described on this man page is for the POSIX 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))

SEE ALSO  sched_setscheduler(3R), sem_wait(3R), semaphore(3T)

NOTES  sem_wait(3R) and sem_trywait(3R) 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.

modified 25 Mar 1993
### NAME
sem_unlink – remove a named semaphore

### SYNOPSIS

```c
cc [ flag ... ] file ... -lposix4 [ library ... ]
#include <semaphore.h>
int sem_unlink(const char *name);
```

### MT-LEVEL
MT-Safe

### 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.

### SEE ALSO
`exec(2)`, `exit(2)`, `sem_close(3R)`, `sem_open(3R)`

### 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.

---

3R-996

modified 12 Aug 1993
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);
           typedef struct {
               ...
           } sem_t; /*opaque POSIX.4 semaphore*/

MT-LEVEL  MT-Safe

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 unsuccessful, the state of the semaphore 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():

modified 20 Aug 1993 3R-997
sem_wait (3R) Realtime Library SunOS 5.5

#include <errno.h>
define TELLERS 10
sem_t bank_line; /* semaphore */
int banking_hours(), deposit_withdrawal;
void *customer(), do_business(), skip_banking_today();
thread_t tid;
...

sem_init(&bank_line,TRUE,TELLERS); /* 10 tellers available */
while(banking_hours())
thr_create(NULL, NULL, customer, (void *)deposit_withdrawal,
THREAD_NEW_LWP, &tid);
...

void *
customer(deposit_withdrawal)
void *deposit_withdrawal;
{
  int this_customer, in_a_hurry = 50;
  this_customer = rand() % 100;
  if (this_customer == in_a_hurry) {
    if (sem_trywait(&bank_line) != 0)
      if (errno == EAGAIN) { /* no teller available */
        skip_banking_today(this_customer);
        return;
      } /*else go immediately to available teller & decrement bank_line*/
  }
  else
    sem_wait(&bank_line); /* wait for next teller, then proceed,
                        and decrement bank_line */

  do_business((int *)deposit_withdrawal);
  sem_post(&bank_line); /* increment bank_line;
                this_customer's teller
                is now available */
}

SEE ALSO sem_post(3R)

NOTES sem_wait() can be interrupted by a signal, which may result in its premature return.
sem_post(3R) increments the semaphore upon its successful return.
BUGS

In Solaris 2.5, these functions always return -1 and set \texttt{errno} to \texttt{ENOSYS}, because this release does not support the Semaphores option. It is our intention to provide support for these interfaces in future releases.
NAME
semaphore, sema_init, sema_destroy, sema_wait, sema_trywait, sema_post – semaphores

SYNOPSIS
cc [ flag ...] file ... -lthread -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);

MT-LEVEL
MT-Safe
sema_post() is Async-Signal-Safe

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.

Default semaphore initialization (intra-process):

sema_t sp;
sema_init(&sp, NULL, NULL);
OR
sema_init(&sp, USYNC_THREAD, NULL);
OR

3T-1000 modified 30 Jun 1995
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 sema-
    phore 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 imple-
mentation. 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().
    sema_trywait() fails and returns the corresponding value if any of the following condi-
    tions are detected:

    EBUSY            The semaphore pointed to by sp has a zero count.

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)

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();
...  

sema_init(&tellers, TELLERS, USYNC_THREAD, NULL);  
/* 10 tellers available */  
while(banking_hours())  
    pthread_create(NULL, NULL, customer, deposit_withdrawal);  
...

void * 
customer(int deposit_withdrawal)  
{  
    int this_customer, in_a_hurry = 50;  
    this_customer = rand() % 100;  

    if (this_customer == in_a_hurry) {  
        if (sema_trywait(&tellers) != 0)  
            if (errno == EAGAIN) { /* no teller available */  
                skip_banking_today(this_customer);  
                return;  
            } /* else go immediately to available teller & 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 */  
}  

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.
NAME  send, sendto, sendmsg – send a message from a socket

SYNOPSIS  cc [ flag ... ] file ... -lsocket -lssl [ 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);

MT-LEVEL  Safe

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 under-
lying 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.
<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMSGSIZE</td>
<td>The socket requires that message be sent atomically, and the message was too long.</td>
</tr>
<tr>
<td>ENOMEM</td>
<td>There was insufficient memory available to complete the operation.</td>
</tr>
<tr>
<td>ENOSR</td>
<td>There were insufficient STREAMS resources available for the operation to complete.</td>
</tr>
<tr>
<td>ENOTSOCK</td>
<td>s is not a socket.</td>
</tr>
<tr>
<td>EWOULDBLOCK</td>
<td>The socket is marked non-blocking and the requested operation would block.</td>
</tr>
</tbody>
</table>

**SEE ALSO**

fcntl(2), poll(2), write(2), connect(3N), getsockopt(3N), recv(3N), select(3C), socket(3N)
NAME  setbuf, setvbuf – assign buffering to a stream

SYNOPSIS  #include <stdio.h>
            void setbuf(FILE *stream, char *buf);
            int setvbuf(FILE *stream, char *buf, int type, size_t size);

MT-LEVEL  MT-Safe

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.

            char 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.

SEE ALSO  fopen(3S), getc(3S), malloc(3C), putc(3S), stdio(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.

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(3) be used to handle buffer allocation when using setvbuf().

modified 22 Jan 1993
NAME  setbuffer, setlinebuf – assign buffering to a stream

SYNOPSIS  
#include <stdio.h>

void setbuffer(FILE *iop, char *abuf, size_t asize);
void setlinebuf(FILE *iop);

DESCRIPTION  setbuffer, setlinebuf – assign buffering to a stream

The three types of buffering available are unbuffered, block buffered, and line buffered. When an output stream is unbuffered, information appears on the destination file or terminal as soon as written; when it is block buffered many characters are saved up and written as a block; when it is line buffered characters are saved up until a NEWLINE is encountered or input is read from stdin. fflush(3S) may be used to force the block out early. Normally all files are block buffered. A buffer is obtained from malloc(3C) upon the first getc(3S) or putc(3S) on the file. If the standard stream stdout refers to a terminal it is line buffered. The standard stream stderr is unbuffered by default.

setbuffer() 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 big an array is needed:

char buf[BUFSIZ];

setlinebuf() is used to change the buffering on a stream from block buffered or unbuffered to line buffered. Unlike setbuffer(), it can be used at any time that the stream, 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 unbuffered by using freopen(3S) followed by setbuf(3S) with a buffer argument of NULL.

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.

3C-1006  modified 03 Mar 1995
NAME    setcat – define default catalog

SYNOPSIS  
#include <pfmt.h>
char *setcat(const char *catalog);

MT-LEVEL  MT-safe

DESCRIPTION  The routine setcat() defines the default message catalog to be used by subsequent calls to
pfmt(), pfmt() or gettxt() 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 argu-
ment will cancel the default catalog.

If no default catalog is specified, or if catalog is an invalid catalog name, Subsequent calls
to gettxt(), pfmt() or lfmt() that do not explicitely specify a catalog name will use Message not found!!

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");
gettxt("10", "hello world\n")

SEE ALSO  gettxt(3C), lfmt(3C), pfmt(3C), setlocale(3C), environ(5)
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, otherwise longjmp() will be returning control to a possibly non-existent environment. All memory-bound data have values as of the time longjmp() was called. The CPU and floating-point data registers are restored to the values they had at the time that setjmp() was called. But, because the register storage class is only a hint to the C compiler, variables declared as register variables may not necessarily be assigned to machine registers, so their values are unpredictable after a longjmp(). This is especially a problem for programmers trying to write machine-independent C routines.

setjmp() and longjmp() save and restore the signal mask while _setjmp() and _longjmp() manipulate only the C stack and registers.

None of these functions save or restore any floating-point status or control registers.
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 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 cc(1B), sigvec(3B), setjmp(3C), signal(3C)

NOTES Use of these interfaces should be restricted to only applications written on BSD platforms. Use of these interfaces with any of the system libraries or in multi-thread applications is unsupported.

BUGS `setjmp()` does not save the current notion of whether the process is executing on the signal stack. The result is that a `longjmp()` to some place on the signal stack leaves the signal stack state incorrect.

On some systems `setjmp()` also saves the register environment. Therefore, all data that are bound to registers are restored to the values they had at the time that `setjmp()` was called. All memory-bound data have values as of the time `longjmp()` was called. However, because the `register` storage class is only a hint to the C compiler, variables declared as `register` variables may not necessarily be assigned to machine registers, so their values are unpredictable after a `longjmp()`. When using compiler options that specify automatic register allocation (see `cc(1B)`), the compiler will not attempt to assign variables to registers in routines that call `setjmp()`.

`longjmp()` never causes `setjmp()` to return zero, so programmers should not depend on `longjmp()` being able to cause `setjmp()` to return zero.
NAME  setjmp, sigsetjmp, longjmp, siglongjmp – non-local goto

SYNOPSIS  
```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);
```

MT-LEVEL  Unsafe

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()`.

`sigtjmp()` saves the calling process’s registers and stack environment (see `sigaltstack`) in `env` for later use by `siglongjmp()`. If `savemask` is non-zero, the calling process’s signal mask (see `sigprocmask`) and scheduling parameters (see `priocntl`) 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 `sigtjmp()` with the corresponding `env` argument. After `siglongjmp()` is completed, program execution continues as if the corresponding call of `sigtjmp()` had just returned the value `val`. `siglongjmp()` cannot cause `sigtjmp()` to return the value 0. If `siglongjmp()` is invoked with a second argument of 0, `sigtjmp()` will return 1. At the time of the second return from `sigtjmp()`, 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 `sigtjmp()` 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`.

modified 22 Jan 1993 3C-1011
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);
    }
}
```

3C-1012 modified 22 Jan 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

**RETURN VALUES**

If `longjmp()` or `siglongjmp()` are invoked with a second argument of 0, `setjmp()` and `sigsetjmp()`, respectively, return 1. Otherwise, `setjmp()` and `sigsetjmp()` return 0.

**SEE ALSO**

`getcontext(2)`, `prioctnl(2)`, `sigaction(2)`, `sigaltstack(2)`, `sigprocmask(2)`, `signal(3C)`

**WARNINGS**

If `longjmp()` or `siglongjmp()` are called even though `env` was never primed by a call to `setjmp()` or `sigsetjmp()`, or when the last such call was in a function that has since returned, absolute chaos is guaranteed.
NAME    setlabel – define the label for pfmt() and lfmt().

MT-LEVEL MT-safe

SYNOPSIS #include <pfmt.h>
int setlabel(const char *label);

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.

SEE ALSO getopt(3C), lfmt(3C), pfmt(3C)
NAME
setlocale – modify and query a program’s locale

SYNOPSIS
#include <locale.h>
char *setlocale(int category, const char *locale);

MT-LEVEL
Safe with exceptions

DESCRIPTION
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() and
tolower(), and multibyte character functions such as mbtowc() and wctomb().

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(), ctime(), getdate(),
and strftime().

LC_COLLATE affects the sort order produced by strcoll() and strxfrm().

LC_MONETARY affects the monetary formatted information returned by localeconv().

LC_MESSAGES affects the behavior of dgettext(), gettext(), and gettxt().

Each category corresponds to a set of databases which contain the relevant information
for each defined locale. The location of a database is given by the following path,
/usr/lib/locale/locale/category, where locale and category are the names of locale and
category, respectively. For example, the database for the LC_CTYPE category of the de
(Deutsch or German) locale would be found in /usr/lib/locale/de/LC_CTYPE

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-
ables. 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>

modified 22 Jan 1993
If a pointer to a string is given for `locale`, `setlocale()` attempts to set the locale for the given category to `locale`. If `setlocale()` succeeds, `locale` is returned. If `setlocale()` fails, a null pointer is returned and the program’s locale is not changed.

For category `LC_ALL`, the behavior is slightly different. If a pointer to a string is given for `locale` and `LC_ALL` is given for `category`, `setlocale()` attempts to set the locale for all the categories to `locale`. The `locale` may be a simple locale, consisting of a single locale, or a composite locale. A composite locale is a string returned by a prior call to `setlocale(LC_ALL,0)`. This string will restore each category to the previous locale. If `setlocale()` fails to set the locale for any category, a null pointer is returned and the program’s locale for all categories is not changed. Otherwise, `locale` is returned.

A null pointer for `locale` causes `setlocale()` to return the current locale associated with the `category`. The program’s locale is not changed.

### FILES
- `/usr/lib/locale/` character type database for `locale`
- `/usr/lib/locale/` numeric format data for `locale`
- `/usr/lib/locale/` date and time formats for `locale`
- `/usr/lib/locale/` sort ordering information for `locale`
- `/usr/lib/locale/` message catalogs for `locale`
- `/usr/lib/locale/` currency format data for `locale`

### SEE ALSO
- `ctype(3C)`, `localeconv(3C)`, `mbchar(3C)`, `strcoll(3C)`, `strftime(3C)`, `gettext(3I)`, `environ(5)`

### NOTES
To change locale in a multi-thread application `setlocale` should be called prior to using any locale sensitive routine. Using `setlocale` to query the current locale is safe and can be used anywhere in a multi-thread application.

It is the user’s responsibility to ensure that mixed locale categories are compatible. For example, setting `LC_CTYPE=C` and `LC_TIME=ja` (where `ja` indicates Japanese) will not work, because Japanese time cannot be represented in the “C” locale’s ASCII codeset.
NAME  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);

MT-LEVEL  MT-Safe

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.

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)).

- 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 atomic with respect to other processes executing shm_open() naming the same shared memory object with OEXCL and O_CREAT set.
O_TRUNC If the shared memory object exists, and it is successfully opened
O_RDWR, the object is truncated to zero length and the mode and
ownership are unchanged by this function call.

RETURN VALUES If successful, shm_open() returns a nonnegative integer representing the lowest num-
bered unused file descriptor, otherwise it returns -1 and sets errno to indicate the error
condition.

ERRORS EACCESS 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.
EINTR The shm_open() operation was interrupted by a signal.
EINVAL name is an invalid file description.
ENFILE 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 shared memory object does not exist.
ENOSPC There is insufficient space for the creation of the new shared memory object.
ENOSYS shm_open() is not supported by this implementation.

FILES /usr/include/fcntl.h

SEE ALSO close(2), dup(2), exec(2), fcntl(2), mmap(2), umask(2), sysconf(3C), shm_unlink(3R),
fcntl(5)

NOTES When a shared memory object is created, the state of the shared memory object, includ-
ing all data associated with the shared memory object, persists until the shared memory
object is unlinked and all other references are gone.

BUGS In Solaris 2.5, these functions always return -1 and set errno to ENOSYS, because this
release does not support the Shared Memory Objects option. It is our intention to pro-
vide support for these interfaces in future releases.
NAME
shm_unlink – remove a shared memory object

SYNOPSIS
cc [ flag ...] file ... -lposix4 [ library ...]
int shm_unlink(const char *name);

MT-LEVEL
MT-Safe

DESCRIPTION
shm_unlink() removes the name of the shared memory object named by the string pointed to by 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 removal of the memory object contents will be postponed until all open and mapped references to the shared memory object have been removed.

RETURN VALUES
If successful, shm_unlink() returns 0, otherwise it returns -1 and sets errno to indicate the error condition, and the named shared memory object is not affected by this function.

ERRORS
EACCES Permission is denied to unlink the named shared memory object.
ENAMETOOLONG The length of the name string exceeds PATH_MAX, or a pathname component is longer than NAME_MAX while _POSIX_NO_TRUNC is in effect.
ENOENT The named shared memory object does not exist.
ENOSYS shm_unlink() is not supported by this implementation.

SEE ALSO
close(2), mmap(2), mlock(3C), shm_open(3R)

BUGS
In Solaris 2.5, these functions always return -1 and set errno to ENOSYS, because this release does not support the Shared Memory Objects option. It is our intention to provide support for these interfaces in future releases.

modified 12 Aug 1993
shutdown – shut down part of a full-duplex connection

cc [ flag ... ] file ... -lsocket -lnsl [ library ... ]

int shutdown(int s, int how);

Safe

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.

A 0 is returned if the call succeeds, −1 if it fails.

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.

**SEE ALSO** [connect(3N)](3N), [socket(3N)](3N)

**NOTES** The `how` values should be defined constants.
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 signals 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, variables modified on the occurrence of the signal are examined to determine that there is no work to be done, and the process pauses awaiting work by using sigpause( ) with the mask returned by sigblock( ).

It is not possible to block SIGKILL, SIGSTOP, or SIGCONT, this restriction is silently imposed by the system.

RETURN VALUES  
sigblock( ) and sigsetmask( ) return the previous set of masked signals. sigpause( ) returns −1 and sets errno to EINTR.

SEE ALSO  
kill(2), sigaction(2), signal(3B), sigvec(3B)

NOTES  
Use of these interfaces should be restricted to only applications written on BSD platforms. Use of these interfaces with any of the system libraries or in multi-thread applications is unsupported.

modified 19 Feb 1993
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);

MT-LEVEL

Safe

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
  FPEFLTINV    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.
EXAMPLES

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_overflow and fp_invalid handlers; set the new
     * fp_overflow handler to sample_handler() and set the new
     * fp_invalid handler to SIGFPE_ABORT (abort on invalid)
     */

    hdl = (sigfpe_handler_type) sample_handler;
    old_handler1 = sigfpe(FPE_FLTOVF, hdl);
    old_handler2 = sigfpe(FPE_FLTINV, SIGFPE_ABORT);
    ...
    ...
    /*
     * restore old fp_overflow and fp_invalid handlers
     */
```
```c
    sigfpe(FPE_FLTOVF, old_handler1);
    sigfpe(FPE_FLTINV, old_handler2);
}
```

**FILES**
/usr/include/ floatingpoint.h  
/usr/include/siginfo.h

**SEE ALSO**
sigaction(2), abort(3C), signal(3C), floatingpoint(5)

**DIAGNOSTICS**
sigfpe() returns BADSIG if code is not zero or a defined SIGFPE code.
<table>
<thead>
<tr>
<th>NAME</th>
<th>siginterrupt – allow signals to interrupt functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYNOPSIS</td>
<td>/usr/ucb/cc [ flag ... ] file ...</td>
</tr>
<tr>
<td></td>
<td>int siginterrupt( sig, flag)</td>
</tr>
<tr>
<td></td>
<td>int sig, flag;</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>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.</td>
</tr>
<tr>
<td>NOTES</td>
<td>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. This library routine uses an extension of the sigvec(3B) function that is not available in 4.2BSD, hence it should not be used if backward compatibility is needed.</td>
</tr>
<tr>
<td>RETURN VALUES</td>
<td>A 0 value indicates that the call succeeded. A −1 value indicates that the call failed and errno is set to indicate the error.</td>
</tr>
<tr>
<td>ERRORS</td>
<td>siginterrupt() may return the following error:</td>
</tr>
<tr>
<td></td>
<td>EINVAL   sig is not a valid signal.</td>
</tr>
<tr>
<td>SEE ALSO</td>
<td>sigblock(3B), sigvec(3B), signal(3C)</td>
</tr>
</tbody>
</table>

modified 19 Feb 1993 3B-1025
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 SIGHWR, 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 zero 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 SIG-KILL or SIGSTOP.

SEE ALSO kill(2), pause(2), sigaction(2), sigsend(2), wait(2), waitid(2), signal(5)

NOTES sighold() in conjunction with sigrelse() or sigpause() may be used to establish critical regions of code that require the delivery of a signal to be temporarily deferred.
If signal() or sigset() is used to set SIGCHLD’s disposition to a signal handler, SIGCHLD will not be sent when the calling process’s children are stopped or continued.
If any of the above functions are used to set SIGCHLD’s disposition to SIG_IGN, the calling process’s child processes will not create zombie processes when they terminate (see exit(2)). If the calling process subsequently waits for its children, it blocks until all of its children terminate; it then returns a value of −1 with errno set to ECHILD (see wait(2), waitid(2)).
NAME
sigqueue – queue a signal to a process

SYNOPSIS
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 */
};

MT-LEVEL
Async-Signal-Safe

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.

SEE ALSO
kill(2), sigwaitinfo(3R), siginfo(5), signal(5)
NAME  sigsetops, sigemptyset, sigfillset, sigaddset, sigdelset, sigismember – manipulate sets of signals

SYNOPSIS  

```c
#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);
```

MT-LEVEL  MT-Safe
          Async-Signal-Safe

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.

SEE ALSO  `sigaction(2)`, `sigpending(2)`, `sigprocmask(2)`, `sigsuspend(2)`, `signal(5)`

modified 22 Jan 1993
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  
sigstack() allows users to define an alternate stack, called the “signal stack”, on which signals are to be processed. When a signal’s action indicates its handler should execute on the signal stack (specified with a sigvec(3B) call), the system checks to see if the process is currently executing on that stack. If the process is not currently executing on the signal stack, the system arranges a switch to the signal stack for the duration of the signal handler’s execution.

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 */

ss_sp is the initial value to be assigned to the stack pointer when the system switches the process to the signal stack. Note that, on machines where the stack grows downwards in memory, this is not the address of the beginning of the signal stack area. ss_onstack field is zero or non-zero depending on whether the process is currently executing on the signal stack or not.

If nss is not a NULL pointer, sigstack() sets the signal stack state to the value in the sigstack() structure pointed to by nss. Note: if ss_onstack is non-zero, the system will think that the process is executing on the signal stack. 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, a value of 0 is returned. Otherwise, a value of −1 is returned and errno is set to indicate the error.

ERRORS  
sigstack() 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)

NOTES  
Use of these interfaces should be restricted to only applications written on BSD platforms. Use of these interfaces with any of the system libraries or in multi-thread applications is unsupported.
WARNINGS

Signal stacks are not “grown” automatically, as is done for the normal stack. If the stack overflows unpredictable results may occur.
NAME       sigvec — software signal facilities

SYNOPSIS   /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:

    void  (*sv_handler)();  /* signal handler */
    int   sv_mask;        /* signal mask to apply */
    int   sv_flags;       /* see signal options */
    #define SV_ONSTACK  /* take signal on signal stack */
    #define SV_INTERRUPT /* do not restart system on signal return */
    #define SV_RESETHAND /* reset handler to SIG_DFL when signal taken */
If the SV_ONSTACK bit is set in the flags for that signal, the system will deliver the signal to the process on the signal stack specified with sigstack(3B) rather than delivering the signal on the current stack.

If nvec is not a NULL pointer, sigvec() assigns the handler specified by sv_handler(), the mask specified by sv_mask(), and the flags specified by sv_flags() to the specified signal. If nvec is a NULL pointer, sigvec() does not change the handler, mask, or flags for the specified signal.

The mask specified in nvec is not allowed to block SIGKILL, SIGSTOP, or SIGCONT. The system enforces this restriction silently.

If ovec is not a NULL pointer, the handler, mask, and flags in effect for the signal before the call to sigvec() are returned to the user. A call to sigvec() with nvec a NULL pointer and ovec not a NULL pointer can be used to determine the handling information currently in effect for a signal without changing that information.

The following is a list of all signals with names as in the include file <signal.h>:

- SIGHUP  hangup
- SIGINT  interrupt
- SIGQUIT  quit
- SIGQUIT  * quit
- SIGILL  illegal instruction
- SIGILL  * illegal instruction
- SIGTRAP  trace trap
- SIGTRAP  * trace trap
- SIGABRT  abort (generated by abort(3C) routine)
- SIGABRT  * abort (generated by abort(3C) routine)
- SIGEMT  emulator trap
- SIGEMT  * emulator trap
- SIGFPE  arithmetic exception
- SIGFPE  * arithmetic exception
- SIGILL  kill (cannot be caught, blocked, or ignored)
- SIGILL  * kill (cannot be caught, blocked, or ignored)
- SIGBUS  bus error
- SIGBUS  * bus error
- SIGSEGV  segmentation violation
- SIGSEGV  * segmentation violation
- SIGSYS  bad argument to function
- SIGSYS  * bad argument to function
- SIGPIPE  write on a pipe or other socket with no one to read it
- SIGPIPE  * write on a pipe or other socket with no one to read it
- SIGALRM  alarm clock
- SIGALRM  software termination signal
- SIGURG  * urgent condition present on socket
- SIGURG  * urgent condition present on socket
- SIGSTOP  stop (cannot be caught, blocked, or ignored)
- SIGSTOP  † stop (cannot be caught, blocked, or ignored)
- SIGTSTP  stop signal generated from keyboard
- SIGTSTP  † stop signal generated from keyboard
- SIGCONT  continue after stop (cannot be blocked)
- SIGCONT  † continue after stop (cannot be blocked)
- SIGCHLD  child status has changed
- SIGCHLD  ● child status has changed
- SIGTIN  background read attempted from control terminal
- SIGTIN  † background read attempted from control terminal
- SIGTOU  background write attempted to control terminal
- SIGTOU  † background write attempted to control terminal
- SIGIO  I/O is possible on a descriptor (see fcntl(2))
- SIGIO  ● I/O is possible on a descriptor (see fcntl(2))
- SIGXCPU  cpu time limit exceeded (see getrlimit(2))
- SIGXCPU  ● cpu time limit exceeded (see getrlimit(2))
- SIGXFSZ  file size limit exceeded (see getrlimit(2))
- SIGXFSZ  ● file size limit exceeded (see getrlimit(2))
- SIGVTALRM  virtual time alarm (see setitimer() on getitimer(2))
- SIGVTALRM  ● virtual time alarm (see setitimer() on getitimer(2))
- SIGPROF  profiling timer 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-catching function, unless the signal is SIGILL, SIGPWR, or SIGTRAP. Also, if this bit is set, the bit for that signal in the signal mask will not be set; unless the signal mask associated with that signal blocks that signal, further occurrences of that signal will not be blocked. The SV_RESETHAND flag is not available in 4.2BSD, hence it should not be used if backward compatibility is needed.

The default action for a signal may be reinstated by setting the signal’s handler to SIG_DFL; this default is termination except for signals marked with • or *. Signals marked with • are discarded if the action is SIG_DFL; signals marked with * cause the process to stop. If the process is terminated, a “core image” will be made in the current working directory of the receiving process if the signal is one for which an asterisk appears in the above list (see core(4)).

If the handler for that signal is SIG_IGNORED, 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:

```c
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 */
};

MT-LEVEL Async-Safe

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 selection 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 structure 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 asynchronously 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
RETURN VALUES
If one of the signals specified by set is either pending or generated, sigwaitinfo() or sigtimedwait() returns the selected signal number. Otherwise, the function returns -1 and sets errno to indicate the error condition.

ERRORS
EINTR The wait was interrupted by an unblocked, caught signal.
ENOSYS sigwaitinfo() or sigtimedwait() is not supported by this implementation.

The following errors relate to only sigtimedwait():
EAGAIN No signal specified by set was delivered within the specified timeout period.
EINVAL timeout specified a tv_nsec value less than 0 or greater than 1,000,000,000.

SEE ALSO time(2), sigqueue(3R), siginfo(5), signal(5)
NAME  
sleep – suspend execution for interval

SYNOPSIS  
/usr/ucb/cc [ flag ... ] file ...

int sleep(seconds)
unsigned seconds;

MT-LEVEL  
Async-Signal-Safe

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 previous state of this timer is saved and restored. If the sleep time exceeds the time to the expiration of the previous value of the timer, the process sleeps only until the timer would have expired, and the signal which occurs with the expiration of the timer is sent one second later.

SEE ALSO  
alarm(2), getitimer(2), longjmp(3C), siglongjmp(3C), sleep(3C), usleep(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.

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);

MT-LEVEL  Safe

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().

SEE ALSO  alarm(2), pause(2), signal(3C)

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 22 Jan 1993
NAME  socket – create an endpoint for communication

SYNOPSIS  cc [ flag ... ] file ... -lsocket -lnsl [ library ... ]

#include <sys/types.h>
#include <sys/socket.h>

int socket(int domain, int type, int protocol);

MT-LEVEL  Safe

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 fam-
ily generally is the same as the address family for the addresses supplied in later opera-
tions 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, pro-
tocol 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  SOCK_DGRAM  SOCK_RAW  SOCK_SEQPACKET  SOCK_RDM

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. How-
ever, 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.

modified 30 Mar 1993
socket (3N)  Network Functions  SunOS 5.5

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENOSR</td>
<td>There were insufficient STREAMS resources available to complete the operation.</td>
</tr>
<tr>
<td>EPROTONOSUPPORT</td>
<td>The protocol type or the specified protocol is not supported within this domain.</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), recvfrom(3N), setsockopt(3N),
- send(3N), shutdown(3N), socketpair(3N)
NAME     socketpair – create a pair of connected sockets

SYNOPSIS  cc [ flag ... ] file ... -lssocket -lnsl [ library ... ]
           \#include <sys/types.h>
           \#include <sys/socket.h>
           int socketpair(int domain, int type, int protocol, int sv[2]);

MT-LEVEL Safe

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.

SEE ALSO pipe(2), read(2), write(2)

NOTES This call is currently implemented only for the AF_UNIX address family.
NAME     spray — scatter data in order to test the network

SYNOPSIS  cc [ flag  ... ] file  ...  -lspray  -lnsl [ library  ... ]

#include <rpcsvc/spray.h>
bool_t xdr_sprayarr(XDR *xdrs, sprayarr *objp);
bool_t xdr_spraycumul(XDR *xdrs, spraycumul *objp);

MT-LEVEL  Unsafe

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:
#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) {

3N-1046 modified 2 Jun 1994
if (clnt_call(clnt, SPRAYPROC_CLEAR,
   xdr_void, NULL, xdr_void, NULL, timeout25)) {
   /* handle this error */
}

while (loop-- > 0) {
   if (clnt_call(clnt, SPRAYPROC_SPRAY,
      xdr_sprayarr, &spray_data, xdr_void, NULL, timeout0)) {
      /* handle this error */
   }
}

if (clnt_call(clnt, SPRAYPROC_GET,
   xdr_void, NULL, xdr_spraycumul, &spray_result, timeout25)) {
    /* handle this error */
}

printf("Acknowledged %ld of 1000 packets in %d secs %d usecs\n",
   spray_result.counter,
   spray_result.clock.sec,
   spray_result.clock.usec);

SEE ALSO spray(1M), rpc_clnt_calls(3N)

NOTES This interface is unsafe in multithreaded applications. Unsafe interfaces should be called only from the main thread.
A spray program is not useful as a networking benchmark as it uses unreliable connectionless transports, (udp for example). It can report a large number of packets dropped when the drops were caused by the program sending packets faster than they can be buffered locally (before the packets get to the network medium).
NAME  
sqrt, cbrt – square root, cube root

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

MT-LEVEL  
MT-Safe

DESCRIPTION  
sqrt(x) returns the square root of x, correctly rounded according to ANSI/IEEE 754-1985.
cbrt(x) returns the cube root of x.  cbrt() is accurate to within 0.7 ulps.

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

SEE ALSO  
matherr(3M)
NAME

ssignal, gsignal – software signals

SYNOPSIS

```
#include <signal.h>

void (*ssignal (int sig, int (*action) (int))) (int);
int gsignal(int sig);
```

MT-LEVEL

Unsafe

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.

SEE ALSO

raise(3C), signal(3C)
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(), fwrite(), 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 associ-
ated 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 asso-
ciated 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 imple-
mentation.

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 imple-
mentation guarantees can be open simultaneously. Note that no more than 255 files may
be opened using fopen(), and only file descriptors 0 through 255 can be used in a stream.

The functions and constants mentioned in the entries of section 3S of this manual are
declared in that header and need no further declaration. The constants and the following
“functions” are implemented as macros (redeclaration of these names is perilous): getc(),
getchar(), putc(), putchar(), ferror(), 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()`
function, two handles exist where one existed before. The application shall assure that, if
both handles will ever be accessed, that they will both be in a state where the other could
become the active handle first. The application shall prepare for a `fork()` exactly as if it
were a change of active handle. (If the only action performed by one of the processes is
one of the `exec` functions or `_exit()`, the handle is never accessed in that process.)

(1) For the first handle, the first applicable condition below shall apply. After the
actions required below are taken, the handle may be closed if it is still open.

(a) If it is a file descriptor, no action is required.

(b) If the only further action to be performed on any handle to this open file
description is to close it, no action need be taken.

(c) If it is a stream that is unbuffered, no action need be taken.

(d) If it is a stream that is line-buffered and the last character written to the
stream was a newline (that is, as if a `putc('n')` was the most recent
operation on that stream), no action need be taken.

(e) If it is a stream that is open for writing or append (but not also open for
reading), either an `fflush()` shall occur or the stream shall be closed.

(f) If the stream is open for reading and it is at the end of the file (`feof()` is
true), no action need be taken.

(g) If the stream is open with a mode that allows reading and the underlying
open file description refers to a device that is capable of seeking, either
an `fflush()` shall occur or the stream shall be closed.

(h) Otherwise, the result is undefined.

(2) For the second handle: if any previous active handle has called a function that
explicitly changed the file offset, except as required above for the first handle, the
application shall perform an `lseek()` or an `fseek()` (as appropriate to the type of
the handle) to an appropriate location.

(3) If the active handle ceases to be accessible before the requirements on the first
handle above have been met, the state of the open file description becomes
undefined. This might occur, for example, during a `fork()` or an `_exit()`.

(4) The `exec` functions shall be considered to make inaccessible all streams that are
open at the time they are called, independent of what streams or file descriptors
may be available to the new process image.

(5) Implementation shall assure that an application, even one consisting of several
processes, shall yield correct results (no data is lost or duplicated when writing,
all data is written in order, except as requested by seeks) when the rules above
are followed, regardless of the sequence of handles used. If the rules above are
not followed, the result is unspecified. When these rules are followed, it is imple-
mentation defined whether, and under what conditions, all input is seen exactly
once.
Use of stdio in Multithreaded Applications

All the stdio functions are safe unless they have the _unlocked suffix. Each file pointer has its own lock to guarantee that only one thread can access it. In the case that output needs to be synchronized, the lock for the FILE pointer can be acquired before performing a series of stdio operations. For example:

FILE iop;
.
.

clockfile(iop);
fprintf(iop, "hello ");
fprintf(iop, "world ");
putc(iop, 'a');
unlockfile(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:

clockfile(iop);
while (!feof(iop)) {
    *c++ = getc_unlocked(iop);
}
unlockfile(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(2), fseek(3S), flockfile(3S), getc(3S), gets(3S), popen(3S), printf(3S), putchar(3S), puts(3S), scanf(3S), setbuf(3S), system(3S), tmpfile(3S), tmpnam(3S), ungetc(3S)
stdipc (3C)  C Library Functions  SunOS 5.5

NAME  stdipc, ftok – standard interprocess communication package

SYNOPSIS  

#include <sys/types.h>
#include <sys/ipc.h>

key_t ftok(const char *path, int id);

MT-LEVEL  See the NOTES section of this page.

DESCRIPTION  Certain interprocess communication facilities require the user to supply a key to be used by the msgget(2), semget(2), and shmget(2) functions to obtain interprocess communication identifiers. One suggested method for forming a key is to use the ftok() subroutine described below. Another way to compose keys is to include the project ID in the most significant byte and to use the remaining portion as a sequence number. There are many other ways to form keys, but it is necessary for each system to define standards for forming them. If some standard is not adhered to, it will be possible for unrelated processes to unintentionally interfere with each other's operation. It is still possible to interfere intentionally. Therefore, it is strongly suggested that the most significant byte of a key in some sense refer to a project so that keys do not conflict across a given system.

ftok() returns a key based on path and id that is usable in subsequent msgget(), semget(), and shmget() functions. path must be the path name of an existing file that is accessible to the process. id is a character that uniquely identifies a project. Note that ftok() returns the same key for linked files when called with the same id and that it returns different keys when called with the same file name but different ids.

RETURN VALUES  ftok() returns (key_t) −1 if path does not exist or if it is not accessible to the process.

SEE ALSO  intro(2), msgget(2), semget(2), shmget(2)

NOTES  If the file whose path is passed to ftok() is removed when keys still refer to the file, future calls to ftok() with the same path and id return an error. If the same file is recreated, then ftok() is likely to return a different key from the original call.

ftok() is MT-Safe in mutli-thread applications.

3C-1054  modified 13 Jul 1990
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 3C-1055
NAME    strccpy, streadd, strcadd, strcpy – copy strings, compressing or expanding escape codes

SYNOPSIS    cc [ flag ... ] file ... -lgen [ library ... ]
#include <libgen.h>
char *strccpy(char *output, const char *input);
char *streadd(char *output, const char *input);
char *strcadd(char *output, const char *input);
char *strecpy(char *output, const char *input, const char *exceptions);
char *strecpy(char *output, const char *input, const char *exceptions);

MT-LEVEL    MT-Safe

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 strocpy(), except that it returns the pointer to the null byte that
terminates the output.

EXAMPLES    /* expand all but newline and tab */
    strocpy(output, input, "\n\t");

    /* concatenate and compress several strings */
    cp = strcadd(output, input1);
    cp = streadd(cp, input2);
    cp = streadd(cp, input3);

SEE ALSO    string(3C), strfind(3G)

NOTES    When compiling multi-thread applications, the _REENTRANT flag must be defined on
the compile line. This flag should only be used in multi-thread applications.
NAME  
strcoll – string collation

SYNOPSIS  
#include <string.h>

int strcoll(const char *s1, const char *s2);

MT-LEVEL  
Safe with exceptions

DESCRIPTION  
strcoll() returns an integer greater than, equal to, or less than zero in direct correlation to whether string 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)).

Both strcoll() and strxfrm() provide for locale-specific string sorting. strcoll() is intended for applications in which the number of comparisons per string is small. When strings are to be compared a number of times, strxfrm() is a more appropriate function because the transformation process occurs only once.

FILES  
/usr/lib/locale/LC_COLLATE    LC_COLLATE database for locale

SEE ALSO  
colltbl(1M), setlocale(3C), string(3C), strxfrm(3C), wsxfrm(3I), environ(5)

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

modified 22 Jan 1993
NAME  strerror – get error message string

SYNOPSIS
    #include <string.h>
    char *strerror(int errnum);

MT-LEVEL   Safe

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.

SEE ALSO  gettext(3I), perror(3C), setlocale(3C)

NOTES  If the application is linked with -lintl, then messages returned from this function are in the native language specified by the LC_MESSAGES locale category; see setlocale(3C).
NAME  strfind, 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);

MT-LEVEL  MT-Safe

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";
a2[] = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
s2 = strtrns(s1, a1, a2, s2);

SEE ALSO  string(3C)

NOTES  When compiling multi-thread applications, the _REENTRANT flag must be defined on the compile line. This flag should only be used in multi-thread applications.

modified 20 Mar 1994  3G-1059
NAME    strfmon – convert monetary value to string

SYNOPSIS  
#include <monetary.h>  
ssize_t strfmon(char *s, size_t maxsize, const char *format, ...);

MT-LEVEL  MT-Safe

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 character. This flag does not affect field width filling which always uses the space character. This flag is ignored unless a left precision (see below) is specified.

=+  Do not format the currency amount with grouping characters. The default is to insert the grouping characters if defined for the current locale.

=!  Suppress the currency symbol from the output conversion.

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.

Specify the alignment. If this flag is present all fields are left-justified (padded to the right) rather than right-justified.
**Field Width**  
\( w \)  
A decimal digit string \( w \) 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**  
The conversion characters and their meanings are:

<table>
<thead>
<tr>
<th>Character</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</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>n</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>%</td>
<td>Convert to a %; no argument is converted. The entire conversion specification must be %%.</td>
</tr>
</tbody>
</table>

**Locale Information**  
The `LC_MONETARY` category of the program’s locale affects the behaviour of this function including the monetary radix character (which may be different from the numeric radix character affected by the `LC_NUMERIC` category), the grouping separator, the currency symbols and formats. The international currency symbol should be conformant with the ISO 4217: 1987 standard.
RETURN VALUES
If the total number of resulting bytes (including the terminating null byte) is not more than \texttt{maxsize}, \texttt{strfmon()} returns the number of bytes placed into the array pointed to by \texttt{s}, not including the terminating null byte. Otherwise, \texttt{-1} is returned, the contents of the array are indeterminate, and \texttt{errno} is set to indicate the error.

ERRORS
\texttt{strfmon()} will fail if:

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{ENOSYS}</td>
<td>The function is not supported.</td>
</tr>
<tr>
<td>\texttt{E2BIG}</td>
<td>Conversion stopped due to lack of space in the buffer.</td>
</tr>
</tbody>
</table>

EXAMPLES
Given a locale for the U.S.A. and the values 123.45, \texttt{-123.45}, and \texttt{3456.781}:

<table>
<thead>
<tr>
<th>Conversion Specification</th>
<th>Output</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{%n} \texttt{$123.45}</td>
<td>\texttt{-123.45} \texttt{$3,456.78}</td>
<td>default formatting</td>
</tr>
<tr>
<td>\texttt{%11n} \texttt{$123.45}</td>
<td>\texttt{-123.45} \texttt{$3,456.78}</td>
<td>right align within an 11 character field</td>
</tr>
<tr>
<td>\texttt{%5n} \texttt{$123.45}</td>
<td>\texttt{-123.45} \texttt{$3,456.78}</td>
<td>aligned columns for values up to 99,999</td>
</tr>
<tr>
<td>\texttt{%=*5n} \texttt{$***123.45}</td>
<td>\texttt{-***123.45} \texttt{$+3,456.78}</td>
<td>specify a fill character</td>
</tr>
<tr>
<td>\texttt{%=0#5n} \texttt{$000123.45}</td>
<td>\texttt{-$000123.45} \texttt{$03,456.78}</td>
<td>fill characters do not use grouping even if the fill character is a digit</td>
</tr>
<tr>
<td>\texttt{%=*5n} \texttt{$123.45}</td>
<td>\texttt{-123.45} \texttt{$3456.78}</td>
<td>disable the grouping separator</td>
</tr>
<tr>
<td>\texttt{%*5.0n} \texttt{$123}</td>
<td>\texttt{-123} \texttt{$3457}</td>
<td>round off to whole units</td>
</tr>
<tr>
<td>\texttt{%*5.4n} \texttt{$123.4500}</td>
<td>\texttt{-123.4500} \texttt{$3456.7810}</td>
<td>increase the precision</td>
</tr>
<tr>
<td>\texttt{%(#5n} \texttt{123.45 ($123.45)} \texttt{3,456.78}</td>
<td>\texttt{use an alternative pos/neg style}</td>
<td></td>
</tr>
<tr>
<td>\texttt{%(#!5n} \texttt{123.45 (123.45)} \texttt{3,456.78}</td>
<td>\texttt{disable the currency symbol}</td>
<td></td>
</tr>
</tbody>
</table>
SEE ALSO
localeconv(3C)

NOTES
This interface is expected to be mandatory in a future issue of this document. Lower-case conversion characters are reserved for future use and upper-case for implementation-dependent use.

modified 13 May 1994
NAME  strftime, cftime, ascftime – 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( ), ascftime( ), 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 *)0, then the locale’s default format is used. For strftime( ) the default format is the same as %c; for cftime( ) and ascftime( ) the default format is the same as %C. cftime( ) and ascftime( ) first try to use the value of the environment variable CFTIME, and if that is undefined or empty, the default format is used.

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 ascftime( ), 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
- %D  date as %m/%d/%y
- %e  day of month [1,31]; single digits are preceded by a space
- %h  locale’s abbreviated month name
- %H  hour (24-hour clock) [0,23]; single digits are preceded by 0
- %I  hour (12-hour clock) [1,12]; single digits are preceded by 0
- %j  day number of year [1,366]; single digits are preceded by 0
- %k  hour (24-hour clock) [0,23]; single digits are preceded by a blank

MT-LEVEL  MT-Safe
%I  hour (12-hour clock) [1,12]; single digits are preceded by a blank
%m  month number [1,12]; single digits are preceded by 0
%M  minute [00,59]; leading 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

modified 17 Jul 1995
%EY full alternate year representation
%Od day of the month using the locale’s alternate numeric symbols
%OE same as %Od
%OH hour (24-hour clock) using the locale’s alternate numeric symbols
%OI hour (12-hour clock) using the locale’s alternate numeric symbols
%Om month using the locale’s alternate numeric symbols
%OM minutes using the locale’s alternate numeric symbols
%OS seconds using the locale’s alternate numeric symbols
%OU week number of the year (Sunday as the first day of the week) using the locale’s alternate numeric symbols
%OW 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(). 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".

FILES
/usr/lib/locale/locale/LC_TIME/time locale specific date and time information

SEE ALSO
date(1), ctime(3C), mktime(3C), setlocale(3C), strftime(3C), tzset(3C), TIMEZONE(4), strftime(4), environ(5), xpg4(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, strncat, strchr, strrchr, strcpy, strncpy, strspn, strspn, strdup, strlen, strpbrk, strstr, strtok, strtok_r – string operations

SYNOPSIS  

#include <string.h>

int strcasecmp(const char *s1, const char *s2);
int strncasecmp(const char *s1, const char *s2, int n);
char *strcat(char *dst, const char *src);
char *strncat(char *dst, const char *src, size_t n);
char *strchr(const char *s, int c);
char *strrchr(const char *s, int c);
int strcmp(const char *s1, const char *s2);
int strncmp(const char *s1, const char *s2, size_t n);
char *strcpy(char *dst, const char *src);
char *strncpy(char *dst, const char *src, size_t n);
size_t strcspn(const char *s1, const char *s2);
size_t strspn(const char *s1, const char *s2);
char *strdup(const char *s1);
size_t strlen(const char *s);
char *strpbrk(const char *s1, const char *s2);
char *strstr(const char *s1, const char *s2);
char *strtok(char *s1, const char *s2);
char *strtok_r(char *s1, const char *s2, char **lasts);

MT-LEVEL  See the NOTES section of this page.

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 character 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.

modified 3 May 1994  3C-1067
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. strchr() 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.

strstr() makes the same comparison but looks at a maximum of n bytes. Bytes following a null byte are not compared.

strcpy() copies string src to dst including the terminating null character, stopping after the null character has been copied. strncpy() copies exactly n bytes, truncating src or adding null characters to dst if necessary. The result will not be null-terminated if the length of src is n or more. Each function returns dst.

strcspn() returns the length of the initial segment of string s1 that consists entirely of characters not from string s2. strspn() returns the length of the initial segment of string s1 that consists entirely of characters from string s2.

strdup() returns a pointer to a new string that is a duplicate of the string pointed to by s1. The space for the new string is obtained using malloc(3C). If the new string cannot be created, a null pointer is returned.

strlen() returns the number of bytes in s, not including the terminating null character.

strpbrk() returns a pointer to the first occurrence in string s1 of any character from string s2, or a null pointer if no character from s2 exists in s1.

strstr() locates the first occurrence of the string s2 (excluding the terminating null character) in string s1. strstr() returns a pointer to the located string, or a null pointer if the string is not found. If s2 points to a string with zero length (that is, the string "") the function returns s1.

strtok() can be used to break the string pointed to by s1 into a sequence of tokens, each of which is delimited by one or more characters from the string pointed to by s2. strtok() considers the string s1 to consist of a sequence of zero or more text tokens separated by spans of one or more characters from the separator string s2. The first call (with pointer s1 specified) returns a pointer to the first character of the first token, and will have written a null character into s1 immediately following the returned token. The function keeps track of its position in the string between separate calls, so that subsequent calls (which must be made with the first argument being a null pointer) will work through the string s1 immediately following that token. In this way subsequent calls will work through the string s1 until no tokens remain. The separator string s2 may be different from call to call. When no token remains in s1, a null pointer is returned.

strtok_r() has the same functionality as strtok() except that a pointer to a string placeholder lasts must be supplied by the caller. The lasts pointer is to keep track of the next substring in which to search for the next token.
SEE ALSO malloc(3C), setlocale(3C), strxfrm(3C)

NOTES

The **strtok_r()** interface is as proposed in the POSIX.4a Draft #6 document, and is subject to change to be compliant to the standard when it is accepted.

When compiling multi-thread applications, the `_REENTRANT` flag must be defined on the compile line. This flag should only be used in multi-thread applications.

All of these functions assume the default locale “C.” For some locales, **strxfrm()** should be applied to the strings before they are passed to the functions.

**strtok()** is unsafe in multi-thread applications. **strtok_r()** should be used instead.

string(), strcasecmp(), strcat(), strchr(), strcmp(), strcpy(), strcspn(), strdup(), strlen(), strncasecmp(), strncat(), strncmp(), strncpy(), strpbrk(), strrchr(), strspn(), and **strstr()**, are MT-Safe in multi-thread applications.
NAME    string_to_decimal, file_to_decimal, func_to_decimal – parse characters into decimal record

SYNOPSIS #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);

MT-LEVEL MT-Safe

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 'Y' 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 (dotfrac_form)
    digits.digits (intdotfrac_form)

efield consists of

    echar digits

3-1070  modified 13 Jul 1992
or

\[ echar \text{ sign digits} \]

where \( echar \) is one of [Ee], and \( digits \) contains one or more digits.

When \( \text{fortran_conventions} \) 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 \( \text{fortran_conventions} \) is nonzero, \( echar \) may also be one of [DdQq], and \( efield \) may also have the form

\[ \text{sign digits} \]

When \( \text{fortran_conventions} \geq 2 \), blanks may appear in the \( digits \) strings for the integer, fraction, and exponent fields and may appear between \( echar \) and the exponent sign and after the infinity and NaN forms. If \( \text{fortran_conventions} \geq 2 \), the blanks are ignored. When \( \text{fortran_conventions} \geq 3 \), the blanks that appear in \( digits \) strings are interpreted as zeros, and other blanks are ignored.

When \( \text{fortran_conventions} \) is zero, the current locale’s decimal point character is used as the decimal point; when \( \text{fortran_conventions} \) is nonzero, the period is used as the decimal point.

The form of the accepted decimal string is placed in \( *\text{peform} \). If an \( efield \) is recognized, \( *\text{pechar} \) is set to point to the \( echar \).

On input, \( *\text{pc} \) points to the beginning of a character string buffer of length \( \geq nmax \). On output, \( *\text{pc} \) points to a character in that buffer, one past the last accepted character.

\( \text{string_to_decimal()} \) gets its characters from the buffer; \( \text{file_to_decimal()} \) gets its characters from \( *\text{pf} \) and records them in the buffer, and places a null after the last character read. \( \text{func_to_decimal()} \) gets its characters from an int function \( (*\text{pget})() \).

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.

\( \text{file_to_decimal()} \) and \( \text{func_to_decimal()} \) set \( *\text{pnread} \) to the number of characters read from the file; if greater than \( nmax \), some characters were lost. If no characters were lost, \( \text{file_to_decimal()} \) and \( \text{func_to_decimal()} \) attempt to push back, with ungetc(3S) or \( (*\text{punget})() \), as many as possible of the excess characters read, adjusting \( *\text{pnread} \) accordingly. If all unget calls are successful, then \( **\text{pc} \) will be NULL. No push back will be attempted if \( (*\text{punget})() \) is NULL.

modified 13 Jul 1992
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().

SEE ALSO  ctype(3C), localeconv(3C), scanf(3S), setlocale(3C), strtod(3C), ungetc(3S)
NAME
strptime – date and time conversion

SYNOPSIS
#include <time.h>
char *strptime(const char *buf, const char *format, struct tm *tm);

MT-LEVEL
MT-Safe

DESCRIPTION
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:

- %
- %a
day of week, using the locale’s weekday names; either the abbreviated or full name may be specified
- %A
- %b
month, using the locale’s month names; either the abbreviated or full name may be specified
- %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
date as %d
- %h
- %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 zero is 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
- %S
seconds [0,61]; leading zero is permitted but not required
- %t
any white space
- %T
time as %H:%M:%S
- %U
week number of the year as a decimal number [0,53], with Sunday as the first day of the week; leading zeros are permitted but not required
- %w
weekday as a decimal number [0,6], with 0 representing Sunday;
%W  week number of the year as a decimal number [0,53], with Monday as the first
day of the week; leading zero is permitted but not required
%x  locale’s appropriate date representation
%X  locale’s appropriate time representation
%y  year within the century [0,99]; leading zero is permitted but not required
%Y  year, including the century (for example, 1993)
%Z  timezone name or no characters if no time zone information exists

Modified Conversion Specifications
Some conversion specifications can be modified by the E and O modifier characters to
indicate that an alternate format or specification should be used rather than the one nor-
mally used by the unmodified specification. If the alternate format or specification does
not exist in the current locale, the behaviour will be as if the unmodified conversion
specification were used.
%Ec  locale’s alternate appropriate date and time representation
%EC  name of the base year (era) in the locale’s alternate representation
%Ex  locale’s alternate date representation
%EX  locale’s alternate time representation
%Ey  offset from %EC (year only) in the locale’s alternate representation
%EY  full alternate year representation
%Od  day of the month using the locale’s alternate numeric symbols
%Oe  same as %Od
%OH  hour (24-hour clock) using the locale’s alternate numeric symbols
%Oo  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
%Ow  week number of the year (Sunday as the first day of the week) 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 combina-
tion 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 \textit{tm} structure members are set to values corresponding to the locale information. If no match is found, \texttt{strptime()} fails and no more characters are scanned.

The month names, weekday names, era names, and alternate numeric symbols can consist of any combination of upper and lower case letters. The user can request that the input date or time specification be in a specific language by setting the \texttt{LC\_TIME} category using \texttt{setlocale(3C)}.

**RETURN VALUES**

Upon successful completion, \texttt{strptime()} returns a pointer to the character following the last character parsed. Otherwise, a null pointer is returned.

**FILES**

/\texttt{usr/lib/locale/LC\_TIME/time}\hfill /\texttt{usr/lib/locale/LC\_CTYPE/ctype}

locale specific date and time information  
character characterization information

**SEE ALSO** \texttt{isspace(3C), getdate(3C), setlocale(3C), strftime(3C)}

**NOTES**

Several “same as” formats, and the special processing of white-space characters are provided in order to ease the use of identical \textit{format} strings for \texttt{strftime()} and \texttt{strptime(\texttt{)}}. The range of values for \texttt{\%S} is [00,61] rather than [00,59] to allow for the occasional leap second and even more occasional double leap second.
NAME       strsignal – get error message string

SYNOPSIS   #include <string.h>
            char *strsignal(int sig);

MT-LEVEL   Safe

DESCRIPTION strsignal() maps the signal number in 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.

SEE ALSO   gettext(3I), psignal(3C), setlocale(3C), str2sig(3C)

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

strtod, atof – convert string to double-precision number

SYNOPSIS

#include <stdlib.h>

double strtod(const char *nptr, char **endptr);
double atof(const char *nptr);

MT-LEVEL

MT-Safe

DESCRIPTION

strtod() returns as a double-precision floating-point number the value represented by the character string pointed to by nptr. The string is scanned up to the first unrecognized character.

strtod() recognizes an optional string of “white-space” characters (as defined by isspace() in ctype(3C)), then an optional sign, then a string of digits optionally containing a decimal point character, then an optional exponent part including e or E followed by an optional sign, followed by an integer. The decimal point character is defined by the program’s locale (category LC_NUMERIC). In the "C" locale, or in a locale where the decimal point character is not defined, the decimal point character defaults to a period (.).

If the value of endptr is not (char **)NULL, a pointer to the character terminating the scan is returned in the location pointed to by endptr. If no number can be formed, *endptr is set to nptr, and zero is returned.

atof(nptr) is equivalent to:

    strtod(nptr, (char **)NULL).

LC_NUMERIC

Determines how strtod and atof handle numeric formats. In the "C" locale, numeric handling follows the U.S. rules.

RETURN VALUES

If the correct value would cause overflow, ±HUGE 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.

When the −Xc or −Xa compilation options are used, HUGE_VAL is returned instead of HUGE.

If nptr is NaN, then atof() returns NaN.

FILES

/usr/lib/locale/locale/LC_NUMERIC/numeric

   LC_NUMERIC database for locale

SEE ALSO

cctype(3C), matherr(3M), scanf(3S), strtol(3C), math(5)

modified 20 Feb 1994

3C-1077
NAME
strtol, strtoll, strtoul, strtoull, atol, atoll, atoi, lltostr, ulltostr – conversion routines

SYNOPSIS
#include <stdlib.h>

long strtol(const char *str, char **ptr, int base);
long long strtoll(const char *str, char **ptr, int base);
unsigned long strtoul(const char *str, char **ptr, int base);
unsigned long long strtoull(const char *str, char **ptr, int base);
long atol(const char *str);
long long atoll(const char *str);
int atoi(const char *str);
char *lltostr(long long value, char *ptr);
char *ulltostr(unsigned long long value, char *ptr);

MT-LEVEL
MT-Safe

DESCRIPTION
strtol() returns as a long integer the value represented by the character string pointed to
by str. The string is scanned up to the first character inconsistent with base. Leading
“white-space” characters (as defined by isspace() in ctype(3C)) are ignored.

strtol() is similar to strtol() except that the value is returned as a long long.

If the value of ptr is not (char **)NULL, a pointer to the character terminating the scan is
returned in the location pointed to by ptr. If no integer can be formed, that location is set
to str, and zero is returned.

If base is positive (and not greater than 36), it is used as the base for conversion. After an
optional leading sign, leading zeros are ignored, and “0x” or “0X” is ignored if base is 16.

If base is zero, the string itself determines the base as follows: After an optional leading
sign a leading zero indicates octal conversion, and a leading “0x” or “0X” hexadecimal
conversion. Otherwise, decimal conversion is used.

Truncation from long to int can, of course, take place upon assignment or by an explicit
cast.

If the value represented by str would cause overflow, LONG_MAX or LONG_MIN is
returned (according to the sign of the value), and errno is set to the value, ERANGE.

strtoul() and strtoull() are similar to strtol() except that strtoul() returns the value
represented by str as an unsigned long integer and strtoull() returns this value as an
unsigned long long. If the value represented by str would cause overflow,
ULONG_MAX is returned, and errno is set to the value, ERANGE.

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.

`ulltostr()` is similar to `lltostr()` except that `value` is an `unsigned long long`.

**RETURN VALUES**

If `strtol()` is given a `base` greater than 36, it returns 0 and sets `errno` to `EINVAL`.

**SEE ALSO**

`ctype(3C), scanf(3S), strtod(3C)`

**NOTES**

`strtol()` no longer accepts values greater than `LONG_MAX` as valid input. Use `strtol()` instead.
### NAME
strxfrm – string transformation

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

size_t strxfrm(char *dst, const char *src, size_t n);
```

### MT-LEVEL
Safe with exceptions

### DESCRIPTION
strxfrm() transforms the string src and places the resulting string into the array dst. If strcmp() is applied to two transformed strings, it will return the same result as strcoll() applied to the same two original strings. The transformation is based on the program's locale for category LC_COLLATE (see setlocale(3C)).

No more than n bytes will be placed into the resulting array pointed to by dst, including the terminating null character. If n is 0 and dst is a NULL parameter, strxfrm() returns the number of bytes required for the transformed string. If copying takes place between objects that overlap, the behavior is undefined.

strxfrm() returns the length of the transformed string (not including the terminating null character). If the value returned is n or more, the contents of the array dst are indeterminate.

### RETURN VALUES
On failure, strxfrm() returns (size_t)−1.

### EXAMPLES
The value of the following expression is the size of the array needed to hold the transformation of the string pointed to by s.

\[
1 + \text{strxfrm}(	ext{NULL}, s, 0);
\]

### FILES
/usr/lib/locale/LC_COLLATE LC_COLLATE database for locale

### SEE ALSO
colltbl(1M), setlocale(3C), strcoll(3C), string(3C), wscoll(3I), environ(5)

### NOTES
strxfrm can be used safely in a multi-thread application, as long as setlocale(3C) is not being called to change the locale.
NAME  swab – swap bytes

SYNOPSIS  
#include <stdlib.h>
void swab(const char *from, char *to, int nbytes);

MT-LEVEL  MT-Safe

DESCRIPTION  swab() copies nbytes bytes pointed to by from to the array pointed to by to, exchanging adjacent even and odd bytes. nbytes should be even and non-negative. If nbytes is odd and positive, swab() uses nbytes–1 instead. If nbytes is negative, swab() does nothing.
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.
NAME
sysconf – get configurable system variables

SYNOPSIS
#include <unistd.h>
long sysconf(int name);

MT-LEVEL
MT-Safe, Async-Signal-Safe

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.

<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_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_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>System memory page size.</td>
</tr>
<tr>
<td>_SC_PAGESIZE</td>
<td>PAGESIZE</td>
<td>Job control supported?</td>
</tr>
<tr>
<td>_SC_JOB_CONTROL</td>
<td>_POSIX_JOB_CONTROL</td>
<td>Saved IDs (seteuid()) supported?</td>
</tr>
<tr>
<td>_SC_SAVED_IDS</td>
<td>_POSIX_SAVED_IDS</td>
<td>POSIX.1 version supported.</td>
</tr>
<tr>
<td>_SC_VERSION</td>
<td>_POSIX_VERSION</td>
<td>Number of processors configured.</td>
</tr>
<tr>
<td>_SC_XOPEN_VERSION</td>
<td>XOPEN_VERSION</td>
<td>Number of processors online.</td>
</tr>
<tr>
<td>_SC_JOBNAME_MAX</td>
<td>LOGNAME_MAX</td>
<td>Total number of pages of physical memory in system.</td>
</tr>
<tr>
<td>_SC_NPROCESSORS_CONF</td>
<td></td>
<td>Number physical memory pages not currently in use by system.</td>
</tr>
<tr>
<td>_SC_NPROCESSORS_ONLN</td>
<td></td>
<td>Max number of I/O operations in a single list I/O call supported by implementation.</td>
</tr>
<tr>
<td>_SC_PHYS_PAGES</td>
<td></td>
<td>Max number of outstanding asynchronous I/O operations supported by implementation.</td>
</tr>
<tr>
<td>_SC_AIO_LISTIO_MAX</td>
<td>AIO_LISTIO_MAX</td>
<td>Max amount by which a process can decrease its asynchronous I/O priority level from its own scheduling priority.</td>
</tr>
<tr>
<td>_SC_AIO_MAX</td>
<td>AIO_MAX</td>
<td></td>
</tr>
<tr>
<td>_SC_AIO_PRIO_DELTA_MAX</td>
<td>AIO_PRIO_DELTA_MAX</td>
<td></td>
</tr>
</tbody>
</table>

modified 22 Aug 1995
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_SC_DELAYTIMER_MAX</td>
<td>Max number of timer expiration overruns.</td>
</tr>
<tr>
<td>_SC_MQ_OPEN_MAX</td>
<td>Max number of open message queues a process may hold.</td>
</tr>
<tr>
<td>_SC_MQ_PRIO_MAX</td>
<td>Max number of message priorities supported by implementation.</td>
</tr>
<tr>
<td>_SC_RTSIG_MAX</td>
<td>Max number of realtime signals reserved for application use in this implementation.</td>
</tr>
<tr>
<td>_SC_SEM_NSEMS_MAX</td>
<td>Max number of semaphores that a process may have.</td>
</tr>
<tr>
<td>_SC_SEM_VALUE_MAX</td>
<td>Max value a semaphore may have.</td>
</tr>
<tr>
<td>_SC_SIGQUEUE_MAX</td>
<td>Max number of queued signals that a process may send and have pending at receiver(s) at a time.</td>
</tr>
<tr>
<td>_SC_TIMER_MAX</td>
<td>Max number of timers per process supported by implementation.</td>
</tr>
<tr>
<td>_SC_ASYNCHRONOUS_IO</td>
<td>Supports Asynchronous I/O.</td>
</tr>
<tr>
<td>_SC_FSYNC</td>
<td>Supports File Synchronization.</td>
</tr>
<tr>
<td>_SC_MAPPED_FILES</td>
<td>Supports Memory Mapped Files.</td>
</tr>
<tr>
<td>_SC_MEMLOCK</td>
<td>Supports Process Memory Locking.</td>
</tr>
<tr>
<td>_SC_MEMLOCK_RANGE</td>
<td>Supports Range Memory Locking.</td>
</tr>
<tr>
<td>_SC_MEMORY_PROTECTION</td>
<td>Supports Memory Protection.</td>
</tr>
<tr>
<td>_SC_MESSAGE_PASSING</td>
<td>Supports Message Passing.</td>
</tr>
<tr>
<td>_SC_PRIORITIZED_IO</td>
<td>Supports Prioritized I/O.</td>
</tr>
<tr>
<td>_SC_PRIORITY_SCHEDULING</td>
<td>Supports Process Scheduling.</td>
</tr>
<tr>
<td>_SC_REALTIME_SIGNALS</td>
<td>Supports Realtime Signals.</td>
</tr>
<tr>
<td>_SC_SEMAPHORES</td>
<td>Supports Semaphores.</td>
</tr>
<tr>
<td>_SC_SHARED_MEMORY_OBJECTS</td>
<td>Supports Shared Memory Objects.</td>
</tr>
<tr>
<td>_SC_SYNCHRONIZED_IO</td>
<td>Supports Synchronized I/O.</td>
</tr>
<tr>
<td>_SC_TIMERS</td>
<td>Supports Timers.</td>
</tr>
<tr>
<td>_SC_GETGR_R_SIZE_MAX</td>
<td>Max length of login name.</td>
</tr>
<tr>
<td>_SC_GETPW_R_SIZE_MAX</td>
<td>Number attempts made to destroy thread-specific data upon exit.</td>
</tr>
<tr>
<td><em>SC_THREAD_DESTRUCTOR</em></td>
<td>Max number of data keys per process.</td>
</tr>
<tr>
<td>ITERATIONS</td>
<td>Min byte size of thread stack storage.</td>
</tr>
<tr>
<td>_SC_THREAD_KEYS_MAX</td>
<td>Max number of threads per process.</td>
</tr>
<tr>
<td>_SC_THREAD_STACK_MIN</td>
<td>Max length of terminal device name.</td>
</tr>
<tr>
<td>_SC_THREAD_THREADS_MAX</td>
<td>Supports Threads option.</td>
</tr>
<tr>
<td>_SC_TTY_NAME_MAX</td>
<td>Supports Threads option.</td>
</tr>
</tbody>
</table>

3C-1084 modified 22 Aug 1995
RETURN VALUES
If *name* is an invalid value, *sysconf()* will return -1 and set *errno* to indicate the error. If *sysconf()* fails due to a value of *name* that is not defined on the system, the function will return a value of -1 without changing the value of *errno*.

SEE ALSO
*fpthconf*(2), *seteuid*(2), *setrlimit*(2)

NOTES
A call to *setrlimit()* may cause the value of *OPEN_MAX* to change.
Multiplying *sysconf(_SC_PHYS_PAGES)* or *sysconf(_SC_AVPHYS_PAGES)* by *sysconf(_SC_PAGESIZE)* to determine memory amount in bytes can exceed the maximum values representable in a long or unsigned long.
*SC_PHYS_PAGES* and *SC_AVPHYS_PAGES* are specific to Solaris 2.3 and later releases.
The value of *CLK_TCK* may be variable and it should not be assumed that *CLK_TCK* is a compile-time constant.
NAME    syslog, openlog, closelog, setlogmask – control system log

SYNOPSIS #include <syslog.h>

void openlog(char *ident, int logopt, int facility);
void syslog(int priority, char *logstring, /* parameters */ ...);
void closelog(void);
int setlogmask(int maskpri);

MT-LEVEL Safe

DESCRIPTION syslog() passes a message to syslogd(1M), which may append it to a log file, write it to
the system console, or forward it (to either a list of users or syslogd on another host on
the network), depending on the configuration of /etc/syslog.conf. logstring is tagged with
a priority of priority, and looks like a printf(3B) string with one additional allowable for-
mat specification, %m, which is replaced with the error message string corresponding to
the error number in errno. A trailing NEWLINE is added if needed. Options passed to
openlog() may cause the size of the message to expand. The maximum size of the mes-
sage passed to syslogd is 1024 bytes.

Priorities are encoded as a facility and a level. The facility describes the part of the system
generating the message. The level is selected from the bitwise inclusive OR of zero or
more of the following flags, defined in the header <syslog.h>.

LOG_EMERG    A panic condition. This is normally broadcast to all users.
LOG_ALERT    A condition that should be corrected immediately, such as
             a corrupted system database.
LOG_CRIT     Critical conditions, such as hard device errors.
LOG_ERR      Errors.
LOG_WARNING  Warning messages.
LOG_NOTICE   Conditions that are not error conditions, but that may
             require special handling.
LOG_INFO     Informational messages.
LOG_DEBUG    Messages that contain information normally of use only
             when debugging a program.

If special processing is needed, openlog() can be called to initialize the log file. The
parameter ident is a string that is prepended to every message. logopt is a bit field indicat-
ing logging options. Values for logopt are:

LOG_PID     Log the process ID with each message. This is useful for
            identifying specific daemon processes (for daemons that
            fork).
LOG_CONS
Write messages to the system console if they cannot be sent to syslogd(1M). This option is safe to use in daemon processes that have no controlling terminal, since syslog() forks before opening the console.

LOG_NDELAY
Open the connection to syslogd(1M) immediately. Normally the open is delayed until the first message is logged. This is useful for programs that need to manage the order in which file descriptors are allocated.

LOG_NOWAIT
Do not wait for child processes that have been forked to log messages onto the console. This option should be used by processes that enable notification of child termination using SIGCHLD, since syslog() may otherwise block waiting for a child whose exit status has already been collected.

The facility parameter encodes a default facility to be assigned to all messages that do not have an explicit facility already encoded:

LOG_KERN
Messages generated by the kernel. These cannot be generated by any user processes.

LOG_USER
Messages generated by random user processes. This is the default facility identifier if none is specified.

LOG_MAIL
The mail system.

LOG_DAEMON
System daemons, such as in.ftpd(1M).

LOG_AUTH
The authorization system: login(1), su(1M), getty(1M), etc.

LOG_LPR
The line printer spooling system: lpr(1B), lpc(1B), etc.

LOG_NEWS
Reserved for the USENET network news system.

LOG_UUCP
Reserved for the UUCP system; it does not currently use syslog.

LOG_CRON
The cron/at facility; crontab(1), at(1), cron(1M), etc.

LOG_LOCAL0
Reserved for local use.

LOG_LOCAL1
Reserved for local use.

LOG_LOCAL2
Reserved for local use.

LOG_LOCAL3
Reserved for local use.

LOG_LOCAL4
Reserved for local use.

LOG_LOCAL5
Reserved for local use.

LOG_LOCAL6
Reserved for local use.

LOG_LOCAL7
Reserved for local use.

closelog() can be used to close the log file.
setlogmask() sets the log priority mask to maskpri and returns the previous mask. Calls to syslog() with a priority not set in maskpri are rejected. The mask for an individual priority pri is calculated by the macro LOG_MASK(pri); the mask for all priorities up to and including toppri is given by the macro LOG_UPTO(toppri). The default allows all priorities to be logged.

**EXAMPLES**

This call logs a message at priority LOG_ALERT:

```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");
```

**SEE ALSO** at(1), crontab(1), logger(1), login(1), lpc(1B), lpr(1B), cron(1M), getty(1M), in.ftpd(1M), su(1M), syslogd(1M), printf(3B), syslog.conf(4)
NAME
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);

MT-LEVEL  

MT-Safe

DESCRIPTION  

system() 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.

Solaris  

system() uses /usr/bin/sh (see sh(1)).

XPG4  

system() uses the XPG4-compliant shell /usr/bin/ksh (see ksh(1)).

RETURN VALUES  

system() 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  

system() fails if one or more of the following are true:

EAGAIN  

The system-imposed limit on the total number of processes under execution by a single user would be exceeded.

EINTR  

system() was interrupted by a signal.

ENOMEM  

The new process requires more memory than is available.

SEE ALSO  

ksh(1), sh(1), useradd(1M), exec(2), fork(2), setuid(2), waitpid(2), xpg4(5)

NOTES  

system() 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 connect request

SYNOPSIS  
cc [ flag ...] file ... -lnsl [ library ...]
#include <tiuser.h>
int t_accept(int fildes, int resfd, struct t_call *call);

MT-LEVEL  
MT-Safe

DESCRIPTION  
This function is issued by a transport user to accept a connect request. fildes identifies the
local transport endpoint where the connect 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;

struct netbuf contains the following members:

- unsigned int maxlen;
- unsigned int len;
- char *buf;

In call, addr is the address of the caller, opt indicates any protocol-specific parameters
associated with the connection, udata points to any user data to be returned to the caller,
and sequence is the value returned by t_listen that uniquely associates the response with
a previously received connect indication.

A transport user may accept a connection on either the same, or on a different, local tran-
sport endpoint from the one on which the connect indication arrived. If the same end-
point is specified (that is, resfd=fildes), the connection can be accepted unless the following
condition is true: The user has received other indications on that endpoint but has not
responded to them (with t_accept() or t_snddis(3N)). For this condition, t_accept(3N)
will fail and set t_errno to TBADF.

If a different transport endpoint is specified (resfd! = fildes), the endpoint must be bound to
a protocol address and must be in the T_IDLE state (see t_getstate(3N)) before the
(t_accept(3N) is issued.

For both types of endpoints, t_accept() will fail and set t_errno to TLOOK if there are
indications (for example, a connect or disconnect) waiting to be received on that end-
point.

modified 22 Jan 1993
The values of parameters specified by `opt` and the syntax of those values are protocol specific. The `udata` argument enables the called transport user to send user data to the caller and the amount of user data must not exceed the limits supported by the transport provider, as returned in the `connect` field of the `info` argument of `t_open(3N)` or `t_getinfo(3N)`. If the `len` field of `udata` is zero, no data will be sent to the caller.

**RETURN VALUES**
Upon successful completion, a value of 0 is returned. Otherwise, a value of −1 is returned, `t_errno` is set to indicate the error, and possibly `errno` is set.

**ERRORS**
On failure, `t_errno` will be set to one of the following:

- **TACCES**
  The user does not have permission to accept a connection on the responding transport endpoint or to use the specified options.

- **TBADF**
  The specified file descriptor does not refer to a transport endpoint, or the user is illegally accepting a connection on the same transport endpoint on which the connect indication arrived.

- **TBADDATA**
  The amount of user data specified was not within the bounds allowed by the transport provider.

- **TBADOPT**
  The specified options were in an incorrect format or contained illegal information.

- **TBADSEQ**
  An invalid sequence number was specified.

- **TLOOK**
  An asynchronous event has occurred on the transport endpoint referenced by `fildes` and requires immediate attention.

- **TNOTSUPPORT**
  This function is not supported by the underlying transport provider.

- **TOUTSTATE**
  The function was issued in the wrong sequence on the transport endpoint referenced by `fildes`, or the transport endpoint referred to by `resfd` is not in the `T_IDLE` state.

- **TSYSERR**
  A system error has occurred during execution of this function, `errno` will be set to the specific error.

**SEE ALSO**
`t_accept(3N), t_connect(3N), t_getinfo(3N), t_getstate(3N), t_listen(3N), t_open(3N), t_snddis(3N), t_rcvconnect(3N)`

*Transport Interfaces Programming Guide*

**NOTES**
This interface is safe in multithreaded applications.
NAME  t_alloc — allocate a library structure

SYNOPSIS  cc [ flag ...] file ... -lnsl [ library ... ]
    #include <tiuser.h>
    char *t_alloc(int fildes, int struct_type, int fields);

MT-LEVEL  MT-Safe

DESCRIPTION  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 struct_type, and 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

where each of these structures may subsequently be used as an argument to one or more transport functions.

Each of the above structures, except T_INFO, contains at least one field of type struct netbuf. netbuf is described in t_connect(3N). For each field of this type, the user may specify that the buffer for that field should be allocated as well. The fields argument specifies this option, where the argument is the bitwise-OR of any of the following:

T_ADDR  The addr field of the t_bind, t_call, t_unitdata, or t_uderr structures.
T_OPT   The opt field of the t_optmgmt, t_call, t_unitdata, or t_uderr structures.
T_UDATA The udata field of the t_call, t_discon, or t_unitdata structures.
T_ALL   All relevant fields of the given structure.

For each field specified in fields, t_alloc() will allocate memory for the buffer associated with the field, and initialize the buf pointer and maxlen (see netbuf in t_connect(3N) for description of buf and maxlen) field accordingly. The length of the buffer allocated will be based on the same size information that is returned to the user on t_open(3N) and t_getinfo(3N). Thus, fildes must refer to the transport endpoint through which the newly allocated structure will be passed, so that the appropriate size information can be accessed. If the size value associated with any specified field is −1, the underlying service provider can support a buffer of unlimited size. If this is the case, t_alloc() will allocate a buffer with the default size 1024 bytes. See the NOTES section for information regarding memory allocation for buffers other than 1024 bytes. If the size value is −2, t_alloc() will set the buffer pointer to NULL and the buffer maximum size to 0, and then will return...
success (see `t_open(3N)` or `t_getinfo(3N)`).
For any field not specified in `fields`, `buf` will be set to NULL and `maxlen` will be set to zero.
Use of `t_alloc()` to allocate structures will help ensure the compatibility of user programs
with future releases of the transport interface.

**RETURN VALUES**

On successful completion, `t_alloc()` returns a pointer to the newly allocated structure.
On failure, NULL is returned, `t_errno` is set to indicate the error, and possibly `errno` is set.

**ERRORS**

On failure, `t_errno` will be set to one of the following:

- **TBADF**  
The specified file descriptor does not refer to a transport endpoint.
- **TSYSERR**  
A system error has occurred during execution of this function, `errno` will be set to the specific error.

**SEE ALSO**

`t_connect(3N), t_free(3N), t_getinfo(3N), t_open(3N)`

*Transport Interfaces Programming Guide*

**NOTES**

If the underlying service provider supports a buffer of unlimited size in the `netbuf` structure (see `t_connect(3N)`), `t_alloc()` will return a buffer of size 1024 bytes. If a larger size buffer is required, it will need to be allocated separately using a memory allocation routine such as `malloc(3C)`. The `buf` and `maxlen` fields of the `netbuf` data structure can then be updated with the address of the new buffer and the 1024 byte buffer originally allocated by `t_alloc()` can be freed using `free(3C)`.

This interface is safe in multithreaded applications.
T_BASE – bind an address to a transport endpoint

SYNOPSIS
c c [ flag . . ] file . . . −lnsl [ library . . ]
#include <tiuser.h>

int t_bind(int fildes, const struct t_bind *req, struct t_bind *ret);

MT-LEVEL
MT-Safe

DESCRIPTION
This function associates a protocol address with the transport endpoint specified by fildes
and activates that transport endpoint. In connection mode, the transport provider may
begin accepting or requesting connections on the transport endpoint. In connectionless
mode, the transport user may send or receive data units through the transport endpoint.
The req and ret arguments point to a struct 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 outstanding
connect indications.

req is used to request that an address, represented by the netbuf structure, be bound to
the given transport endpoint. len (see netbuf in t_connect(3N); also for buf and maxlen)
specifies the number of bytes in the address and buf points to the address buffer. maxlen
has no meaning for the req argument. On return, ret contains the address that the trans-
port provider actually bound to the transport endpoint; this may be different from the
address specified by the user in req. In ret, the user specifies maxlen, which is the max-
imum size of the address buffer, and buf, which points to the buffer where the address is
to be placed. On return, len specifies the number of bytes in the bound address and buf
points to the bound address. If maxlen is not large enough to hold the returned address,
an error will result.

If the requested address is not available, or if no address is specified in req (the len field
of addr in req is zero) the transport provider may assign an appropriate address to be
bound, and will return that address in the addr field of ret. 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.

req may be NULL if the user does not wish to specify an address to be bound. Here, the
value of qlen is assumed to be zero, and the transport provider must assign an address to
the transport endpoint. Similarly, ret may be NULL if the user does not care what address
was bound by the provider and is not interested in the negotiated value of qlen. It is
valid to set req and ret to NULL for the same call, in which case the provider chooses the
address to bind to the transport endpoint and does not return that information to the
user.
The `qlen` field has meaning only when initializing a connection-mode service. It specifies
the number of outstanding connect indications the transport provider should support for
the given transport endpoint. An outstanding connect indication is one that has been
passed to the transport user by the transport provider. A value of `qlen` greater than zero
is only meaningful when issued by a passive transport user that expects other users to
call it. The value of `qlen` will be negotiated by the transport provider and may be
changed if the transport provider cannot support the specified number of outstanding
connect indications. On return, the `qlen` field in `ret` will contain the negotiated value.

This function allows more than one transport endpoint to be bound to the same protocol
address (however, the transport provider must support this capability also), but it is not
allowable to bind more than one protocol address to the same transport endpoint. If a
user binds more than one transport endpoint to the same protocol address, only one end-
point can be used to listen for connect indications associated with that protocol address.
In other words, only one `t_bind()` for a given protocol address may specify a value of
`qlen` greater than zero. In this way, the transport provider can identify which transport
endpoint should be notified of an incoming connect indication. If a user attempts to bind
a protocol address to a second transport endpoint with a value of `qlen` greater than zero,
the transport provider will assign another address to be bound to that endpoint. If a user
accepts a connection on the transport endpoint that is being used as the listening end-
point, the bound protocol address will be found to be busy for the duration of that con-
nection. No other transport endpoints may be bound for listening while that initial listen-
ing endpoint is in the data transfer phase. This will prevent more than one transport end-
point bound to the same protocol address from accepting connect indications.

**RETURN VALUES**

`t_bind()` returns 0 on success. On failure, `t_bind()` returns −1, `t_errno` is set to indicate
the error, and possibly `errno` is set.

**ERRORS**

On failure, `t_errno` will be set to one of the following:

- **TACCESS**  The user does not have permission to use the specified address.
- **TBADADDR**  The specified protocol address was in an incorrect format or con-
tained illegal information.
- **TBADF**  The specified file descriptor does not refer to a transport endpoint.
- **TBUFOVFLW**  The number of bytes allowed for an incoming argument is not
sufficient to store the value of that argument. The provider’s state
will change.
- **TNOADDR**  The transport provider could not allocate an address. `T_IDLE` and
the information to be returned in `ret` will be discarded.
TOUTSTATE  The function was issued in the wrong sequence.
T SYSERR  A system error has occurred during execution of this function,
          **errno** will be set to the specific error.

SEE ALSO  **t_connect**(3N), **t_open**(3N), **t_optmgmt**(3N), **t_unbind**(3N)

Transport Interfaces Programming Guide

NOTES  This interface is safe in multithreaded applications.
NAME  
t_close – close a transport endpoint

SYNOPSIS  
cc [ flag ... ] file ... -lnsl [ library ... ]
#include <tiuser.h>
int t_close(int fildes);

MT-LEVEL  
MT-Safe

DESCRIPTION  
The t_close() function informs the transport provider that the user is finished with the
transport endpoint specified by fildes, and frees any local library resources associated
with the endpoint. In addition, t_close() closes the file associated with the transport end-
point.

t_close() should be called from the T_UNBND state (see t_getstate(3N)). However, this
function does not check state information, so it may be called from any state to close a
transport endpoint. If this occurs, the local library resources associated with the endpoint
will be freed automatically. In addition, close(2) will be issued for that file descriptor; if
no other process has the file descriptor open, the close will terminate any connection that
may be associated with that endpoint. The connection termination will be abortive or
orderly depending on the service type supported by the underlying transport provider.

RETURN VALUES  
t_close returns 0 on success. On failure t_close returns -1, t_errno is set to indicate the
error, and possibly errno is set.

ERRORS  
On failure, t_errno will be set to the following:

TBADF     The specified file descriptor does not refer to a transport endpoint.
TSYSERR   A system error occurred during execution of this function, errno
          will be set to the specific error.

SEE ALSO  
close(2), t_getstate(3N), t_open(3N), t_unbind(3N)

Transport Interfaces Programming Guide

NOTES  
This interface is safe in multithreaded applications.
NAME  t_connect – establish a connection with another transport user

SYNOPSIS  cc [ flag  ... ] file  ...  –lnsl [ library  ... ]
#include <tiuser.h>
int t_connect(int fildes, const struct t_call *sndcall, struct t_call *rcvcall);

MT-LEVEL  MT-Safe

DESCRIPTION  This function enables a transport user to request a connection to the specified destination transport user. fildes identifies the local transport endpoint where communication will be established, while sndcall and rcvcall point to a t_call structure that contains the following members:

    struct netbuf addr;
    struct netbuf opt;
    struct netbuf udata;
    int sequence;

sndcall specifies information needed by the transport provider to establish a connection and rcvcall specifies information that is associated with the newly established connection. The address is specified in the netbuf structure which has the following format:

    struct netbuf {
        unsigned int maxlen;
        unsigned int len;
        char *buf;
    }

where maxlen specifies the maximum length of the buffer in bytes, len specifies the bytes of data in the buffer, and buf points to the buffer that contains the data.

In sndcall, addr specifies the protocol address of the destination transport user, opt presents any protocol-specific information that might be needed by the transport provider, udata points to optional user data that may be passed to the destination transport user during connection establishment, and sequence has no meaning for this function.

On return in rcvcall, addr returns the protocol address associated with the responding transport endpoint, opt presents any protocol-specific information associated with the connection, udata points to optional user data that may be returned by the destination transport user during connection establishment, and sequence has no meaning for this function.

The opt argument implies no structure on the options that may be passed to the transport provider. The transport provider is free to specify the structure of any options passed to it. These options are specific to the underlying protocol of the transport provider. The user may choose not to negotiate protocol options by setting the len field of opt to zero. In this case, the provider may use default options.
The `udata` argument enables the caller to pass user data to the destination transport user and receive user data from the destination user during connection establishment. However, the amount of user data must not exceed the limits supported by the transport provider as returned in the `connect` field of the `info` argument of `t_open(3N)` or `t_getinfo(3N)`. If the `len` (see `netbuf` in `t_connect(3N)`) field of `udata` is zero in `sndcall`, no data will be sent to the destination transport user.

On return, the `addr`, `opt`, and `udata` fields of `rcvcall` will be updated to reflect values associated with the connection. Thus, the `maxlen` (see `netbuf` in `t_connect(3N)`) field of each argument must be set before issuing this function to indicate the maximum size of the buffer for each. However, `rcvcall` may be NULL, in which case no information is given to the user on return from `t_connect( )`.

By default, `t_connect( )` executes in synchronous mode, and will wait for the destination user’s response before returning control to the local user. A successful return (that is, return value of zero) indicates that the requested connection has been established. However, if `O_NDELAY` or `O_NONBLOCK` is set (using `t_open(3N)` or `fcntl(2)`), `t_connect( )` executes in asynchronous mode. In this case, the call will not wait for the remote user’s response, but will return control immediately to the local user and return -1 with `t_errno` set to `T_NODATA` to indicate that the connection has not yet been established. In this way, the function simply initiates the connection establishment procedure by sending a connect request to the destination transport user.

**RETURN VALUES**

- `t_connect( )` returns 0 on success. On failure `t_connect( )` returns -1, `t_errno` is set to indicate the error, and possibly `errno` is set.

**ERRORS**

On failure, `t_errno` will be set to one of the following:

- **T_ACESS**
  The user does not have permission to use the specified address or options.

- **T_BADADDR**
  The specified protocol address was in an incorrect format or contained illegal information.

- **T_BADDATA**
  The amount of user data specified was not within the bounds allowed by the transport provider.

- **T_BADDF**
  The specified file descriptor does not refer to a transport endpoint.

- **T_BADOPT**
  The specified protocol options were in an incorrect format or contained illegal information.

- **TBUF_OVERFLOW**
  The number of bytes allocated for an incoming argument is not sufficient to store the value of that argument. If executed in synchronous mode, the provider’s state, as seen by the user, changes to `T_DATAFER`, and the `connect` indication information to be returned in `rcvcall` is discarded.

- **TLOOK**
  An asynchronous event has occurred on this transport endpoint and requires immediate attention.

- **T_NODATA**
  `O_NDELAY` or `O_NONBLOCK` was set, so the function successfully initiated the connection establishment procedure, but did not wait.
for a response from the remote user.

**TNOTSUPPORT**  This function is not supported by the underlying transport provider.

**TOUTSTATE**  The function was issued in the wrong sequence.

**TSYSERR**  A system error has occurred during execution of this function, `errno` will be set to the specific error.

**SEE ALSO**  `fcntl(2)`, `t_accept(3N)`, `t_getinfo(3N)`, `t_listen(3N)`, `t_open(3N)`, `t_optmgmt(3N)`, `t_rcvconnect(3N)`

`Transport Interfaces Programming Guide`

**NOTES**  This interface is safe in multithreaded applications.
NAME  
t_error – produce error message

SYNOPSIS  
cc [ flag ... ] file ... -lnsl [ library ... ]
#include <tiuser.h>
void t_error(const char *errmsg);
extern int t_errno;
extern char *t_errlist[];
extern int t_nerr;

MT-LEVEL  
MT-Safe

DESCRIPTION  
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.

t_error() prints the user-supplied error message followed by a colon and the standard transport function error message for the current value contained in t_errno. If t_errno is TSYSERR, t_error will also print the standard error message for the current value contained in errno (see intro(2)).

t_errlist is the array of message strings, to allow user message formatting. t_errno can be used as an index into this array to retrieve the error message string (without a terminating newline). t_nerr is the maximum index value for the t_errlist array.

t_errno is set when an error occurs and is not cleared on subsequent successful calls.

EXAMPLES  
If a t_connect() 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 print as:

t_connect failed on fd2: Incorrect transport address format

where "t_connect failed on fd2" tells the user which function failed on which transport endpoint, and ‘Incorrect transport address format’ identifies the specific error that occurred.

SEE ALSO  
intro(2)

Transport Interfaces Programming Guide

WARNINGS  
This interface is deprecated. Use t_strerror(3N) instead. For details on the ramifications of referencing t_errlist[] directly, please refer to the Copy Relocations section of Chapter 4 in the Linker and Libraries Guide.

NOTES  
This interface is safe in multithreaded applications.
NAME  
t_free – free a library structure

SYNOPSIS  
cc [ flag . . .] file . . . -lnsl [ library . . .]
#include <tiuser.h>
int t_free(char *ptr, int struct_type);

MT-LEVEL  
MT-Safe

DESCRIPTION  
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 refer-
enced by the structure.

ptr points to one of the six structure types described for t_alloc(3N), and struct_type
identifies the type of that structure, 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

where each of these structures is used as an argument to one or more transport functions.

The t_free() will check the addr, opt, and udata fields of the given structure (as appropriate),
and free the buffers pointed to by the buf field of the netbuf (see t_connect(3N)) struc-
ture. If buf is NULL, t_free() will not attempt to free memory. After all buffers are freed,
t_free() will free the memory associated with the structure pointed to by 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(3N).

RETURN VALUES  
t_free returns 0 on success. On failure t_free returns -1, t_errno is set to indicate the
error, and possibly errno is set.

ERRORS  
On failure, t_errno will be set to the following:

TSYSERR A system error has occurred during execution of this function, errno
will be set to the specific error.

SEE ALSO  
t_connect(3N), t_alloc(3N)
Transport Interfaces Programming Guide

NOTES  
This interface is safe in multithreaded applications.

modified 27 Jan 1994

3N-1103
NAME  
t_getinfo – get protocol-specific service information

SYNOPSIS  
cc [ flag ... ] file ... -lnsl [ library ... ]
#include <tiuser.h>
int t_getinfo(int fildes, struct t_info *info);

MT-LEVEL  
MT-Safe

DESCRIPTION  
This function returns the current characteristics of the underlying transport protocol
associated with file descriptor fildes. The info structure is used to return the same infor-
mation returned by t_open(3N). 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 of transport protocol address */
long options; /* max number of bytes of protocol-specific options */
long tsdu; /* max size of transport service data unit (TSDU) */
long etsd; /* max size of expedited transport service data unit (ETSDU) */
long connect; /* max amount of data allowed on connection establishment
functions */
long discon; /* max amount of data allowed on t_snddis(3N) and
t_rcvdis(3N) functions */
long servtype; /* service type supported by the transport provider */

The values of the fields have the following meanings:

addr  
A value greater than or equal to zero indicates the maximum size of a
transport protocol address; a value of −1 specifies that there is no limit on
the address size; and a value of −2 specifies that the transport provider
does not provide user access to transport protocol addresses.

options  
A value greater than or equal to zero indicates the maximum number of
bytes of protocol-specific options supported by the provider; a value of
−1 specifies that there is no limit on the option size; and a value of −2 specifies that the transport provider does not support user-settable
options.

tsdu  
A value greater than zero specifies the maximum size of a transport ser-
vice data unit (TSDU); a value of zero specifies that the transport provider
does not support the concept of TSDU, although it does support the send-
ing across a connection of a data stream with no logical boundaries
preserved; a value of −1 specifies that there is no limit on the size of a
TSDU; and a value of −2 specifies that the transfer of normal data is not
supported by the transport provider.

3N-1104 modified 22 Jan 1993
etsdu
A value greater than zero specifies the maximum size of an expedited transport service data unit (ETSDU); a value of zero specifies that the transport provider does not support the concept of ETSDU, although it does support the sending across a connection of an expedited data stream with no logical boundaries preserved. A value of −1 specifies that there is no limit on the size of an ETSDU; and a value of −2 specifies that the transfer of expedited data is not supported by the transport provider.

connect
A value greater than or equal to zero specifies the maximum amount of data that may be associated with connection establishment functions; a value of −1 specifies that there is no limit on the amount of data sent during connection establishment; and a value of −2 specifies that the transport provider does not allow data to be sent with connection establishment functions.

discon
A value greater than or equal to zero specifies the maximum amount of data that may be associated with the t_snddis(3N) and t_rcvdis(3N) functions; a value of −1 specifies that there is no limit on the amount of data sent with these abortive release functions; and a value of −2 specifies that the transport provider does not allow data to be sent with the abortive release functions.

servtype
This field specifies the service type supported by the transport provider, as described below.

If a transport user is concerned with protocol independence, the above sizes may be accessed to determine how large the buffers must be to hold each piece of information. Alternatively, the t_alloc() function may be used to allocate these buffers. An error will result if a transport user exceeds the allowed data size on any function. The value of each field may change as a result of option negotiation, and t_getinfo() enables a user to retrieve the current characteristics.

The servtype field of info may specify one of the following values on return:

T_COTS
The transport provider supports a connection-mode service but does not support the optional orderly release facility.

T_COTS_ORD
The transport provider supports a connection-mode service with the optional orderly release facility.

T_CLTS
The transport provider supports a connectionless-mode service. For this service type, t_open(3N) will return −2 for etsdu, connect, and discon.

RETURN VALUES
 t_getinfo() returns 0 on success. On failure t_getinfo() returns −1, t_errno is set to indicate the error, and possibly errno is set.
Errors
On failure, `t_errno` will be set to one of the following:

- **TBADF**: The specified file descriptor does not refer to a transport endpoint.
- **TSYSERR**: A system error has occurred during execution of this function, `errno` will be set to the specific error.

See Also
`t_open(3N), t_rcvdis(3N), t_snddis(3N)`

Transport Interfaces Programming Guide

Notes
This interface is safe in multithreaded applications.
NAME  
t_getstate – get the current state

SYNOPSIS  
cc [ flag ... ] file ... -lnsl [ library ... ]
#include <tiuser.h>
int t_getstate(int fildes);

MT-LEVEL  
MT-Safe

DESCRIPTION  
The t_getstate() function returns the current state of the provider associated with the
transport endpoint specified by fildes.

RETURN VALUES  
t_getstate() returns the current state on successful completion. On failure t_getstate()
returns −1, t_errno is set to indicate the error, and possibly errno is set. The current state
may be one of the following:
T_UNBND    unbound
T_IDLE     idle
T_OUTCON   outgoing connection pending
T_INCON    incoming connection pending
T_DATAXFER data transfer
T_OUTREL   outgoing orderly release (waiting for an orderly release indication)
T_INREL    incoming orderly release (waiting for an orderly release request)
If the provider is undergoing a state transition when t_getstate() is called, the function
will fail.

ERRORS  
On failure, t_errno will be set to one of the following:
TBADF    The specified file descriptor does not refer to a transport endpoint.
TSTATECHNG The transport provider is undergoing a state change.
TSYSERR   A system error has occurred during execution of this function, errno
           will be set to the specific error.

SEE ALSO  
t_open(3N)

Transport Interfaces Programming Guide

NOTES  
This interface is safe in multithreaded applications.

modified 22 Jan 1993
t_listen – listen for a connect request

NAME

SYNOPSIS

cc [ flag ... ] file ... -lnsl [ library ... ]
#include <tuser.h>
int t_listen(int fildes, struct t_call *call);

MT-LEVEL

DESCRIPTION

This function listens for a connect request from a calling transport user. fildes identifies the local transport endpoint where connect indications arrive, and on return, call contains information describing the connect indication. 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). In call, addr returns the protocol address of the calling transport user, opt returns protocol-specific parameters associated with the connect request, udata returns any user data sent by the caller on the connect request, and sequence is a number that uniquely identifies the returned connect indication. The value of sequence enables the user to listen for multiple connect indications before responding to any of them.

Since this function returns values for the addr, opt, and udata fields of call, the maxlen (see netbuf in t_connect(3N)) field of each must be set before issuing t_listen() to indicate the maximum size of the buffer for each.

By default, t_listen() executes in synchronous mode and waits for a connect indication to arrive before returning to the user. However, if O_NDELAY or O_NONBLOCK is set (using t_open() or fcntl()), t_listen() executes asynchronously, reducing to a poll for existing connect indications. If none are available, it returns −1 and sets t_errno to TNO-DATA.

RETURN VALUES

t_listen() returns 0 on success. On failure t_listen() returns −1, t_errno is set to indicate the error, and possibly errno is set.

ERRORS

On failure, t_errno will be set to one of the following:

TBADF The specified file descriptor does not refer to a transport endpoint.
TBUFOVFLW The number of bytes allocated for an incoming argument is not sufficient to store the value of that argument. The provider’s state, as seen by the user, changes to T_INCON, and the connect indication information to be returned in call is discarded.
<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TLOOK</td>
<td>An asynchronous event has occurred on this transport endpoint and requires immediate attention.</td>
</tr>
<tr>
<td>TNODATA</td>
<td><code>O_NDELAY</code> or <code>O_NONBLOCK</code> was set, but no connect indications had been queued.</td>
</tr>
<tr>
<td>TNOTSUPPORT</td>
<td>This function is not supported by the underlying transport provider.</td>
</tr>
<tr>
<td>TSYSERR</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>

**SEE ALSO**
- `t_accept(3N)`, `t_bind(3N)`, `t_connect(3N)`, `t_open(3N)`, `t_rcvconnect(3N)`
- *Transport Interfaces Programming Guide*

**WARNINGS**
If a user issues `t_listen()` in synchronous mode on a transport endpoint that was not bound for listening (that is, `qlen` was zero on `t_bind()`), the call will wait forever because no connect indications will arrive on that endpoint.

**NOTES**
This interface is safe in multithreaded applications.

modified 27 Jan 1994
NAME  

t_look – look at the current event on a transport endpoint

SYNOPSIS  

cc [ flag ... ] file ... -lnsl [ library ... ]
#include <tiuser.h>
int t_look(int fildes);

MT-LEVEL

MT-Safe

DESCRIPTION

This function returns the current event on the transport endpoint specified by fildes. This function enables a transport provider to notify a transport user of an asynchronous event when the user is issuing functions in synchronous mode. Certain events require immediate notification of the user and are indicated by a specific error, TLOOK, on the current or next function to be executed.

This function also enables a transport user to poll a transport endpoint periodically for asynchronous events.

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_LISTEN  
connection indication received

T_CONNECT  
connect confirmation received

T_DATA  
normal data received

T_EXDATA  
expedited data received

T_DISCONNECT  
disconnect received

T_UDERR  
datagram error indication

T_ORDREL  
orderly release indication

On failure, -1 is returned, t_errno is set to indicate the error, and possibly errno is set.

ERRORS

On failure, t_errno will be set to one of the following:

TBADF  
The specified file descriptor does not refer to a transport endpoint.

TSYSERR  
A system error has occurred during execution of this function, errno will be set to the specific error.

SEE ALSO

t_open(3N)

Transport Interfaces Programming Guide

NOTES

This interface is safe in multithreaded applications.
NAME  t_open – establish a transport endpoint

SYNOPSIS  cc [ flag ... ] file ... -lnsl [ library ... ]
#include <tiuser.h>
#include <fcntl.h>

int t_open(const char *path, int oflag, struct t_info *info);

MT-LEVEL  MT-Safe

DESCRIPTION  t_open() must be called as the first step in the initialization of a transport endpoint. This
function establishes a transport endpoint by opening a file that identifies a particular
transport provider (that is, transport protocol) and returning a file descriptor that
identifies that endpoint. For example, opening the file /dev/iso_cots identifies an OSI
connection-oriented transport layer protocol as the transport provider.

path points to the path name of the file to open, and oflag identifies any open flags (as in
open(2)). oflag may be constructed from O_NDELAY or O_NONBLOCK OR-ed with
O_RDWR. These flags are defined in the header <fcntl.h>. t_open() returns a file
descriptor that will be used by all subsequent functions to identify the particular local
transport endpoint.

This function also returns various default characteristics of the underlying transport pro-
tocol by setting fields in the t_info structure. See t_getinfo(3N) for a description of the
t_info structure.

If info is set to NULL by the transport user, no protocol information is returned by
t_open().

RETURN VALUES  t_open() returns a valid file descriptor on success. On failure t_open() returns −1,
t_errno is set to indicate the error, and possibly errno is set.

ERRORS  On failure, t_errno will be set to the following:

TBADFLAG  An invalid flag is specified.

TSYSERR  A system error has occurred during execution of this function,
errno will be set to the specific error.

SEE ALSO  open(2), t_getinfo(3N)
Transport Interfaces Programming Guide

NOTES  This interface is safe in multithreaded applications.

modified 22 Jan 1993  3N-1111
NAME  
\*t_optmgmt - manage options for a transport endpoint\* 

SYNOPSIS  
\*cc [ flag ...] file ... -lnsl [ library ...] \*
\#include <tiuser.h> 
\*int t_optmgmt(int fildes, const struct t_optmgmt *req, struct t_optmgmt *ret); \*

MT-LEVEL  
MT-Safe 

DESCRIPTION  
The \*t_optmgmt()\* function enables a transport user to retrieve, verify, or negotiate protocol options with the transport provider. \*fildes\* identifies a bound transport endpoint. 

The \*req\* and \*ret\* arguments point to a \*t_optmgmt\* structure containing the following members:

\verbatim
struct netbuf opt;  
long flags; 
\endverbatim

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\* (see \*t_connect(3N)\*; also for \*len, buf, and maxlen\*) structure in a manner similar to the address in \*t_bind(3N)\*. \*req\* is used to request a specific action of the provider and to send options to the provider. \*len\* specifies the number of bytes in the options, \*buf\* points to the options buffer, and \*maxlen\* has no meaning for the \*req\* argument. The transport provider may return options and flag values to the user through \*ret\*. For \*ret\*, \*maxlen\* specifies the maximum size of the options buffer and \*buf\* points to the buffer where the options are to be placed. On return, \*len\* specifies the number of bytes of options returned. \*maxlen\* has no meaning for the \*req\* argument, but must be set in the \*ret\* argument to specify the maximum number of bytes the options buffer can hold. The actual structure and content of the options is imposed by the transport provider.

The \*flags\* field of \*req\* can specify one of the following 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.
If issued as part of the connectionless-mode service, `t_optmgmt()` may block due to flow control constraints. The function will not complete until the transport provider has processed all previously sent data units.

**RETURN VALUES**

`t_optmgmt()` returns 0 on success. On failure `t_optmgmt()` returns -1, `t_errno` is set to indicate the error, and possibly `errno` is set.

**ERRORS**

On failure, `t_errno` will be set to one of the following:

- **TBADF**
  The specified file descriptor does not refer to a transport endpoint.

- **TACCES**
  The user does not have permission to negotiate the specified options.

- **TBADFLAG**
  An invalid flag was specified.

- **TBADOPT**
  The specified protocol options were in an incorrect format or contained illegal information.

- **TBUFOVFLW**
  The number of bytes allowed for an incoming argument is not sufficient to store the value of that argument. The information to be returned in `ret` will be discarded.

- **TOUTSTATE**
  The function was issued in the wrong sequence.

- **TSYSERR**
  A system error has occurred during execution of this function, `errno` will be set to the specific error.

**SEE ALSO**

`t_bind(3N), t_connect(3N), t_getinfo(3N), t_open(3N)`

*Transport Interfaces Programming Guide*

**NOTES**

This interface is safe in multithreaded applications.
NAME
t_rcv – receive data or expedited data sent over a connection

SYNOPSIS
cc [ flag ...] file ... -lnsl [ library ...]
int t_rcv(int fildes, char *buf, unsigned nbytes, int *flags);

MT-LEVEL
MT-Safe

DESCRIPTION
This function receives either normal or expedited data. fildes 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_NDELAY or O_NONBLOCK is set (using t_open(3N) or fcntl(2)), t_rcv() will execute in asynchronous mode and will fail if no data is available. (See TNODATA below.)

On return from the call, if T_MORE is 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) must be received in multiple t_rcv() calls. Each t_rcv() with the T_MORE flag set indicates that another t_rcv() must follow to get more data for the current TSDU. The end of the TSDU is identified by the return of a t_rcv() call with the T_MORE flag not set. If the transport provider does not support the concept of a TSDU as indicated in the info argument on return from t_open(3N) or t_getinfo(3N), the T_MORE flag is not meaningful and should be ignored.

On return, the data returned is expedited data if T_EXPEDITED is set in flags. If the number of bytes of expedited data exceeds nbytes, t_rcv() will set T_EXPEDITED and T_MORE on return from the initial call. Subsequent calls to retrieve the remaining ETSDU will have T_EXPEDITED set on return. The end of the ETSDU is identified by the return of a t_rcv() call with the T_MORE flag not set.

If expedited data arrives after part of a ETSDU has been retrieved, receipt of the remainder of the TSDU will be suspended until the ETSDU has been processed. Only after the full ETSDU has been retrieved (T_MORE not set) will the remainder of the TSDU be available to the user.

RETURN VALUES
On successful completion, t_rcv() returns the number of bytes received. On failure t_rcv() returns -1, t_errno is set to indicate the error, and possibly errno is set.

ERRORS
On failure, t_errno will be set to one of the following:
TBADF The specified file descriptor does not refer to a transport endpoint.
TLOOK An asynchronous event has occurred on this transport endpoint and requires immediate attention.
TNODATA O_NDELAY or O_NONBLOCK was set, but no data is currently available from the transport provider.
<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TNOTSUPPORT</td>
<td>This function is not supported by the underlying transport provider.</td>
</tr>
<tr>
<td>TSYSERR</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>

**SEE ALSO**  
`fcntl(2), t_getinfo(3N), t_open(3N), t_snd(3N)`

*Transport Interfaces Programming Guide*

**NOTES**  
This interface is safe in multithreaded applications.
NAME  t_rcvconnect – receive the confirmation from a connect request

SYNOPSIS  cc [ flag  ... ] file  ...  −lnsl [ library  ... ]
# include <tiuser.h>
int t_rcvconnect(int fildes, struct t_call *call);

MT-LEVEL  MT-Safe

DESCRIPTION  This function enables a calling transport user to determine the status of a previously sent connect request and is used in conjunction with t_connect(3N) to establish a connection in asynchronous mode. The connection will be established on successful completion of this function.

fildes identifies the local transport endpoint where communication will be established, and call contains information associated with the newly established connection. 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). In call, addr returns the protocol address associated with the responding transport endpoint, opt presents any protocol-specific information associated with the connection, udata points to optional user data that may be returned by the destination transport user during connection establishment, and sequence has no meaning for this function.

The maxlen (see netbuf in t_connect(3N)) field of each argument must be set before issuing this function to indicate the maximum size of the buffer for each. However, call may be NULL, in which case no information is given to the user on return from t_rcvconnect(). By default, t_rcvconnect() executes in synchronous mode and waits for the connection to be established before returning. On return, the addr, opt, and udata fields reflect values associated with the connection.

If O_NDELAY or O_NONBLOCK is set (using t_open(3N) or fcntl(2)), t_rcvconnect() executes in asynchronous mode, and reduces to a poll for existing connect confirmations. If none are available, t_rcvconnect() fails and returns immediately without waiting for the connection to be established. (See TNODATA below.) t_rcvconnect() must be re-issued at a later time to complete the connection establishment phase and retrieve the information returned in call.

RETURN VALUES  t_rcvconnect() returns 0 on success. On failure t_rcvconnect() returns −1, t_errno is set to indicate the error, and possibly errno is set.
On failure, \texttt{t_errno} will be set to one of the following:

- **TBADF**: The specified file descriptor does not refer to a transport endpoint.
- **TBUFOVFLW**: The number of bytes allocated for an incoming argument is not sufficient to store the value of that argument and the connect information to be returned in \textit{call} will be discarded. The provider’s state, as seen by the user, will be changed to \texttt{DATAXFER}.
- **TLOOK**: An asynchronous event has occurred on this transport connection and requires immediate attention.
- **TNODATA**: \texttt{O_NDELAY} or \texttt{O_NONBLOCK} was set, but a connect confirmation has not yet arrived.
- **TNOTSUPPORT**: This function is not supported by the underlying transport provider.
- **TSYSERR**: A system error has occurred during execution of this function, \texttt{errno} will be set to the specific error.

See also \texttt{fcntl(2)}, \texttt{t_accept(3N)}, \texttt{t_bind(3N)}, \texttt{t_connect(3N)}, \texttt{t_listen(3N)}, \texttt{t_open(3N)}

Transport Interfaces Programming Guide

Notes: This interface is safe in multithreaded applications.
NAME  
t_rcvdis – retrieve information from disconnect

SYNOPSIS  
cc [ flag . . . ] file . . . -lnsl [ library . . . ]
#include <tiuser.h>
int t_rcvdis(int fd, struct t_discon *discon);

MT-LEVEL  
MT-Safe

DESCRIPTION  
This function is used to identify the cause of a disconnect, and to retrieve any user data
sent with the disconnect.  
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;
/*
 struct netbuf {
 unsigned int maxlen;
 unsigned int len;
 char *buf;
 };
*/
```

reason specifies the reason for the disconnect through a protocol-dependent reason code,
udata identifies any user data that was sent with the disconnect, and sequence may identify an outstanding connect indication with which the disconnect is associated.  sequence is only meaningful when t_rcvdis() is issued by a passive transport user who has executed one or more t_listen(3N) functions and is processing the resulting connect indications.  If a disconnect indication occurs, sequence can be used to identify which of the outstanding connect indications is associated with the disconnect.

If a user does not care if there is incoming data and does not need to know the value of reason or sequence, discon may be NULL and any user data associated with the disconnect will be discarded.  However, if a user has retrieved more than one outstanding connect indication (using t_listen(3N) ) and discon is NULL, the user will be unable to identify which connect indication the disconnect is associated with.

RETURN VALUES  
t_rcvdis returns 0 on success.  On failure t_rcvdis returns -1, t_errno is set to indicate the error, and possibly errno is set.

ERRORS  
On failure, t_errno will be set to one of the following:

TBADF  
The specified file descriptor does not refer to a transport endpoint.

3N-1118  
modified 22 Jan 1993
TBUFOVFLW  The number of bytes allocated for incoming data is not sufficient to store the data. The provider’s state, as seen by the user, will change to T_IDLE, and the disconnect indication information to be returned in discon will be discarded.

TNODIS  No disconnect indication currently exists on the specified transport endpoint.

TNOTSUPPORT  This function is not supported by the underlying transport provider.

TSYSERR  A system error has occurred during execution of this function, errno will be set to the specific error.

SEE ALSO  t_connect(3N), t_listen(3N), t_open(3N), t_snddis(3N)

Transport Interfaces Programming Guide

NOTES  This interface is safe in multithreaded applications.
NAME  t_rcvrel – acknowledge receipt of an orderly release indication

SYNOPSIS  cc [ flag . . . ] file . . . −lnsl [ library . . . ]
        #include <tiuser.h>
        int t_rcvrel(int fildes);

MT-LEVEL  MT-Safe

DESCRIPTION  This function is used to acknowledge receipt of an orderly release indication. fildes identifies the local transport endpoint where the connection exists. After receipt of this indication, the user should not attempt to receive more data because such an attempt will block forever. However, the user may continue to send data over the connection if t_sndrel() has not been issued by the user.

This function is an optional service of the transport provider, and is only supported if the transport provider returned service type T_COTS_ORD on t_open() or t_getinfo().

RETURN VALUES  t_rcvrel() returns 0 on success. On failure t_rcvrel() returns −1, t_errno is set to indicate the error, and possibly errno is set.

ERRORS  On failure, t_errno will be set to one of the following:
         TBADF       The specified file descriptor does not refer to a transport endpoint.
         TLOOK      An asynchronous event has occurred on this transport endpoint and requires immediate attention.
         TNOREL     No orderly release indication currently exists on the specified transport endpoint.
         TNOTSUPPORT This function is not supported by the underlying transport provider.
         TSYSERR    A system error has occurred during execution of this function, errno will be set to the specific error.

SEE ALSO  t_open(3N), t_sndrel(3N)

Transport Interfaces Programming Guide

NOTES  This interface is safe in multithreaded applications.

3N-1120 modified 22 Jan 1993
NAME  
t_rcvudata – receive a data unit

SYNOPSIS  
cc [ flag ... ] file ... -lnsl [ library ... ]
#include <tiuser.h>
int t_rcvudata(int fildes, struct t_unitdata *unitdata, int *flags);

MT-LEVEL  
MT-Safe

DESCRIPTION  
This function is used in connectionless mode to receive a data unit from another trans-
port user. fildes 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;

The maxlen (see netbuf in t_connect(3N)) field of addr, opt, and udata must be set before
issuing this function to indicate the maximum size of the buffer for each.

On return from this call, addr specifies the protocol address of the sending user, opt
identifies protocol-specific options that were associated with this data unit, and udata specifies the user data that was received.

By default, t_rcvudata() operates in synchronous mode and will wait for a data unit to
arrive if none is currently available. However, if O_NDELAY or O_NONBLOCK is set
(using t_open(3N) or fcntl(2)), t_rcvudata() will execute in asynchronous mode and will
fail if no data units are available.

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 t_rcvudata() call(s) will return zero for the length of the address and options until
the full data unit has been received.

RETURN VALUES  
t_rcvudata() returns 0 on successful completion. On failure t_rcvudata() returns −1,
t_errno is set to indicate the error, and possibly errno is set.

ERRORS  
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 is greater than zero but not sufficient to store the informa-
tion. The unit data information to be returned in unitdata will
be discarded.

TLOOK  
An asynchronous event has occurred on this transport endpoint
and requires immediate attention.
TNODATA  O_NDELAY or O_NONBLOCK was set, but no data units are currently available from the transport provider.
TNOTSUPPORT  This function is not supported by the underlying transport provider.
TSYSERR  A system error has occurred during execution of this function, \texttt{errno} will be set to the specific error.

SEE ALSO  fcntl(2), t_connect(3N), t_rcvuderr(3N), t_sndudata(3N)

Transport Interfaces Programming Guide

NOTES  This interface is safe in multithreaded applications.
NAME  
t_rcvuderr – receive a unit data error indication

SYNOPSIS  
cc [ flag ... ] file ... -lnsl [ library ... ]
#include <tiuser.h>
in t_rcvuderr(int fildes, struct t_uderr *uderr);

MT-LEVEL  
MT-Safe

DESCRIPTION  
This function is used in connectionless mode to receive information concerning an error on a previously sent data unit, and should be issued only after a unit data error indication. It informs the transport user that a data unit with a specific destination address and protocol options produced an error. fildes 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:

    struct netbuf addr;
    struct netbuf opt;
    long error;

netbuf is described in t_connect(3N). The maxlen (see netbuf in t_connect(3N)) field of addr and opt must be set before issuing this function to indicate the maximum size of the buffer for each.

On return from this call, the addr structure specifies the destination protocol address of the erroneous data unit, the opt structure identifies protocol-specific options that were associated with the data unit, and error specifies a protocol-dependent error code.

If the user does not care to identify the data unit that produced an error, uderr may be set to NULL and t_rcvuderr() will simply clear the error indication without reporting any information to the user.

RETURN VALUES  
t_rcvuderr() returns 0 on successful completion. On failure t_rcvuderr() returns −1, t_errno is set to indicate the error, and possibly errno is set.

ERRORS  
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 is 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.

TSYSERR  
A system error has occurred during execution of this function, errno will be set to the specific error.
SEE ALSO

$\texttt{t\_connect(3N)}, \texttt{t\_rcvudata(3N)}, \texttt{t\_sndudata(3N)}$

*Transport Interfaces Programming Guide*

NOTES

This interface is safe in multithreaded applications.
NAME  
t_snd – send data or expedited data over a connection

SYNOPSIS  
cc [ flag ...] file ... -lnsl [ library ...]
#include <tiuser.h>
int t_snd(int fildes, char *buf, unsigned nbytes, int flags);

MT-LEVEL  
MT-Safe

DESCRIPTION  
This function is used to send either normal or expedited data. fildes identifies the local transport endpoint over which data should be sent, buf points to the user data, nbytes specifies the number of bytes of user data to be sent, and flags specifies any optional flags described below.

By default, t_snd() operates in synchronous mode and may wait if flow control restrictions prevent the data from being accepted by the local transport provider at the time the call is made. However, if O_NDELAY or O_NONBLOCK is set (using t_open(3N) or fcntl(2)), t_snd() will execute in asynchronous mode, and will fail immediately if there are flow control restrictions.

Even when there are no flow control restrictions, t_snd() will wait if STREAMS internal resources are not available, regardless of the state of O_NDELAY or O_NONBLOCK.

On successful completion, t_snd() returns the number of bytes accepted by the transport provider. Normally this will equal the number of bytes specified in nbytes. However, if O_NDELAY or O_NONBLOCK is set, it is possible that only part of the data will be accepted by the transport provider. In this case, t_snd() will set T_MORE for the data that was sent (see below) and will return a value less than nbytes. If nbytes is zero and sending of zero bytes is not supported by the underlying transport provider, t_snd() will return −1 with t_errno set to TBADDATA. A return value of zero indicates that the request to send a zero-length data message was sent to the provider.

If T_EXPEDITED is set in flags, the data will be sent as expedited data, and will be subject to the interpretations of the transport provider.

If T_MORE is set in flags, or is set as described above, an indication is sent to the transport provider that the transport service data unit (TSDU) or expedited transport service data unit (ETSDU) is being sent through multiple t_snd() calls. Each t_snd() with the T_MORE flag set indicates that another t_snd() will follow with more data for the current TSDU. The end of the TSDU (or ETSDU) is identified by a t_snd() call with the T_MORE flag not set. Use of T_MORE enables a user to break up large logical data units without losing the boundaries of those units at the other end of the connection. The flag implies nothing about how the data is packaged for transfer below the transport interface. If the transport provider does not support the concept of a TSDU as indicated in the info argument on return from t_open(3N) or t_getinfo(3N), the T_MORE flag is not meaningful and should be ignored.

modified 22 Jan 1993
The size of each TSDU or ETSDU must not exceed the limits of the transport provider as returned by `t_open(3N)` or `t_getinfo(3N)`. If the size is exceeded, a `TSYSERR` with system error `EPROTO` will occur. However, the `t_snd()` may not fail because `EPROTO` errors may not be reported immediately. In this case, a subsequent call that accesses the transport endpoint will fail with the associated `TSYSERR`.

If `t_snd()` is issued from the `T_IDLE` state, the provider may silently discard the data. If `t_snd()` is issued from any state other than `T_DATAFER`, `T_INREL`, or `T_IDLE`, the provider will generate a `TSYSERR` with system error `EPROTO` (which may be reported in the manner described above).

**RETURN VALUES**

On successful completion, `t_snd()` returns the number of bytes accepted by the transport provider. On failure `t_snd()` returns `-1`, `t_errno` is set to indicate the error, and possibly `errno` is set.

**ERRORS**

On failure, `t_errno` will be set to one of the following:

- **TBADDATA**
  - `nbytes` is zero and sending zero bytes is not supported by the transport provider.

- **TBADF**
  - The specified file descriptor does not refer to a transport endpoint.

- **TFLOW**
  - `O_NDELAY` or `O_NONBLOCK` was set, but the flow control mechanism prevented the transport provider from accepting data at this time.

- **TNOTSUPPORT**
  - This function is not supported by the underlying transport provider.

- **TSYSERR**
  - A system error (see `intro(2)`) has been detected during execution of this function.

**SEE ALSO**

`fcntl(2)`, `t_getinfo(3N)`, `t_open(3N)`, `t_rcv(3N)`

*Transport Interfaces Programming Guide*

**NOTES**

This interface is safe in multithreaded applications.
NAME  t_snddis – send user-initiated disconnect request

SYNOPSIS  
cc [ flag ... ] file ... -lnsl [ library ... ]
#include <tiuser.h>
int t_snddis(int fildes, struct t_call *call);

MT-LEVEL  MT-Safe

DESCRIPTION  This function is used to initiate an abortive release on an already established connection or to reject a connect request. fildes identifies the local transport endpoint of the connection, and call specifies information associated with the abortive release. call points to a t_call structure that 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 connect request, call must be non-NULL and contain a valid value of sequence to identify uniquely the rejected connect indication to the transport provider. The addr and opt fields of call are ignored. In all other cases, call need only be used when data is being sent with the disconnect request. The addr, opt, and sequence fields of the t_call structure are ignored. If the user does not wish to send data to the remote user, the value of call may be NULL.

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.

RETURN VALUES  t_snddis() returns 0 on success. On failure t_snddis() returns -1, t_errno is set to indicate the error, and possibly errno is set.

ERRORS  On failure, t_errno will be set to one of the following:

- TBADDATA  The amount of user data specified was not within the bounds allowed by the transport provider. The transport provider’s outgoing queue will be flushed, so data may be lost.
- TBADF   The specified file descriptor does not refer to a transport endpoint.
- TBADSEQ  An invalid sequence number was specified, or a NULL call structure was specified when rejecting a connect request. The transport provider’s outgoing queue will be flushed, so data may be lost.
- TLOOK   An asynchronous event has occurred on this transport endpoint and requires immediate attention.

modified 27 Jan 1994
<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TNOTSUPPORT</td>
<td>This function is not supported by the underlying transport provider.</td>
</tr>
<tr>
<td>TOUTSTATE</td>
<td>The function was issued in the wrong sequence. The transport provider’s outgoing queue may be flushed, so data may be lost.</td>
</tr>
<tr>
<td>TSYSERR</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>

**SEE ALSO**

`t_connect(3N)`, `t_getinfo(3N)`, `t_listen(3N)`, `t_open(3N)`

*Transport Interfaces Programming Guide*

**NOTES**

This interface is safe in multithreaded applications.
NAME  
t_sndrel – initiate an orderly release

SYNOPSIS  
c{ flag ... } file ... -lnsl [ library ... ]  
#include <tiuser.h>  
int t_sndrel(int fildes);

MT-LEVEL  
MT-Safe

DESCRIPTION  
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. fildes 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() or t_getinfo().  
If t_sndrel() is issued from an invalid state, the provider will generate an EPROTO proto-  

col error; however, this error may not occur until a subsequent reference to the transport  
endpoint.

RETURN VALUES  
t_sndrel() returns 0 on success. On failure t_sndrel() returns -1, t_errno is set to indicate  
the error, and possibly errno is set.  
TSYSERR – A system error has occurred during execution of this function, errno will be  
set to the specific error.

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.  
TFLOW O_NDELAY or O_NONBLOCK was set, but the flow control  
mechanism prevented the transport provider from accepting the  
function at this time.  
TNOTSUPPORT This function is not supported by the underlying transport pro-  
vider.

SEE ALSO  
t_open(3N), t_rcvrel(3N)  
Transport Interfaces Programming Guide

NOTES  
This interface is safe in multithreaded applications.

modified 22 Jan 1993
t_sndudata (3N)          Network Functions          SunOS 5.5

NAME  t_sndudata – send a data unit

SYNOPSIS  cc [ flag ... ] file ... -Insi [ library ... ]
#include <tiuser.h>
int t_sndudata(int fildes, struct t_unitdata *unitdata);

MT-LEVEL  MT-Safe

DESCRIPTION  This function is used in connectionless mode to send a data unit to another transport
user. fildes 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 protocol-specific options that the user wants associ-
ated with this request, and udata specifies the user data to be sent. The user may choose
not to specify what protocol options are associated with the transfer by setting the len
field of opt to zero. In this case, the provider may use default options.

If the len field of udata is zero, and the sending of zero bytes is not supported by the
underlying transport provider, t_sndudata will return −1 with t_errno set to TBADDATA.

By default, t_sndudata() operates in synchronous mode and may wait if flow control restric-
tions prevent the data from being accepted by the local transport provider at the time
the call is made. However, if O_NDELAY or O_NONBLOCK is set (using t_open(3N) or
fcntl(2)), t_sndudata() will execute in asynchronous mode and will fail under such con-
ditions.

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(3N) or
t_getinfo(3N), the provider will generate an EPROTO protocol error. (See TSYSERR
below.) If the state is invalid, this error may not occur until a subsequent reference is
made to the transport endpoint.

RETURN VALUES  t_sndudata() returns 0 on successful completion. On failure t_sndudata() returns −1,
t_errno is set to indicate the error, and possibly errno is set.

ERRORS  On failure, t_errno will be set to one of the following:

TBADDATA  nbytes is zero and sending zero bytes is not supported by the trans-
port provider.

TBADF  The specified file descriptor does not refer to a transport endpoint.

TFLOW]  O_NDELAY or O_NONBLOCK was set, but the flow control mechanism prevented the transport provider from accepting data at this
time.

3N-1130  modified 27 Jan 1994
TNOTSUPP  This function is not supported by the underlying transport provider.

TSYSERR  A system error has occurred during execution of this function, errno will be set to the specific error.

SEE ALSO  fcntl(2), t_connect(3N), t_getinfo(3N), t_rcvudata(3N), t_rcvuderr(3N)

Transport Interfaces Programming Guide

NOTES  This interface is safe in multithreaded applications.

modified 27 Jan 1994
NAME

t_strerror – get error message string

SYNOPSIS

cc [ flag ...] file ... -lnsl [ library ...]
#include <tiuser.h>
char *t_strerror(int errnum);

MT-LEVEL

Unsafe

DESCRIPTION

The t_strerror() function maps the error number in errnum that corresponds to a TLI error to a language-independent error message string and returns a pointer to that string. The string pointed to will not be modified by the program, but may be overwritten by a subsequent call to the t_strerror() function. The string is not terminated by a newline character. The language for the error message strings written by t_strerror() is implementation-defined. If an error code is unknown and the language is English, t_strerror() returns the string:

"<error>: error unknown"

where <error> is the error number supplied as input. In other languages, an equivalent text is provided.

SEE ALSO

ggettext(3I), perror(3C), setlocale(3C), strerror(3C), t_error(3N)

VALID STATES

All – apart from T_UNINIT.

NOTES

If the application is linked with -lintl, then messages returned from this function are in the native language specified by the LC_MESSAGES locale category; see setlocale(3C).
NAME  t_sync – synchronize transport library

SYNOPSIS  cc [ flag ...] file ... -lnsl [ library ... ]
#include <tiuser.h>
int t_sync(int fildes);

MT-LEVEL  MT-Safe

DESCRIPTION  For the transport endpoint specified by fildes, 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 a raw file descriptor (obtained using open(2), dup(2), or
as a result of a fork(2) and exec(2)) to an initialized transport endpoint, assuming that file
descriptor referenced a transport provider. This function also allows two cooperating
processes to synchronize their interaction with a transport provider.

For example, if a process fork(2)S a new process and issues an exec(2), the new process
must issue a t_sync() to build the private library data structure associated with a tran-
sport endpoint and to synchronize the data structure with the relevant provider inform-
ation.

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 provider. t_sync() returns
the current state of the provider to the user, thereby enabling the user to verify the state
before taking further action. This coordination is only valid among cooperating
processes; it is possible that a process or an incoming event could change the provider's
state after a t_sync() is issued.

If the provider is undergoing a state transition when t_sync() is called, the function will
fail.

RETURN VALUES  t_sync() returns the state of the transport provider on successful completion. t_sync()
returns -1 on failure, t_errno is set to indicate the error, and possibly errno is set. The
state returned may be one of the following:

T_UNBND  unbound
T_IDLE  idle
T_OUTCON  outgoing connection pending
T_INCON  incoming connection pending
T_DATAXFER  data transfer
T_OUTREL  outgoing orderly release (waiting for an orderly release indication)
T_INREL  incoming orderly release (waiting for an orderly release request)
t_sync (3N)  Network Functions  SunOS 5.5

ERRORS
On failure, t_errno will be set to one of the following:
TBADF  The specified file descriptor does not refer to a transport endpoint.
TSTATECHNG  The transport provider is undergoing a state change.
TSYSERR  A system error has occurred during execution of this function,
          errno will be set to the specific error.

SEE ALSO  dup(2), exec(2), fork(2), open(2)

Transport Interfaces Programming Guide

NOTES
This interface is safe in multithreaded applications.
NAME  

t_unbind – disable a transport endpoint

SYNOPSIS  

cc [ flag ...] file ... -lnsl [ library ...]
#include <tiuser.h>
int t_unbind(int fildes);

MT-LEVEL  

MT-Safe

DESCRIPTION  

The t_unbind() function disables the transport endpoint specified by fildes which was previously bound by t_bind(3N). On completion of this call, no further data or events destined for this transport endpoint will be accepted by the transport provider.

RETURN VALUES  

t_unbind() returns 0 on success. On failure t_unbind() returns -1, t_errno is set to indicate the error, and possibly errno is set.

ERRORS  

On failure, t_errno will be set to one of the following:

TBADF  The specified file descriptor does not refer to a transport endpoint.
TLOOK  An asynchronous event has occurred on this transport endpoint.
TOUTSTATE  The function was issued in the wrong sequence.
TSYSERR  A system error has occurred during execution of this function,errno will be set to the specific error.

SEE ALSO  

T_bind(3N)

Transport Interfaces Programming Guide

NOTES  

This interface is safe in multithreaded applications.

modified 22 Jan 1993 3N-1135
NAME
tcsetpgrp – set foreground process group ID of terminal

SYNOPSIS
#include <unistd.h>
int tcsetpgrp(int fildes, pid_t pgid);

MT-LEVEL
MT-Safe

DESCRIPTION
tcsetpgrp() sets the foreground process group ID of the terminal specified by fildes to pgid. 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 must match a process group ID of a process in the same session as the calling process.

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

ERRORS
tcsetpgrp() fails if one or more of the following is true:
EBADF The fildes argument is not a valid file descriptor.
EINVAL The fildes argument is a terminal that does not support tcsetpgrp(), or pgid is not a valid process group ID.
ENOTTY The calling process does not have a controlling terminal, or the file is not the controlling terminal, or the controlling terminal is no longer associated with the session of the calling process.
EPERM pgid does not match the process group ID of an existing process in the same session as the calling process.

SEE ALSO
termio(7I)
NAME
termios, tcgetattr, tcsetattr, tcsendbreak, tcdrain, tcflush, tcflow, cfgetospeed, cfgetispeed, cfsetispeed, cfgetospeed, tcgetpgrp, tcsetpgrp, tcgetsid – 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);

MT-LEVEL
MT-Safe
tcgetattr(), tcsetattr(), tcsendbreak(), tcdrain(), tcflush(), tcflow(), cfgetospeed(), cfgetispeed(), cfsetispeed(), cfgetospeed(), tcgetpgrp(), and tcsetpgrp() are Async-Signal-Safe

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 here is the preferred user interface.

Many of the functions described here have a termios_p argument that is a pointer to a termios structure.
mios structure. This structure contains the following members:

- tcflag_t c_iflag; /* input modes */
- tcflag_t c_oflag; /* output modes */
- tcflag_t c_cflag; /* control modes */
- tcflag_t c_lflag; /* local modes */
- cc_t c_cc[NCCS]; /* control chars */

These structure members are described in detail in termio(7I).

**Get and Set Terminal Attributes**

The tcgetattr() function gets the parameters associated with the object referred by fildes and stores them in the mios structure referenced by termios_p. This function may be invoked from a background process; however, the terminal attributes may be subsequently changed by a foreground process.

The tcsetattr() function sets the parameters associated with the terminal (unless support is required from the underlying hardware that is not available) from the mios structure referenced by termios_p as follows:

- If optional_actions is TCSANOW, the change occurs immediately.
- If optional_actions is TCSADRAIN, the change occurs after all output written to fildes has been transmitted. This function should be used when changing parameters that affect output.
- If optional_actions is TCSAFLUSH, the change occurs after all output written to the object referred by fildes has been transmitted, and all input that has been received but not read is discarded before the change is made.

The symbolic constants for the values of optional_actions are defined in <termios.h>.

**Line Control**

If the terminal is using asynchronous serial data transmission, the tcsendbreak() function causes transmission of a continuous stream of zero-valued bits for a specific duration. If duration is zero, it causes transmission of zero-valued bits for at least 0.25 seconds, and not more than 0.5 seconds. If duration is not zero, it behaves in a way similar to tcdrain().

If the terminal is not using asynchronous serial data transmission, the tcsendbreak() function sends data to generate a break condition or returns without taking any action.

The tcdrain() function waits until all output written to the object referred to by fildes has been transmitted.

The tcflush() function discards data written to the object referred to by fildes 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 TCOFLUSH, it flushes both data received but not read, and data written but not transmitted.

3-1138 modified 2 Mar 1995
The `tcflow()` function suspends transmission or reception of data on the object referred to by `fildes`, depending on the value of `action`:

- If `action` is `TCOFF`, it suspends output.
- If `action` is `TCOON`, it restarts suspended output.
- If `action` is `TCIOFF`, the system transmits a STOP character, which causes the terminal device to stop transmitting data to the system.
- If `action` is `TCION`, the system transmits a START character, which causes the terminal device to start transmitting data to the system.

**Get and Set Baud Rate**

The baud rate functions get and set the values of the input and output baud rates in the `termios` structure. The effects on the terminal device described below do not become effective until the `tcsetattr()` function is successfully called.

The input and output baud rates are stored in the `termios` structure. The values shown in the table are supported. The names in this table are defined in `<termios.h>`.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>B0</td>
<td>Hang up</td>
<td>B4800</td>
<td>4800 baud</td>
</tr>
<tr>
<td>B50</td>
<td>50 baud</td>
<td>B9600</td>
<td>9600 baud</td>
</tr>
<tr>
<td>B75</td>
<td>75 baud</td>
<td>B19200</td>
<td>19200 baud</td>
</tr>
<tr>
<td>B110</td>
<td>110 baud</td>
<td>B38400</td>
<td>38400 baud</td>
</tr>
<tr>
<td>B134</td>
<td>134.5 baud</td>
<td>B57600</td>
<td>57600 baud</td>
</tr>
<tr>
<td>B150</td>
<td>150 baud</td>
<td>B76800</td>
<td>76800 baud</td>
</tr>
<tr>
<td>B200</td>
<td>200 baud</td>
<td>B115200</td>
<td>115200 baud</td>
</tr>
<tr>
<td>B300</td>
<td>300 baud</td>
<td>B153600</td>
<td>153600 baud</td>
</tr>
<tr>
<td>B600</td>
<td>600 baud</td>
<td>B230400</td>
<td>230400 baud</td>
</tr>
<tr>
<td>B1200</td>
<td>1200 baud</td>
<td>B307200</td>
<td>307200 baud</td>
</tr>
<tr>
<td>B1800</td>
<td>1800 baud</td>
<td>B460800</td>
<td>460800 baud</td>
</tr>
<tr>
<td>B24000</td>
<td>24000 baud</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

`cfgetospeed()` returns the output baud rate stored in the `termios` structure pointed to by `termios_p`.

`cfsetospeed()` sets the output baud rate stored in the `termios` structure pointed to by `termios_p` to `speed`. The zero baud rate, `B0`, is used to terminate the connection. If `B0` is specified, the modem control lines are no longer be asserted. Normally, this disconnects the line.

`cfgetispeed()` returns the input baud rate stored in the `termios` structure pointed to by `termios_p`.

`cfsetispeed()` sets the input baud rate stored in the `termios` structure pointed to by `termios_p` to `speed`. If the input baud rate is set to zero, the input baud rate is specified by the value of the output baud rate. Both `cfsetispeed()` and `cfsetospeed()` return a value of zero if successful and −1 to indicate an error. This refers both to changes to baud rates not supported by the hardware, and to changes setting the input and output baud rates to

modified 2 Mar 1995
different values if the hardware does not support this.

**Get and Set Terminal Foreground Process Group ID**

tcsetpgrp() sets the foreground process group ID of the terminal specified by fildes to pgid. 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. pgid must match a process group ID of a process in the same session as the calling process.

tcsetpgrp() 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.

**tcgetpgrp()** returns the foreground process group ID of the terminal specified by fildes. tcgetpgrp() is allowed from a process that is a member of a background process group; however, the information may be subsequently changed by a process that is a member of a foreground process group.

**Get Terminal Session ID**
tcgetsid() returns the session ID of the terminal specified by fildes.

**RETURN VALUES**

On success, tcgetpgrp() returns the process group ID of the foreground process group associated with the specified terminal. Otherwise, it returns −1 and sets errno to indicate the error.

On success, tcgetsid() returns the session ID associated with the specified terminal. Otherwise, it returns −1 and sets errno to indicate the error.

On success, all other functions return a value of 0. Otherwise, they return −1 and set errno to indicate the error.

**ERRORS**

All of the functions fail if one of more of the following is true:

- **EBADF** The fildes argument is not a valid file descriptor.
- **ENOTTY** The file associated with fildes is not a terminal.

**tcsetattr()** also fails if the following is true:

- **EINVAL** The optional_actions argument is not a proper value, or an attempt was made to change an attribute represented in the termios structure to an unsupported value.

**tcsendbreak()** also fails if the following is true:

- **EINVAL** The device does not support the tcsendbreak() function.

**tcdrain()** also fails if one or more of the following is true:

- **EINVAL** The device does not support the tcdrain() function.
- **EINTR** A signal interrupted the tcdrain() function.

**tcflush()** also fails if the following is true:

- **EINVAL** The device does not support the tcflush() function or the queue_selector argument is not a proper value.

**tcflow()** also fails if the following is true:

- **EINVAL** The device does not support the tcflow() function or the action argument is not a proper value.
tcgetpg() also fails if the following is true:
ENOTTY the calling process does not have a controlling terminal, or fildes does not refer to the controlling terminal.

tcsetpg() also fails if the following is true:
EINVAL  pgid is not a valid process group ID.
ENOTTY  the calling process does not have a controlling terminal, or fildes does not refer to the controlling terminal, or the controlling terminal is no longer associated with the session of the calling process.
EPERM   pgid does not match the process group of an existing process in the same session as the calling process.

tcgetsid() also fails if the following is true:
EACCES  fildes is a terminal that is not allocated to a session.

SEE ALSO  setpgid(2), getsid(2), termio(7I)
NAME  thr_main – identify the main thread

SYNOPSIS  cc [ flag ... ] file ... -lthread [ library ... ]
#include <thread.h>
int thr_main(void)

MT-LEVEL  MT-Safe

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

SEE ALSO  thr_self(3T)
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);

MT-LEVEL
MT-Safe

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:

```c
#include <stdio.h>
#include <thread.h>
main() {
    printf("thr_min_stack() returns %u\n",thr_min_stack());
}
```

SEE ALSO
pthread_attr_init(3T), pthread_create(3T)

NOTES
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:

modified 22 Sep 1994
/* cc thisfile.c -lpthread */
#define _REENTRANT
#include <pthread.h>
#include <stdio.h>

main() {
    printf("minimum POSIX stack size is %u\n", PTHREAD_STACK_MIN);
}
The Solaris threads set/get concurrency functionality described on this man page are not implemented in the POSIX threads interface.
NAME  thr_stksegment – get thread stack bottom and stack size

SYNOPSIS  cc [ flag ... ] file ... -ltthread [ library ... ]
           #include <thread.h>
           #include <sys/signal.h>
           int thr_stksegment(stack_t *)

MT-LEVEL  MT-Safe

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.

SEE ALSO  thr_create(3T)
NAME  thr_suspend, thr_continue – suspend or continue thread execution

SYNOPSIS  

```c
#include <thread.h>

int thr_suspend(thread_t target_thread);
int thr_continue(thread_t target_thread);
```

MT-LEVEL  MT-Safe

DESCRIPTION  thr_suspend() 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.

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

ERRORS  If any of the following conditions are detected, thr_suspend() or thr_continue() fails and returns the corresponding value:

ESRCH  target_thread cannot be found in the current process.

SEE ALSO  thr_create(3T)

NOTES  The are no POSIX counterparts to the Solaris threads suspend and continue functionality described on this man page.
NAME  

thr_yield – thread yield to another thread

SYNOPSIS  

cc [ flag ... ] file ... -lthread [ library ... ]
#include <thread.h>
void thr_yield(void);

MT-LEVEL  

MT-Safe

DESCRIPTION  

thr_yield() causes the current thread to yield its execution in favor of another thread with the same or greater priority.

RETURN VALUES  

thr_yield() returns nothing and does not set errno.

SEE ALSO  

sched_yield(3R), thr_setprio(3T)

NOTES  

Although there is no POSIX “pthreads” counterpart to thr_yield(), the POSIX real-time function, sched_yield(3R) does correspond to thr_yield(); thereby affording the POSIX portability to this functionality.
NAME
threads, pthreads, libpthread, libthread – thread libraries: libpthread and libthread

SYNOPSIS
POSIX
cc [ flag ... ] file ... -lpthread [ -lposix4 library ... ]
#include <pthread.h>

Solaris
cc [ flag ... ] file ... -lthread [ library ... ]
#include <thread.h>
#include <sched.h>

MT-LEVEL
Fork1-Safe MT-Safe

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.

### IMPLEMENTATION

<table>
<thead>
<tr>
<th>POSIX</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creation</td>
<td>pthread_create()</td>
</tr>
<tr>
<td></td>
<td>pthread_attr_init()</td>
</tr>
<tr>
<td></td>
<td>pthread_attr_setdetachstate()</td>
</tr>
<tr>
<td></td>
<td>pthread_attr_getdetachstate()</td>
</tr>
<tr>
<td></td>
<td>pthread_attr_setinheritsched()</td>
</tr>
<tr>
<td></td>
<td>pthread_attr_getinheritsched()</td>
</tr>
<tr>
<td></td>
<td>pthread_attr_setschedparam()</td>
</tr>
<tr>
<td></td>
<td>pthread_attr_getschedparam()</td>
</tr>
<tr>
<td></td>
<td>pthread_attr_setschedpolicy()</td>
</tr>
<tr>
<td></td>
<td>pthread_attr_getschedpolicy()</td>
</tr>
<tr>
<td></td>
<td>pthread_attr_setscope()</td>
</tr>
<tr>
<td></td>
<td>pthread_attr_getscope()</td>
</tr>
<tr>
<td></td>
<td>pthread_attr_setstackaddr()</td>
</tr>
<tr>
<td></td>
<td>pthread_attr_getstackaddr()</td>
</tr>
<tr>
<td></td>
<td>pthread_attr_setstacksize()</td>
</tr>
<tr>
<td></td>
<td>pthread_attr_getstacksize()</td>
</tr>
<tr>
<td>Exit</td>
<td>pthread_exit()</td>
</tr>
<tr>
<td></td>
<td>pthread_join()</td>
</tr>
<tr>
<td></td>
<td>pthread_detach()</td>
</tr>
<tr>
<td>Thread Specific Data</td>
<td>pthread_key_create()</td>
</tr>
<tr>
<td></td>
<td>pthread_setspecific()</td>
</tr>
<tr>
<td></td>
<td>pthread_getspecific()</td>
</tr>
<tr>
<td></td>
<td>pthread_key_delete()</td>
</tr>
<tr>
<td>Signal</td>
<td>pthread_sigmask()</td>
</tr>
<tr>
<td></td>
<td>pthread_kill()</td>
</tr>
<tr>
<td>ID</td>
<td>pthread_self()</td>
</tr>
<tr>
<td></td>
<td>pthread_equal()</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Scheduling</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>pthread_setschedparam()</td>
</tr>
<tr>
<td></td>
<td>pthread_getschedparam()</td>
</tr>
</tbody>
</table>

3T-1150 modified 28 Oct 1994
<table>
<thead>
<tr>
<th>Cancellation</th>
<th>pthread_cancel()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>pthread_setcancelstate()</td>
</tr>
<tr>
<td></td>
<td>pthread_setcanceltype()</td>
</tr>
<tr>
<td></td>
<td>pthread_testcancel()</td>
</tr>
<tr>
<td></td>
<td>pthread_cleanup_pop()</td>
</tr>
<tr>
<td></td>
<td>pthread_cleanup_push()</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mutex</th>
<th>pthread_mutex_init()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>pthread_mutexattr_init()</td>
</tr>
<tr>
<td></td>
<td>pthread_mutexattr_setpshared()</td>
</tr>
<tr>
<td></td>
<td>pthread_mutexattr_getpshared()</td>
</tr>
<tr>
<td></td>
<td>pthread_mutexattr_setprotocol()</td>
</tr>
<tr>
<td></td>
<td>pthread_mutexattr_getprotocol()</td>
</tr>
<tr>
<td></td>
<td>pthread_mutexattr_setprioceiling()</td>
</tr>
<tr>
<td></td>
<td>pthread_mutexattr_getprioceiling()</td>
</tr>
<tr>
<td></td>
<td>pthread_mutexattr_destroy()</td>
</tr>
<tr>
<td></td>
<td>pthread_mutex_lock()</td>
</tr>
<tr>
<td></td>
<td>pthread_mutex_trylock()</td>
</tr>
<tr>
<td></td>
<td>pthread_mutex_unlock()</td>
</tr>
<tr>
<td></td>
<td>pthread_mutex_destroy()</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Condition Variable</th>
<th>pthread_cond_init()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>pthread_condattr_init()</td>
</tr>
<tr>
<td></td>
<td>pthread_condattr_setpshared()</td>
</tr>
<tr>
<td></td>
<td>pthread_condattr_getpshared()</td>
</tr>
<tr>
<td></td>
<td>pthread_condattr_destroy()</td>
</tr>
<tr>
<td></td>
<td>pthread_cond_wait()</td>
</tr>
<tr>
<td></td>
<td>pthread_cond_timedwait()</td>
</tr>
<tr>
<td></td>
<td>pthread_cond_signal()</td>
</tr>
<tr>
<td></td>
<td>pthread_cond_broadcast()</td>
</tr>
<tr>
<td></td>
<td>pthread_cond_destroy()</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Reader/Writer Locking</th>
<th>rwlock_init()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>rw_rdlock()</td>
</tr>
<tr>
<td></td>
<td>rw_tryrdlock()</td>
</tr>
<tr>
<td></td>
<td>rw_wrlock()</td>
</tr>
<tr>
<td></td>
<td>rw_trywrlock()</td>
</tr>
<tr>
<td></td>
<td>rw_unlock()</td>
</tr>
<tr>
<td></td>
<td>rwlock_destroy()</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Semaphore</th>
<th>sem_init()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>sem_open()</td>
</tr>
<tr>
<td></td>
<td>sem_close()</td>
</tr>
<tr>
<td></td>
<td>sem_wait()</td>
</tr>
</tbody>
</table>

modified 28 Oct 1994
Synchronization

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().

FILES

POSIX

/usr/include/pthread.h
/lib/libpthread.*
/lib/libposix4.*

Solaris

/usr/include/thread.h
/usr/include/sched.h
/lib/libthread.*

3T-1152 modified 28 Oct 1994
<table>
<thead>
<tr>
<th>SEE ALSO</th>
<th>fork(2), intro(3), pthread_atfork(3T), pthread_create(3T)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERRORS</td>
<td>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.</td>
</tr>
</tbody>
</table>
NAME  
timer_create – create a timer

SYNOPSIS  
cc [ flag ...] file ... -lposix4 [ library ... ]
#include <signal.h>
#include <time.h>

int timer_create(clockid_t clock_id, struct sigevent *evp, timer_t *timerid);

struct sigevent {
  int sigev_notify; /* notification type */
  int sigev_signo;  /* signal number */
  union sigval sigev_value; /* signal value */
};
union sigval {
  int sival_int; /* integer value */
  void *sival_ptr; /* pointer value */
};

MT-LEVEL  
MT-Safe with exceptions

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_TIMERS 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.

3R-1154  modified 12 Aug 1993
timer_create( ) returns 0 upon success and creates a timer_t, timerid, which can be passed to the timer calls; otherwise it returns -1 and sets errno to indicate the error condition.

ERRORS

EAGAIN  The system lacks sufficient signal queuing resources to honor the request.
         The calling process has already created all of the timers it is allowed by this implementation.
EINVAL  The specified clock ID, clock_id, is not defined.
ENOSYS  timer_create( ) is not supported by this implementation.

SEE ALSO  exec(2), fork(2), time(2), clock_settime(3R), timer_delete(3R), timer_settime(3R)

NOTES  Timers are not inherited by a child process across a fork(2) and can be disarmed and deleted by an exec(2).
       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);

MT-LEVEL MT-Safe with exceptions

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.

SEE ALSO timer_create(3R)
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 */
};

MT-LEVEL
Async-Signal-Safe

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 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.

modified 24 Aug 1993
3R-1157
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()`.
  
  The timer, `timerid`, had already been deleted by `timer_delete()`.

- **A value structure specified a nanosecond value less than zero or greater than or equal to 1,000,000,000.**

- **ENOSYS**: `timer_settime()`, `timer_gettime()`, or `timer_getoverrun()` is not supported by this implementation.

### SEE ALSO

`clock_settime(3R)`, `timer_create(3R)`, `timer_delete(3R)`
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 ter-
minated 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 plat-
forms. Use of these interfaces with any of the system libraries or in multi-thread applica-
tions is unsupported.

times() has been superseded by getrusage(3C).

modified 18 Apr 1994
NAME       tmpfile – create a temporary file

SYNOPSIS   #include <stdio.h>
            FILE *tmpfile(void);

MT-LEVEL    Safe

DESCRIPTION tmpfile() creates a temporary file using a name generated by the tmpnam() routine and returns a corresponding FILE pointer. If the file cannot be opened, a NULL pointer is returned. The file is automatically deleted when the process using it terminates or when the file is closed. The file is opened for update ("w+").

SEE ALSO   creat(2), open(2), unlink(2), fopen(3S), mktemp(3C), perror(3C), stdio(3S), tmpnam(3S)
NAME  tmpnam, tmpnam_r, tempnam – create a name for a temporary file

SYNOPSIS  
```
#include <stdio.h>
char *tmpnam(char *s);
char *tmpnam_r(char *s);
char *tempnam(const char *dir, const char *pfx);
```

MT-LEVEL  See the NOTES section of this page.

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.

SEE ALSO  creat(2), unlink(2), fopen(3S), free(3C), malloc(3C), mktemp(3C), tmpfile(3S)
NOTES

The `tmpnam_r()` interface is as proposed in the POSIX.4a Draft #6 document, and is subject to change to be compliant to the standard when it is accepted.

When compiling multi-thread applications, the `_REENTRANT` flag must be defined on the compile line. This flag should only be used in multi-thread applications.

These functions generate a different file name each time they are called.

Files created using these functions and either `fopen(3S)` or `creat(2)` are temporary only in the sense that they reside in a directory intended for temporary use, and their names are unique. It is the user's responsibility to remove the file when its use is ended.

If called more than `TMP_MAX` (defined in `<stdio.h>`) times in a single process, these functions start recycling previously used names.

Between the time a file name is created and the file is opened, it is possible for some other process to create a file with the same name. This can never happen if that other process is using these functions or `mktemp(3C)` and the file names are chosen to render duplication by other means unlikely.

`tempnam()` is safe in 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`.

3S-1162 modified 18 Nov 1993
NAME   tnf_process_disable, tnf_process_enable, tnf_thread_disable, tnf_thread_enable – probe control internal interface

SYNOPSIS cc [flag ...] file ... -ltnfprobe [ 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);

AVAILABILITY    SUNWtnfd

MT-LEVEL    MT-Safe

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.

SEE ALSO    prex(1), tnfdump(1), TNF_DECLARE_RECORD(3X), TNF_PROBE(3X)

NOTES    A probe is considered enabled only if:
    
    • prex(1) has enabled the probe AND
    • the process has probing enabled — which is the default or could be set via tnf_process_enable() AND
    • the thread that hits the probe has probing enabled — which is every thread’s default or could be set via tnf_thread_enable().

    There is a run time cost associated with determining that the probe is disabled. To reduce the performance effect of probes, this cost should be minimized. The quickest way that a probe can be determined to be disabled is by the enable control that prex(1) uses. Therefore, to disable all the probes in a process use the disable command in prex(1) rather than tnf_process_disable().

    tnf_process_disable() and tnf_process_enable() should only be used to toggle probing based on some internal program condition. tnf_thread_disable() should be used to turn off probing for threads that are uninteresting.

modified 29 Sep 1994
NAME trig, sin, cos, tan, asin, acos, atan, atan2 – trigonometric functions

SYNOPSIS cc [ flag ... ] file ... -lm [ library ... ]
#include <math.h>

double sin(double x);
double cos(double x);
double tan(double x);
double asin(double x);
double acos(double x);
double atan(double x);
double atan2(double y, double x);

MT-LEVEL MT-Safe

DESCRIPTION sin(x), cos(x) and tan(x) return trigonometric functions of radian arguments. Trigonometric argument reduction is carried out with respect to the infinitely precise $\pi$. asin(x) returns the arc sine of $x$ in the range $-\pi/2$ to $\pi/2$. acos(x) returns the arc cosine of $x$ in the range $0$ to $\pi$. atan(x) returns the arc tangent of $x$ in the range $-\pi/2$ to $\pi/2$. atan2(y, x) and hypot(x, y) (see hypot(3M)) convert rectangular coordinates (x, y) to polar (r, $\theta$); atan2(y, x) computes $\theta$, the argument or phase, by computing an arc tangent of $y/x$ in the range $-\pi$ to $\pi$.

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

SEE ALSO hypot(3M), matherr(3M)

DIAGNOSTICS In IEEE 754 mode (i.e. the -xlibmieee cc compilation option), these functions handle exceptional arguments in the spirit of ANSI/IEEE Std 754-1985. sin($\pm\infty$), cos($\pm\infty$) and tan($\pm\infty$) return NaN; asin(x) and acos(x) return NaN if $|x|>1$. atan($\pm\infty$) returns $\pm\pi/2$. For atan2( ),
- atan2($\pm0$, x) returns $\pm0$ for $x>0$ or $x=+0$;
- atan2($\pm0$, x) returns $\pm\pi$ for $x<0$ or $x=-0$;
- atan2(y, $\pm0$) returns $\pi/2$ for $y>0$;
- atan2(y, $\pm0$) returns $-\pi/2$ for $y<0$;
- atan2($ty$, $\infty$) returns $\pm0$ for finite $y>0$;
- atan2($\pm\infty$, x) returns $\pm\pi/2$ for finite $x$;
- atan2($ty$, $-\infty$) returns $\pm\pi$ for finite $y<0$;
- atan2($\pm\infty$, $\pm\infty$) returns $\pm\pi/4$;
- atan2($\pm\infty$, $-\infty$) returns $\pm3\pi/4$.

3M-1164 modified 4 Mar 1994
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);

MT-LEVEL    MT-Safe

DESCRIPTION    The file whose name is given by path or referenced by the descriptor fildes has its size set to length bytes.
If the file was previously longer than length, bytes past length will no longer be accessible.
If it was shorter, bytes from the EOF before the call to the EOF after the call will be read in as zeros. The effective user ID of the process must have write permission for the file, and for ftruncate() the file must be open for writing.

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    truncate() fails if one or more of the following are true:
EACCES    Search permission is denied on a component of the path prefix.
EACCES    Write permission is denied for the file referred to by path.
EFAULT    path points outside the process’s allocated address space.
EINTR    A signal was caught during execution of the truncate routine.
EINVAL    path is not an ordinary file.
EIO    An I/O error occurred while reading from or writing to the file system.
EISDIR    The file referred to by path is a directory.
ELOOP    Too many symbolic links were encountered in translating path.
EMFILE    The maximum number of file descriptors available to the process has been reached.
EMULTIHOP    Components of path require hopping to multiple remote machines and file system type does not allow it.
ENAMETOOLONG    The length of a path component exceeds NAME_MAX characters, or the length of path exceeds PATH_MAX characters.
ENFILE    Could not allocate any more space for the system file table.
ENOENT    Either a component of the path prefix or the file referred to by path does not exist.
ENOLINK    path points to a remote machine and the link to that machine is no longer active.
ENOTDIR    A component of the path prefix of path is not a directory.

modified 22 Jan 1993 3C-1165
EROFS  The file referred to by path resides on a read-only file system.

ftruncate() fails if one or more of the following are true:

EAGAIN  The file exists, mandatory file/record locking is set, and there are outstanding record locks on the file (see chmod(2)).

EBADF  fildes is not a file descriptor open for writing.

EINTR  A signal was caught during execution of the ftruncate routine.

EIO  An I/O error occurred while reading from or writing to the file system.

ENOLINK  fildes points to a remote machine and the link to that machine is no longer active.

EINVAL  fildes does not correspond to an ordinary file.

SEE ALSO chmod(2), fcntl(2), open(2)
NAME
tsearch, tfind, tdelete, twalk – manage binary search trees

SYNOPSIS
#include <search.h>

void *tsearch(const void *key, void **rootp,
               int (*compar)(const void *, const void *));

void *tfind(const void *key, void * const *rootp,
            int (*compar)(const void *, const void *));

void *tdelete(const void *key, void **rootp,
               int (*compar)(const void *, const void *));

void twalk(void *root, void(*action)(void *, VISIT, int));

MT-LEVEL
Safe

DESCRIPTION
tsearch( ), tfind( ), tdelete( ), and twalk( ) 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 com-
pare 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 vari-
able 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

modified 22 Jan 1993
3C-1167
The tree), or whether the node is a leaf. The third argument is the level of the node in the
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-
cast into type pointer-to-element.

RETURN VALUES
A NULL pointer is returned by tsearch() if there is not enough space available to create a
new node. A NULL pointer is returned by tfind() and tdelete() if rootp is NULL on entry.
If the datum is found, both tsearch() and tfind() return a pointer to it. If not, tfind()
returns NULL, and tsearch() returns a pointer to the inserted item.

EXAMPLES
The following code reads in strings and stores structures containing a pointer to each
string and a count of its length. It then walks the tree, printing out the stored strings and
their lengths in alphabetical order.

```
#include <string.h>
#include <stdio.h>
#include <search.h>

struct node {
    char *string;
    int length;
};
char string_space[10000];
struct node nodes[500];
void *root = NULL;

int node_compare(const void *node1, const void *node2) {
    return strcmp(((const struct node *) node1)->string,
                 ((const struct node *) node2)->string);
}

void print_node(void **node, VISIT order, int level) {
    if (order == preorder || order == leaf) {
        printf("length=%d, string=%20s\n",
               ((struct node **)node)->length,
               ((struct node **)node)->string);
    }
}

main()
{
    char *strptr = string_space;
    struct node *nodeptr = nodes;
    int i = 0;
```
while (getline(strptr) != NULL && i++ < 500) {
    nodeptr->string = strptr;
    nodeptr->length = strlen(strptr);
    (void) tsearch((void *)nodeptr,
                    &root, node_compare);
    strptr += nodeptr->length + 1;
    nodeptr++;
}

twalk(root, print_node);
}

SEE ALSO bsearch(3C), hsearch(3C), lsearch(3C)

NOTES The root argument to twalk() is one level of indirection less than the rootp arguments to
tsearch() and tdelete().
There are two nomenclatures used to refer to the order in which tree nodes are visited.
tsearch uses preorder, postorder and endorder to refer respectively to visiting a node
before any of its children, after its left child and before its right, and after both its chil-
dren. The alternate nomenclature uses preorder, inorder and postorder to refer to the
same visits, which could result in some confusion over the meaning of postorder.
If the calling function alters the pointer to the root, results are unpredictable.
NAME  ttyname, ttyname_r, isatty – find name of a terminal

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

char *ttyname(int fd);  
char *ttyname_r(int fd, char *buf, int len);  
int isatty(int fd);
```

POSIX  
```c
int ttyname_r(int fd, char *name, size_t namesize);
```

MT-LEVEL  See the NOTES section of this page.

DESCRIPTION  
`ttyname()` returns a pointer to a string containing the null-terminated path name of the terminal device associated with file descriptor `fd`.  
`ttyname_r()` has the same functionality as `ttyname()` except that the caller must supply a buffer `buf` with length `len` to store the result; `buf` must be at least `_POSIX_PATH_MAX` in size (defined in `<limits.h>`). The POSIX version of `ttyname_r()` takes a `namesize` parameter of type `size_t`.  
`isatty()` returns 1 if `fd` is associated with a terminal device, 0 otherwise.

RETURN VALUES  
`ttyname()` and `ttyname_r()` return a NULL pointer if `fd` does not describe a terminal device in directory `/dev`. The POSIX `ttyname_r()` returns zero if successful, or the error number upon failure.

ERRORS  
`ttyname_r()` will fail if the following is true:

```
ERANGE The size of the buffer is smaller than the result to be returned.
```

FILES  
`/dev/*`

SEE ALSO  
`gettext(3I), setlocale(3C)`

NOTES  
When compiling multi-thread applications, the `_REENTRANT` flag must be defined on the compile line. This flag should only be used in multi-thread applications.  
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.  
`ttyname()` is unsafe in multi-thread applications. `ttyname_r()` is MT-Safe, and should be used instead. `isatty()` is MT-Safe in multi-thread applications.  
The new `ttyname_r()` interface is as specified in POSIX 1003.1c Draft #10.
### NAME

ttslot – find the slot in the utmp file of the current user

### SYNOPSIS

```c
#include <stdlib.h>

int ttslot(void);
```

### MT-LEVEL

Safe

### DESCRIPTION

ttslot() returns the index of the current user’s entry in the `/var/adm/utmp` file. The returned index is accomplished by scanning files in `/dev` for the name of the terminal associated with the standard input, the standard output, or the standard error output (0, 1, or 2).

### RETURN VALUES

A value of −1 is returned if an error was encountered while searching for the terminal name or if none of the above file descriptors are associated with a terminal device.

### FILES

 `/var/adm/utmp`

### SEE ALSO

getutent(3C), ttyname(3C)
NAME ualarm – schedule signal after interval in microseconds

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

unsigned int ualarm(unsigned int usecs,
                    unsigned int interval);
```

DESCRIPTION ualarm() sends signal SIGALRM (see signal(3C)), to the caller in a number of microseconds given by the usecs argument. Unless caught or ignored, the signal terminates the caller.

If the interval argument is non-zero, the SIGALRM signal will be sent to the caller every interval microseconds after the timer expires (for instance, after usecs microseconds have passed).

Because of scheduling delays, resumption of execution when the signal is caught may be delayed an arbitrary amount.

Interactions between ualarm() and either alarm(2) or sleep(3C) are unspecified.

RETURN VALUES The return value is the amount of time previously remaining in the alarm clock.

SEE ALSO alarm(2), getitimer(2), setitimer(2), sighold(3C), signal(3C), sleep(3C), usleep(3C)

NOTES ualarm() is a simplified interface to setitimer(2); (see getitimer(2)).
NAME
ungetc – push character back onto input stream

SYNOPSIS
#include <stdio.h>
int ungetc(int c, FILE *stream);

MT-LEVEL
MT-Safe

DESCRIPTION
ungetc() 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 indica-
tor 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(3C) 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.

SEE ALSO
intro(3), fseek(3S), fsetpos(3C), getc(3S), setbuf(3S), stdio(3S)

modified 22 Jan 1993
**NAME**    ungetwc – push a Process Code character back into input stream

**SYNOPSIS**

```c
cc [ flag ... ] file ... -lw [ library ... ]
#include <stdio.h>
#include <widec.h>
int ungetwc(int c, FILE *stream);
```

**MT-LEVEL**    MT-Safe

**DESCRIPTION**

ungetwc() pushes back the Process Code character `c` onto an input stream. That character will be returned by the next `getwc(3I)` call on that stream. ungetwc() returns `c`, and leaves the file `stream` unchanged.

One character of pushback is guaranteed provided something has been read from the stream and the stream is actually buffered. In the case that `stream` is `stdin`, one character may be pushed back onto the buffer without a previous read statement.

If `c` equals EOF, ungetwc() does nothing to the buffer and returns EOF.

An `fseek(3S)` erases all memory of pushed back characters.

**RETURN VALUES**

ungetwc() returns EOF if it can’t push a character back.

**SEE ALSO**

`fseek(3S), getwc(3I), setbuf(3S), ungetc(3S)`
### NAME
unlockpt – unlock a pseudo-terminal master/slave pair

### SYNOPSIS
```c
int unlockpt(int fd);  
```

### MT-LEVEL
Safe

### DESCRIPTION
The function `unlockpt()` clears a lock flag associated with the slave pseudo-terminal device associated with its master pseudo-terminal counterpart so that the slave pseudo-terminal device can be opened. `fd` is a file descriptor returned from a successful open of a master pseudo-terminal device.

### RETURN VALUES
Upon successful completion, the function `unlockpt()` returns 0; otherwise it returns -1. A failure may occur if `fd` is not an open file descriptor or is not associated with a master pseudo-terminal device.

### SEE ALSO
- `open(2)`, `grantpt(3C)`, `ptsname(3C)`
- STREAMS Programming Guide
**NAME**
usleep – suspend execution for interval in microseconds

**SYNOPSIS**
```c
#include <unistd.h>

int usleep(unsigned int useconds);
```

**DESCRIPTION**
Suspend the caller for the number of microseconds specified by the argument. The actual suspension time may be an arbitrary amount longer because of other activity in the system, or because of the time spent in processing the call.

The routine is implemented by setting an interval timer and pausing until it occurs. The previous state of this timer is saved and restored. If the sleep time exceeds the time to the expiration of the previous timer, the process sleeps only until the signal would have occurred, and the signal is sent a short time later.

Interactions between `usleep()` and either `alarm(2)` or `sleep(3C)` are unspecified.

**SEE ALSO**
alarm(2), getitimer(2), poll(2), sigprocmask(2), select(3C), sleep(3C), ualarm(3C)

**NOTES**
A microsecond is .000001 seconds.
NAME  vlfmt – display error message in standard format and pass to logging and monitoring services

MT-LEVEL  MT-safe

SYNOPSIS  
```c
#include <stdarg.h>
#include <pfmt.h>

int vlfmt(FILE *stream, long flags, char *format, va_list ap);
```

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);  
    va_end(ap);  
    (void) abort();  
}

NOTES Since vlfmt() uses gettxt(3C), it is recommended that vlfmt() not be used.

SEE ALSO gettxt(3C), lfmt(3C), stdarg(5)
NAME
volmgt_check ∼ have Volume Management check for media

SYNOPSIS
cc [ flag . . . file . . . ] -lvolmgt [ library. . . . ]
#include <volmgt.h>
int volmgt_check(char *pathname);

MT-LEVEL
MT-Safe

DESCRIPTION
This routine asks Volume Management to check the specified pathname and determine if
ew 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.

SEE ALSO
cc(1B), volcheck(1), vold(1M), open(2), stat(2), volmgt_inuse(3X), volmgt_running(3X),
vold.conf(4), 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_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);

MT-LEVEL MT-Safe

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");
   }

SEE ALSO cc(1B), vold(1M), open(2), stat(2), errno(3C), volmgt_check(3X), volmgt_running(3X), 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 to check errno to differentiate the two, and to ensure that Volume Management is running.
NAME    volmgt_root – return the Volume Management root directory

SYNOPSIS  cc [ flag ... ] file ... -lvolmgt [ library... ]
#include <volmgt.h>
char *volmgt_root(void);

MT-LEVEL    MT-Safe

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:

```c
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

SEE ALSO  cc(1B), vold(1M), open(2), volmgt_check(3X), volmgt_inuse(3X), volmgt_running(3X), volfs(7FS)

NOTES  This routine will return the default root directory location even when Volume Management is not running.
### NAME
volmgt_running – return whether or not Volume Management is running

### SYNOPSIS
```c
cc [ flag ... ] file ... -lvolmgt [ library... ]
#include <volmgt.h>
int volmgt_running(void);
```

### MT-LEVEL
MT-Safe

### 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:
```c
if (volmgt_running() != 0) {
    (void) printf("Volume Management is running\n");
} else {
    (void) printf("Volume Management is NOT running\n");
}
```

### SEE ALSO
`cc(1B), vold(1M), open(2), stat(2), volmgt_check(3X), volmgt_inuse(3X), volfs(7FS)`

### NOTES
Volume Management must be running for many of the Volume Management library routines to work.
NAME
volmgt_symname, volmgt_symdev – convert between Volume Management symbolic
names, and the devices that correspond to them

SYNOPSIS
cc [ flag . . . ] file . . . -lvolmgt [ library. . . ]
#include <volmgt.h>
char *volmgt_symname(char *pathname);
char *volmgt_symdev(char *symname);

MT-LEVEL
MT-Safe

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 dev-

cle.

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.
EINTR 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.
EINTR 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):

for (i=0; i < 10; i++) {
    (void) sprintf(path, "%s", i);
    if (volmgt_symdev(path) != NULL) {
        (void) printf("volume %s is in drive %d\n", path, i);
    }
}

modified 30 May 1995
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\n");
} else {
    (void) printf("path managed as %s\n", nm);
}
```

SEE ALSO cc(1B), vold(1M), open(2), stat(2), free(3C), malloc(3C), volmgt_check(3X), volmgt_inuse(3X), volmgt_running(3X), volfs(7FS)

NOTES These routines only work when Volume Management is running.

BUGS There should be a straightforward way to query Volume Management for a list of all
media types it's managing, and how many of each type are being managed.
NAME
vpfmt – display error message in standard format and pass to logging and monitoring services

MT-LEVEL
MT-safe

SYNOPSIS
#include <stdarg.h>
#include <pfmt.h>

int vpfmt(FILE *stream, long flags, char *format, va_list ap);

DESCRIPTION
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.

RETURN VALUE
Upon success, lfmt() returns the number of bytes transmitted. Upon failure, it returns a negative value:
−1 write error to stream.

EXAMPLE
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(...)
{
```

modified 07 December 1993
va_list ap;
char *format;
va_start(ap,);
format = va_arg(ap, char *);
(void) vpfmt(stderr, MM_ERROR, format, ap);
va_end(ap);
(void) abort();
}

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

SEE ALSO pfmt(3C),stdarg(5)
NAME  vprintf, vfprintf, vsprintf – print formatted output of a variable argument list

SYNOPSIS  
#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);

MT-LEVEL  See the NOTES section of this page.

DESCRIPTION  vprintf(), vfprintf(), and vsprintf() are the same as printf(), fprintf(), and sprintf() respectively, except that instead of being called with a variable number of arguments, they are called with an argument list as defined by the <stdarg.h> header.

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  vprintf() and vfprintf() return the number of characters transmitted, or return −1 if an error was encountered.

EXAMPLES  The following demonstrates how vfprintf() could be used to write an error routine:

```c
#include <stdio.h>
#include <stdarg.h>
...

/*
 * error should be called like
 *   error(function_name, format, arg1, ...);
 */
void error(char *function_name, char *format, ...) {
  va_list ap;
  va_start(ap, );
  /* print out name of function causing error */
  (void) fprintf(stderr, "ERR in %s: ", function_name);
```

modified 13 Apr 1994 3S-1187
/* print out remainder of message */
(void) vfprintf(stderr, format, ap);
va_end(ap);
(void) abort;
}

SEE ALSO printf(3S), stdarg(5)

NOTES vprintf(), vfprintf(), and vsprintf() are MT-Safe in multi-thread applications.
NAME
vsyslog – log message with a varargs argument list

SYNOPSIS
#include <syslog.h>
#include <varargs.h>
int vsyslog(int priority, const char *message, va_list ap);

MT-LEVEL
Safe

DESCRIPTION
vsyslog() is the same as syslog(3) except that instead of being called with a variable number of arguments, it is called with an argument list as defined by varargs(5).

EXAMPLES
The following demonstrates how vsyslog() could be used to write an error routine.

```c
#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();
}

SEE ALSO
syslog(3), varargs(5)
```
NAME
wait, wait3, wait4, waitpid, WIFSTOPPED, WIFSIGNALED, WIFEXITED – wait for process to terminate or stop

SYNOPSIS
/usr/ucb/cc [ flag ... ] file ...
#include <sys/wait.h>
int wait( statusp )
int *statusp;
int wait3( pid, statusp, options )
int pid;
int *statusp;
int options;
#include <sys/time.h>
#include <sys/resource.h>
int wait3( status, 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 \textit{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 \textit{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 \textit{w_status} contain the number of the signal that terminated the process. In addition, if the low-order seventh bit of \textit{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 \textit{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 \textit{pid} has a value of −1 and \textit{options} has a value of zero. Otherwise, the behavior of \texttt{waitpid()} is modified by the values of \textit{pid} and \textit{options} as follows:

\textit{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 \textit{pid} is equal to −1, status is requested for any child process. In this respect, \texttt{waitpid()} is then equivalent to \texttt{wait()}.  
- If \textit{pid} is greater than zero, it specifies the process ID of a single child process for which status is requested.  
- If \textit{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 \textit{pid} is less than −1, status is requested for any child process whose process group ID is equal to the absolute value of \textit{pid}.

\textit{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 \textit{pid}.  
  \item \texttt{WUNTRACED}  
  The status of any child processes specified by \textit{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 \textit{status} parameter is defined as above. The \textit{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{SIGTTIN}, \texttt{SIGTTOU}, \texttt{SIGTSTP}, 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 \textit{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 are

modified 5 Mar 1993
used are currently available. They are returned in `rusage.ru_utime` and `rusage.ru_stime`, respectively.

When the `WNOHANG` option is specified and no processes have status to report, `wait3()` returns 0. The `WNOHANG` and `WUNTRACED` options may be combined by ORing the two values.

`wait4()` is another alternate interface. With a `pid` argument of 0, it is equivalent to `wait3()`. If `pid` has a nonzero value, then `wait4()` returns status only for the indicated process ID, but not for any other child processes.

`WIFSTOPPED`, `WIFSIGNALED`, `WIFEXITED`, are macros that take an argument `status`, of type `int`, as returned by `wait()`, or `wait3()`, or `wait4()`. `WIFSTOPPED` evaluates to true (1) when the process for which the `wait()` call was made is stopped, or to false (0) otherwise. `WIFSIGNALED` evaluates to true when the process was terminated with a signal. `WIFEXITED` evaluates to true when the process exited by using an `exit(2)` call.

**RETURN VALUES**

If `wait()` or `waitpid()` returns due to a stopped or terminated child process, the process ID of the child is returned to the calling process. Otherwise, a value of –1 is returned and `errno` is set to indicate the error.

If `wait()` or `waitpid()` return due to the delivery of a signal to the calling process, a value of –1 is returned and `errno` is set to EINTR. If `waitpid()` function was invoked with `WNOHANG` set in `options`, it has at least one child process specified by `pid` for which status is not available, and status is not available for any process specified by `pid`, a value of zero is returned. Otherwise, a value of –1 is returned, and `errno` is set to indicate the error.

`wait3()` and `wait4()` returns 0 if `WNOHANG` is specified and there are no stopped or exited children, and returns the process ID of the child process if it returns due to a stopped or terminated child process. Otherwise, they returns a value of –1 and sets `errno` to indicate the error.

**ERRORS**

`wait()`, `wait3()` or `wait4()` will fail and return immediately if one or more of the following are true:

- **ECHILD** The calling process has no existing unwaited-for child processes.
- **EFAULT** The `status` or `rusage` arguments point to an illegal address.

`waitpid()` may set `errno` to:

- **ECHILD** The process or process group specified by `pid` does not exist or is not a child of the calling process.
- **EINVAL** The value of `options` is not valid.
- **EINTR** The function was interrupted by a signal. The value of the location pointed to by `statusp` is undefined.

`wait()`, and `wait3()`, and `wait4()` will terminate prematurely, return –1, and set `errno` to EINTR upon the arrival of a signal whose `SY_interrupt` bit in its flags field is set (see `sigvec(3B)` and `siginterrupt(3B)`). `signal(3B)`, sets this bit for any signal it catches.
SEE ALSO  exit(2), ptrace(2), wait(2), waitpid(2), getrusage(3C), siginterrupt(3B), signal(3B), sigvec(3B), signal(3C)

NOTES  Use of these interfaces should be restricted to only applications written on BSD platforms. Use of these interfaces with any of the system libraries or in multi-thread applications is unsupported.

If a parent process terminates without waiting on its children, the initialization process (process ID = 1) inherits the children.

wait(), and wait3(), and wait4() are automatically restarted when a process receives a signal while awaiting termination of a child process, unless the SV_INTERRUPT bit is set in the flags for that signal.

Calls to wait() with an argument of 0 should be cast to type ‘int *’, as in:

wait((int *)0)

Previous SunOS releases used union wait *statusp and union wait status in place of int *statusp and int status. The union contained a member w_status that could be treated in the same way as status.

Other members of the wait union could be used to extract this information more conveniently:

- If the w_stopval member had the value WSTOPPED, the child process had stopped; the value of the w_stopsig member was the signal that stopped the process.
- If the w_termsig member was non-zero, the child process terminated due to a signal; the value of the w_termsig member was the number of the signal that terminated the process. If the w_coredump member was non-zero, a core dump was produced.
- Otherwise, the child process terminated due to a call to exit(). The value of the w_retcode member was the low-order 8 bits of the argument that the child process passed to exit().

union wait is obsolete in light of the new specifications provided by IEEE Std 1003.1-1988 and endorsed by SVID89 and XPG3. SunOS Release 4.1 supports union wait for backward compatibility, but it will disappear in a future release.
NAME

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.
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)`

modified 3 Mar 1995 3C-1195
### NOTES

If a parent process terminates without waiting on its children, the initialization process (process ID = 1) inherits the children. `wait3()`, and `wait4()` are automatically restarted when a process receives a signal while awaiting termination of a child process, unless the **SA_RESTART** bit is not set in the flags for that signal.
NAME  wconv, towupper, towlower – Process Code character conversion macros

SYNOPSIS  cc [ flag ... ] file ... -lw [ library ... ]
#include <widec.h>
#include <wctype.h>
int towlower(int c);
int towupper(int c);

MT-LEVEL  MT-Safe with exceptions

CHARACTER CONVERSION MACROS  These macros perform simple case conversions on Latin characters in Process Code, wchar_t, from the primary and supplementary codesets, by table lookup.

  towupper(c)  converts the lower-case Latin character c to its upper-case equivalent. If c is not a lower-case Latin character c is returned.

  towlower(c)  converts the upper-case character c to its lower-case equivalent. If c is not an upper-case Latin character c is returned.

SEE ALSO  ctype(3C), setlocale(3C), iswalpha(3I), stdio(3S)

NOTES  towupper and towlower can be used safely in a multi-thread application, as long as setlocale(3C) is not being called to change the locale.
NAME  wcscoll, wscoll – wide character string comparison using collating information

SYNOPSIS  
```
cc [ flag ... ] file ... -lw [ library ... ]
#include <wchar.h>
int wcscoll(const wchar_t *ws1, const wchar_t *ws2);
int wscoll(const wchar_t *ws1, const wchar_t *ws2);
```

MT-LEVEL  MT-Safe

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.

SEE ALSO  setlocale(3C), wcscmp(3I), wcsxfrm(3I)

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.

wcsxfrm(3I) and wcscmp(3I) should be used for sorting large lists.

3I-1198  modified 16 May 1994
NAME        wcsftime – convert date and time to wide character string

SYNOPSIS    cc [ flag … ] file … -lw [ library … ]
              #include <wchar.h>
              size_t wcsftime(wchar_t *wcs, size_t maxsize, const char *format,
                               const struct tm *timptr);

DESCRIPTION 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 behaviour is undefined.

RETURN VALUES 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 ALSO malloc(3C), mbstowcs(3C), strftime(3C)
NAME
wcstod, wstod, watof – convert wide character string to double-precision number

SYNOPSIS
cc [ flag ... ] file ... -lw [ library ... ]
#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);

MT-LEVEL
MT-Safe

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(3I)); a subject sequence interpreted as a floating-point constant; and a final wide-character string of one or more unrecognised wide-character codes, including the terminating null wide character code of the input wide character string. They then attempt to convert the subject sequence to a floating-point number, and return the result.

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.

SEE ALSO
iswspace(3I), localeconv(3C), scanf(3S), setlocale(3C), wcstol(3I)

NOTES
Because 0 is returned on error and is also a valid return on success, an application wishing to check for error situations should set errno to 0, call wcstod() or wstod(), then check errno and if it is non-zero, assume an error has occurred.
NAME
wcstol, wstol, watol, watoll, watoi – convert wide character string to long integer

SYNOPSIS
cc [ flag ... ] file ... -lw [ library ... ]
#include <wchar.h>

long int wcstol(const wchar_t *nptr, wchar_t **endptr, int base);
long int wstol(const wchar_t *nptr, wchar_t **endptr, int base);
long watol(wchar_t *nptr);
long long watoll(wchar_t *nptr);
int watoi(wchar_t *nptr);

MT-LEVEL
MT-Safe

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(3I)), 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.

watol() is equivalent to wstol(str, (wchar_t **)NULL, 10).

watoll() is the long-long (double long) version of watol().

watoi() is equivalent to (int)watol().

RETURN VALUES

Upon successful completion, wcstol() and wstol() return the converted value, if any. If no conversion could be performed, 0 is returned, and errno may be set to indicate the error. If the correct value is outside the range of representable values, {LONG_MAX} or {LONG_MIN} is returned (according to the sign of the value), and errno is set to ERANGE.

ERRORS

wcstol() and wstol() will fail if:

EINVAL The value of base is not supported.
ERANGE The value to be returned is not representable.

wcstol() and wstol() may fail if:

EINVAL No conversion could be performed.

SEE ALSO

iswalpha(3I), iswspace(3I), scanf(3S), wcstod(3I),

NOTES

Because 0, {LONG_MIN}, and {LONG_MAX} are returned on error and are also valid returns on success, an application wishing to check for error situations should set errno to 0, call wcstol() or wstol(), then check errno and if it is non-zero assume an error has occurred.

Truncation from long long to long can take place upon assignment or by an explicit cast.
NAME  wcstoul – convert wide character string to unsigned long

SYNOPSIS  cc [ flag ...] file ... -lw [ library ...]
            #include <wchar.h>
            unsigned long int wcstoul(const wchar_t *nptr, wchar_t **endptr, int base);

MT-LEVEL  MT-Safe

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(3I)); 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 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 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.

3I-1204  modified 16 May 1994
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{nptr} is stored in the object pointed to by \textit{endptr}; provided that \textit{endptr} is not a null pointer.

**RETURN VALUE** Upon successful completion, \texttt{wcstoul()} returns the converted value, if any. If no conversion could be performed, 0 is returned and \texttt{errno} may be set to indicate the error. 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** \texttt{wcstoul()} will fail if:

- **EINVAL** The value of \texttt{base} is not supported.
- **ERANGE** The value to be returned is not representable.

\texttt{wcstoul()} function may fail if:

- **EINVAL** No conversion could be performed.

**SEE ALSO** \texttt{isspace(3C), iswalpha(3I), scanf(3S), wcstod(3I), wcstol(3I)}

**WARNINGS** Because 0 and \texttt{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 \texttt{errno} to 0, call \texttt{wcstoul()}, then check \texttt{errno} and if it is non-zero, assume an error has occurred.

Unlike \texttt{wcstod(3I)} and \texttt{wcstol(3I)}, \texttt{wcstoul()} must always return a non-negative number; so, using the return value of \texttt{wcstoul()} for out-of-range numbers with \texttt{wcstoul()}.
NAME
wcstring, wcscat, wscat, wcsncat, wsnpcat, wcsncmp, wcsncmp, wcscpy, wcscpy, wcscncpy, wscncpy, wcscpy, wsslen, wcslen, wcwidth, wcswidth, wcchr, wsshr, wcsrchr, wschr, wcschr, wstrchr, windex, windex, wcspbrk, wscppbrk, wsspcmp, wsspcmp, wsspcmp, wsspcmp, wcscspn, wsspcmp, wsspcmp, wcscspn, wsspcmp, wsspcmp, wcscspn, wcstok, wstrok – wide character string operations

SYNOPSIS
cc [ flag ... ] file ... -lw [ library ... ]
#include <wchar.h>

wchar_t *wcsct(wchar_t *ws1, const wchar_t *ws2);
wchar_t *wscat(const wchar_t *ws1, const wchar_t *ws2);
wchar_t *wcsncat(const wchar_t *ws1, const wchar_t *ws2, size_t n);
wchar_t *wsncat(const wchar_t *ws1, const wchar_t *ws2, size_t n);
int wcsncmp(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 wscmp(const wchar_t *ws1, const wchar_t *ws2, size_t n);
wchar_t *wcscpy(const wchar_t *ws1, const wchar_t *ws2);
wchar_t *wscpy(const wchar_t *ws1, const wchar_t *ws2);
wchar_t *wcsncpy(const wchar_t *ws1, const wchar_t *ws2, size_t n);
wchar_t *wsncpy(const wchar_t *ws1, const wchar_t *ws2, size_t n);
size_t wcslen(const wchar_t *ws);
size_t wslen(const wchar_t *ws);
int wcwidth(wint_t wc);
int wcswidth(const wchar_t *wpsc, size_t n);
wchar_t *wceschr(const wchar_t *ws, wint_t wc);
wchar_t *wscchr(const wchar_t *ws, wint_t wc);
wchar_t *wceschr(const wchar_t *ws, size_t n);
wchar_t *wscchr(const wchar_t *ws, size_t n);
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 wcsspn(const wchar_t *ws1, const wchar_t *ws2);
size_t wsscspsn(const wchar_t *ws1, const wchar_t *ws2);
size_t wcscspn(const wchar_t *ws1, const wchar_t *ws2);
size_t wsscspsn(const wchar_t *ws1, const wchar_t *ws2);
size_t wcsscspsn(const wchar_t *ws1, const wchar_t *ws2);
wchar_t *wcstok(const wchar_t *ws1, const wchar_t *ws2);
wchar_t *wstok(const wchar_t *ws1, const wchar_t *ws2);
MT-LEVEL | MT-Safe
DESCRIPTION | These functions operate on wide character strings terminated by wchar_t NULL characters. During appending or copying, these routines do not check for an overflow condition of the receiving string. In the following, ws, ws1, and ws2 point to wide character strings terminated by a wchar_t NULL.

wcsat(), wscat() | The wcsat() 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 behaviour is undefined. Both functions return s1; no return value is reserved to indicate an error.

wcsncat(), wsncat() | 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.

wscmp(), wscmp() | The wscmp() and wscmp() functions compare the wide character string pointed to by ws1 to the wide character string pointed to by ws2. The sign of a non-zero return value is determined by the sign of the difference between the values of the first pair of wide-character codes that differ in the objects being compared. Upon completion, both functions return an integer greater than, equal to, or less than zero, if the wide character string pointed to by ws1 is greater than, equal to, or less than the wide character string pointed to by ws2.

wcsncmp(), wsncmp() | 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.

wscpy(), wscpy() | The wscpy() 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 behaviour is undefined. Both functions return ws1; no return value is reserved to indicate an error.

wcsncpy(), wsncpy() | 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 behaviour is undefined. Both functions return ws1; no return value is reserved to indicate an error.

modified 16 May 1994
objects that overlap, the behaviour is undefined. If the array pointed to by \textit{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 \textit{ws1}, until a total \( n \) wide-character codes are written. Both functions return \textit{ws1}; no return value is reserved to indicate an error.

\textbf{wcslen()}, \textbf{wslen()}

The \textbf{wcslen()} and \textbf{wslen()} functions compute the number of wide-character codes in the wide character string to which \textit{ws} points, not including the terminating null wide-character code. Both functions return \textit{ws}; no return value is reserved to indicate an error.

\textbf{wcwidth()}

The \textbf{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 \textbf{wchar_t}, and must be a wide-character code corresponding to a valid character in the current locale. The function returns 0 if \( wc \) is a null wide-character code; the number of column positions to be occupied by the wide-character code \( wc \); or \(-1\) if \( wc \) does not correspond to a printing wide-character code.

\textbf{wcswidth()}

The \textbf{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 \textit{pwcs}. The function returns 0 if \textit{pwcs} points to a null wide-character code; the number of column positions to be occupied by the wide character string pointed to by \textit{pwcs}; or \(-1\) if any wide-character code in the wide character string pointed to by \textit{pwcs} is not a printing wide-character code.

\textbf{wcschr()}, \textbf{wschr()}

The \textbf{wcschr()} and \textbf{wschr()} functions locate the first occurrence of \( wc \) in the wide character string pointed to by \textit{ws}. The value of \( wc \) must be a character representable as a type \textbf{wchar_t} and must be a wide-character code corresponding to a valid character in the current locale. The terminating null wide-character code is considered to be part of the wide character string. Upon completion, both functions return a pointer to the wide-character code, or a null pointer if the wide-character code is not found.

\textbf{wcsrchr()}, \textbf{wsrchr()}

The \textbf{wcsrchr()} and \textbf{wsrchr()} functions locate the last occurrence of \( wc \) in the wide character string pointed to by \textit{ws}. The value of \( wc \) must be a character representable as a type \textbf{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.

\textbf{windex()}, \textbf{wrindex()}

The \textbf{windex()} and \textbf{wrindex()} functions behave the same as \textbf{wschr()} and \textbf{wsrchr()}, respectively.

\textbf{wcsbrk()}, \textbf{wspbrk()}

The \textbf{wcsbrk()} and \textbf{wspbrk()} functions locate the first occurrence in the wide character string pointed to by \textit{ws1} of any wide-character code from the wide character string pointed to by \textit{ws2}. Upon successful completion, the function returns a pointer to the wide-character code, or a null pointer if no wide-character code from \textit{ws2} occurs in \textit{ws1}.

3I-1208 modified 16 May 1994
wcswcs() 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.

wcsspn(), wsspn() The wcsspn() and wsspn() 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.

wcscspn(), wscspn() The wcscspn() and wscspn() 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.

wcstok(), wstok() 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 has ws1 as its first argument, and is followed by calls with a null pointer as their first argument. The separator string pointed to by ws2 may be different from call to call. The first call in the sequence searches the wide character string pointed to by ws1 for the first wide-character code that is not contained in the current separator string pointed to by ws2. If no such wide-character code is found, then there are no tokens in the wide character string pointed to by ws1, and wcstok() and wstok() return a null pointer. If such a wide-character code is found, it is the start of the first token.

wcstok() and wstok() then search from that point for a wide-character code that is contained in the current separator string. If no such wide-character code is found, the current token extends to the end of the wide character string pointed to by ws1, and subsequent searches for a token will return a null pointer. If such a wide-character code is found, it is overwritten by a null wide character, which terminates the current token. wcstok() and 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.

SEE ALSO malloc(3C), string(3C), wstring(3I)
NAME    wcsxfrm, wsxfrm – wide character string transformation

SYNOPSIS  cc [ flag ... ] file ... -lw [ library ... ]  
#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);

MT-LEVEL  MT-Safe

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(3I) or wscmp(3I) functions are applied to 
two transformed wide strings, they return a value greater than, equal to, or less than 0, 
corresponding to the result of the wcscoll(3I) or wscoll(3I) function applied to the same 
two original wide character strings. No more than 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 behaviour 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.

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.

SEE ALSO  wcscmp(3I), wcscoll(3I), wscmp(3I), wscoll(3I)
NAME  
wctype - define character class

SYNOPSIS  
cc [ flag ... ] file ... -lw [ library ... ]
#include <wchar.h>

wctype_t wctype(const char *charclass);

MT-LEVEL  
MT-Safe

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", and "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(3I). wctype() determines values of wctype_t according to the rules of the coded character set defined by character type information in the program's locale (category LC_CTYPE). The values returned by wctype() are valid until a call to setlocale(3C) that modifies the category LC_CTYPE.

RETURN VALUES  
wctype() returns 0 if the given character class name is not valid for the current locale (category LC_CTYPE); otherwise it returns an object of type wctype_t that can be used in calls to iswctype().

SEE ALSO  
iswctype(3I), setlocale(3C)
NAME  wordexp, wordfree – perform word expansions

SYNOPSIS
#include <wordexp.h>

int wordexp(const char *words, wordexp_t *pwordexp, int flags);
void wordfree(wordexp_t *pwordexp);

MT-LEVEL  MT-Safe

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, pwordexp->we_offs is used to specify how many NULL pointers to add to the beginning of pwordexp->we_wordv. In other words, pwordexp->we_wordv will point to pwordexp->we_offs NULL pointers, followed by pwordexp->we_wordc word pointers, followed by a NULL pointer.

WRDE_NOCMD  Fail if command substitution is requested.

WRDE_REUSE  The pwordexp argument was passed to a previous successful call to wordexp(), and has not been passed to wordfree(). The result will be the same as if the application had called wordfree() and then called wordexp() without WRDE_REUSE.

WRDE_SHOWERR  Do not redirect stderr to /dev/null.

WRDE_UNDEF  Report error on an attempt to expand an undefined shell variable.

The WRDE_APPEND flag can be used to append a new set of words to those generated by a previous call to wordexp(). The following rules apply when two or more calls to wordexp() are made with the same value of pwordexp and without intervening calls to wordfree():

1. The first such call must not set WRDE_APPEND. All subsequent calls must set it.
2. All of the calls must set WRDE_DOOFFS, or all must not set it.
3. After the second and each subsequent call, pwordexp->we_wordv will point to a list containing the following:
   a. zero or more NULL pointers, as specified by WRDE_DOOFFS and pwordexp->we_offs.
   b. pointers to the words that were in the pwordexp->we_wordv list before the call, in the same order as before.
   c. pointers to the new words generated by the latest call, in the specified order.
4. The count returned in pwordexp->we_wordc will be the total number of words from all of the calls.
5. The application can change any of the fields after a call to wordexp(), but if it does it must reset them to the original value before a subsequent call, using the same pwordexp value, to wordfree() or wordexp() with the WRDE_APPEND or WRDE_REUSE flag.

If words contains an unquoted:

    NEWLINE | & ; < > ( ) { } 

in an inappropriate context, wordexp() will fail, and the number of expanded words will be zero.

Unless WRDE_SHOWERR is set in flags, wordexp() will redirect stderr to /dev/null for any utilities executed as a result of command substitution while expanding words. If WRDE_SHOWERR is set, wordexp() may write messages to stderr if syntax errors are
detected while expanding words.

If WRDE_DOOFFS is set, then \texttt{pwordexp->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 \texttt{flags}.

- \textbf{WRDE\_CMDSUB} Command substitution requested when \texttt{WRDE\_NOCMD} was set in \texttt{flags}.

- \textbf{WRDE\_NOSPACE} Attempt to allocate memory failed.

- \textbf{WRDE\_SYNTAX} Shell syntax error, such as unbalanced parentheses or unterminated string.

**RETURN VALUES**

The following values are returned by \texttt{wordexp()}:

- \texttt{0} successful completion.
- \texttt{non-zero} an error has occurred.

- \textbf{WRDE\_NOSPACE} \texttt{pwordexp->we\_word} 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.

**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 \textbf{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.

**SEE ALSO** \texttt{fnmatch(3C), glob(3C)}
NAME wsprintf – formatted output conversion

SYNOPSIS cc [ flag ... ] file ... -lw [ library ... ]
#include <stdio.h>
#include <widec.h>
int wsprintf(wchar_t *s, const char *format, /* arg */ ...);

MT-LEVEL MT-Safe

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.

SEE ALSO wsscanf(3I), printf(3S), scanf(3S), sprintf(3S)
NAME       wsscanf – formatted input conversion

SYNOPSIS  cc [ flag ...] file ... -lw [ library ... ]
#include <stdio.h>
#include <widec.h>
int wsscanf(wchar_t *s, const char *format,/* pointer */ ...);

MT-LEVEL   MT-Safe

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 sscanf(3S).

RETURN VALUES wsscanf() returns the number of characters matched. On error wsscanf() returns a negative value.

SEE ALSO  wsprintf(3I), printf(3S), scanf(3S)
NAME  wstring, wcasecmp, wncasecmp, wsdup, wcol – Process Code string operations

SYNOPSIS  cc [ flag ... ] file ... −lw [ library ... ]
#include <widec.h>

int wcasecmp(const wchar_t *s1, const wchar_t *s2);
int wncasecmp(const wchar_t *s1, const wchar_t *s2, int n);
wchar_t *wsdup(const wchar_t *s);
int wcol(const wchar_t *s);

MT-LEVEL  MT-Safe

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(), wncasecmp()  The wcasecmp() 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. wncasecmp() 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.

wcol()  The wcol() function returns the screen display width (in columns) of the Process Code string s.

SEE ALSO  malloc(3C), string(3C), wstring(3I)

modified 13 May 1994
### NAME
xdr – library routines for external data representation

### MT-LEVEL
Safe

### DESCRIPTION
XDR routines allow C programmers to describe arbitrary data structures in a machine-independent fashion. Data for remote procedure calls (RPC) are transmitted using these routines.

### Index to Routines
The following table lists XDR routines and the manual reference pages on which they are described:

<table>
<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_float</td>
<td>xdr_simple(3N)</td>
</tr>
<tr>
<td>xdr_free</td>
<td>xdr_simple(3N)</td>
</tr>
<tr>
<td>xdr_getpos</td>
<td>xdr_admin(3N)</td>
</tr>
<tr>
<td>xdr_hyper</td>
<td>xdr_simple(3N)</td>
</tr>
<tr>
<td>xdr_inline</td>
<td>xdr_admin(3N)</td>
</tr>
<tr>
<td>xdr_int</td>
<td>xdr_simple(3N)</td>
</tr>
<tr>
<td>xdr_long</td>
<td>xdr_simple(3N)</td>
</tr>
<tr>
<td>xdr_longlong_t</td>
<td>xdr_simple(3N)</td>
</tr>
<tr>
<td>xdrOpaque</td>
<td>xdr_complex(3N)</td>
</tr>
<tr>
<td>xdr_pointer</td>
<td>xdr_complex(3N)</td>
</tr>
<tr>
<td>xdr_quadruple</td>
<td>xdr_simple(3N)</td>
</tr>
<tr>
<td>xdr_reference</td>
<td>xdr_complex(3N)</td>
</tr>
<tr>
<td>xdr_setpos</td>
<td>xdr_admin(3N)</td>
</tr>
<tr>
<td>xdr_short</td>
<td>xdr_simple(3N)</td>
</tr>
<tr>
<td>xdr_sizeof</td>
<td>xdr_admin(3N)</td>
</tr>
<tr>
<td>xdr_string</td>
<td>xdr_complex(3N)</td>
</tr>
<tr>
<td>xdr_u_char</td>
<td>xdr_simple(3N)</td>
</tr>
<tr>
<td>xdr_u_hyper</td>
<td>xdr_simple(3N)</td>
</tr>
<tr>
<td>xdr_u_int</td>
<td>xdr_simple(3N)</td>
</tr>
<tr>
<td>xdr_u_long</td>
<td>xdr_simple(3N)</td>
</tr>
<tr>
<td>xdr_u_longlong_t</td>
<td>xdr_simple(3N)</td>
</tr>
<tr>
<td>xdr_u_short</td>
<td>xdr_simple(3N)</td>
</tr>
<tr>
<td>xdr_union</td>
<td>xdr_complex(3N)</td>
</tr>
<tr>
<td>xdr_vector</td>
<td>xdr_complex(3N)</td>
</tr>
<tr>
<td>xdr_void</td>
<td>xdr_simple(3N)</td>
</tr>
</tbody>
</table>

3N-1218 modified 30 Mar 1993
xdr_wrapstring xdr_complex(3N)
xdrmem_create xdr_create(3N)
xdrrec_create xdr_create(3N)
xdrrec_endofrecord xdr_admin(3N)
xdrrec_eof xdr_admin(3N)
xdrrec_readbytes xdr_admin(3N)
xdrrec_skiprecord xdr_admin(3N)
xdrstdio_create xdr_create(3N)

SEE ALSO rpc(3N), xdr_admin(3N), xdr_complex(3N), xdr_create(3N), xdr_simple(3N)
<table>
<thead>
<tr>
<th>NAME</th>
<th>xdr_admin, xdr_control, xdr_getpos, xdr_inline, xdrrec_endofrecord, xdrrec_eof, xdrrec_readbytes, xdrrec_skiprecord, xdr_setpos, xdr_sizeof – library routines for external data representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-LEVEL</td>
<td>Safe</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>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.</td>
</tr>
<tr>
<td>Routines</td>
<td>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.</td>
</tr>
</tbody>
</table>

```c
#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:

- XDR_GET_BYTES_AVAIL xdr_bytesrec * return number of bytes left unconsumed in the stream and a flag indicating whether or not this is the last fragment.

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.

```c
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.
```

```c
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.
```

```c
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.
```

```c
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.
```

```c
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.
```

```c
unsigned long xdr_sizeof(xdrproc_t func, void *data);
This routine returns the number of bytes required to encode data using the XDR filter function func, excluding potential overhead such as RPC headers or record markers. 0 is returned on error. This information might be used to select between transport protocols, or to determine the buffer size for various lower levels of RPC client and server creation routines, or to allocate storage when XDR is used outside of the RPC subsystem.
```
SEE ALSO  malloc(3C), rpc(3N), xdr_complex(3N), xdr_create(3N), xdr_simple(3N)
NAME  xdr_complex, xdr_array, xdr_bytes, xdr_opaque, xdr_pointer, xdr_reference, xdr_string, xdr_union, xdr_vector, xdr_wrapstring – library routines for external data representation

MT-LEVEL  Safe

DESCRIPTION  XDR library routines allow C programmers to describe complex data structures in a machine-independent fashion. Protocols such as remote procedure calls (RPC) use these routines to describe the format of the data. These routines are the XDR library routines for complex data structures. They require the creation of XDR stream (see xdr_create(3N)).

Routines  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 *aarp is null when decoding, xdr_array() allocates memory and *aarp 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 xdrobj);
Like xdr_reference() except that it serializes NULL pointers, whereas xdr_reference() does not. Thus, xdr_pointer() can represent recursive data structures, such as binary trees or linked lists. If *objpp is null when decoding, xdr_pointer() allocates memory and *objpp points to it.

bool_t xdr_reference(XDR *xdrs, caddr_t **pp, u_int size, const xdrproc_t *proc);
xdr_reference() provides pointer chasing within structures. The parameter pp is the address of the pointer; size is the sizeof the structure that *pp points to; and proc is an XDR procedure that translates the structure between its C form and its external representation. If *pp is null when decoding, xdr_reference() allocates memory and *pp points to it. This routine returns 1 if it succeeds, 0 otherwise.
Warning: this routine does not understand NULL pointers. Use xdr_pointer() instead.

bool_t xdr_string(XDR *xdrs, char **sp, const u_int maxsize);
xdr_string() translates between C strings and their corresponding external representations. Strings cannot be longer than maxsize. Note: sp is the address of the string’s pointer. If *sp is null when decoding, xdr_string() allocates memory and *sp points to it. This routine returns 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 elsize,
const xdrproc_t *elproc);
xdr_vector() translates between fixed-length arrays and their corresponding external representations. The parameter arrp is the address of the pointer to the array, while size is the element count of the array. The parameter elsize is the sizeof each of the array’s elements, and elproc is an XDR routine that translates
between the array elements’ C form and their external representation. This routine returns TRUE if it succeeds, FALSE otherwise.

```c
bool_t xdr_wrapstring(XDR *xdrs, char **sp);
```

A routine that calls xdr_string(xdrs, sp, maxuint); where maxuint is the maximum value of an unsigned integer.

Many routines, such as xdr_array(), xdr_pointer(), and xdr_vector() take a function pointer of type xdrproc_t(), which takes two arguments. xdr_string(), one of the most frequently used routines, requires three arguments, while xdr_wrapstring() only requires two. For these routines, xdr_wrapstring() is desirable. This routine returns TRUE if it succeeds, FALSE otherwise.

SEE ALSO rpc(3N), xdr_admin(3N), xdr_create(3N), xdr_simple(3N)
### NAME

xdr_create, xdr_destroy, xdrmem_create, xdrrec_create, xdrstdio_create – library routines for external data representation stream creation

### MT-LEVEL

MT-Safe

### DESCRIPTION

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

3N-1226 modified 7 May 1993
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.

**SEE ALSO** `read(2)`, `write(2)`, `malloc(3C)`, `rpc(3N)`, `xdr_admin(3N)`, `xdr_complex(3N)`, `xdr_simple(3N)`, `fclose(3S)`

modified 7 May 1993
NAME

xdr_simple, xdr_bool, xdr_char, xdr_double, xdr_enum, xdr_float, xdr_free, xdr_hyper,
xdr_int, xdr_long, xdr_longlong_t, xdr_quadruple, xdr_short, xdr_u_char, xdr_u_hyper,
xdr_u_int, xdr_u_long, xdr_u_longlong_t, xdr_u_short, xdr_void – library routines for
external data representation

MT-LEVEL

Safe

DESCRIPTION

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 represen-
tations. 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 charac-
ters are not packed, and occupy 4 bytes each. For arrays of characters, it is
worthwhile to consider xdr_bytes(), xdr_opaque(), or xdr_string() (see
xdr_complex(3N)).

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 *lhp);
   xdr_hyper() translates between ANSI C long long integers and their external representations. This routine returns TRUE if it succeeds, FALSE otherwise.

bool_t xdr_int(XDR *xdrs, int *ip);
   xdr_int() translates between C integers and their external representations. This routine returns TRUE if it succeeds, FALSE otherwise.

bool_t xdr_long(XDR *xdrs, long *lp);
   xdr_long() translates between C long integers and their external representations. This routine returns TRUE if it succeeds, FALSE otherwise.

bool_t xdr_longlong_t(XDR *xdrs, longlong_t *llp);
   xdr_longlong_t() translates between ANSI C long long integers and their external representations. This routine returns TRUE if it succeeds, FALSE otherwise. This routine is identical to xdr_hyper().

bool_t xdr_quadruple(XDR *xdrs, long double *pq);
   xdr_quadruple() translates between IEEE quadruple precision floating point numbers and their external representations. This routine returns TRUE if it succeeds, FALSE otherwise.

bool_t xdr_short(XDR *xdrs, short *sp);
   xdr_short() translates between C short integers and their external representations. This routine returns TRUE if it succeeds, FALSE otherwise.

bool_t xdr_u_char(XDR *xdrs, unsigned char *ucp);
   xdr_u_char() translates between unsigned C characters and their external representations. This routine returns TRUE if it succeeds, FALSE otherwise.

bool_t xdr_u_hyper(XDR *xdrs, u_longlong_t *ullp);
   xdr_u_hyper() translates between unsigned ANSI C long long integers and their external representations. This routine returns TRUE if it succeeds, FALSE otherwise.
bool_t xdr_u_int(XDR *xdrs, unsigned *up);
A filter primitive that translates between a C unsigned integer and its external representation. This routine returns TRUE if it succeeds, FALSE otherwise.

bool_t xdr_u_long(XDR *xdrs, unsigned long *ulp);
xdr_u_long() translates between C unsigned long integers and their external representations. This routine returns TRUE if it succeeds, FALSE otherwise.

bool_t xdr_u_longlong_t(XDR *xdrs, u_longlong_t *ullp);
xdr_u_longlong_t() translates between unsigned ANSI C long long integers and their external representations. This routine returns TRUE if it succeeds, FALSE otherwise. This routine is identical to xdr_u_hyper().

bool_t xdr_u_short(XDR *xdrs, unsigned short *usp);
xdr_u_short() translates between C unsigned short integers and their external representations. This routine returns TRUE if it succeeds, FALSE otherwise.

bool_t xdr_void(void);
This routine always returns TRUE. It may be passed to RPC routines that require a function parameter, where nothing is to be done.

SEE ALSO malloc(3C), rpc(3N), xdr_admin(3N), xdr_complex(3N), xdr_create(3N)
NAME 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)
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_attrvalue_t *fn_valuelist_next(FN_valuelist_t *vl, FN_identifier_t **attr_syntax,
                                  FN_status_t *status);
void fn_valuelist_destroy(FN_valuelist_t *vl, FN_status_t *status);
FN_multigetlist_t *fn_attr_multi_get(FN_ctx_t *ctx,
                                     const FN_composite_name_t *name, const FN_attrset_t *attr_ids,
                                     FN_status_t *status);
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);
```

3N-1232  modified 4 Nov 1994
The following describes briefly the operations in the base attribute interface. Detailed descriptions are given in the respective reference manual pages for these operations.

- **fn_attr_get()** returns the attribute identified. **fn_attr_modify()** modifies the attribute identified as described by `mod_op`.
- **fn_attr_get_ids()** returns the identifiers of the attributes of the named object.
- **fn_attr_get_values()** and its set of related operations are used for returning the individual values of an attribute.
- **fn_attr_multi_get()** and its set of related operations are used for returning the requested attributes associated with the named object. **fn_attr_multi_modify()** modifies multiple attributes associated with the named object in a single invocation.
- **fn_ctx_get_syntax_attrs()** returns the syntax attributes associated with the named context.

### ERRORS

`status` is set as described in **FN_status_t**(3N) and **xfn_status_codes**(3N). The following status codes are of special relevance to attribute operations:

- **FN_E_ATTR_VALUE_REQUIRED**
  The operation attempted to create an attribute without a value, and the specific naming system does not allow this.
- **FN_E_ATTR_NO_PERMISSION**
  The caller did not have permission to perform the attempted attribute operation.
- **FN_E_INSUFFICIENT_RESOURCES**
  There is insufficient resources to retrieve the requested attribute(s).
- **FN_E_INVALID_ATTR_IDENTIFIER**
  The attribute identifier was not in a format acceptable to the naming system, or its contents was not valid for the format specified for the identifier.
- **FN_E_INVALID_ATTR_VALUE**
  One of the values supplied was not in the appropriate form for the given attribute.
- **FN_E_NO_SUCH_ATTRIBUTE**
  The object did not have an attribute with the given identifier.
- **FN_E_TOO_MANY_ATTR_VALUES**
  The operation attempted to associate more values with an attribute than the naming system supported.

### APPLICATION 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 are a range of attribute models in which an attribute is associated with a name in a context, or an attribute is associated with the object named, or both. XFN accommodates all of these alternatives. Invoking an attribute operation using the
target context and the terminal atomic name accesses either the attributes that are associated with the target name or target named object — this is dependent on the underlying attribute model. This document uses the term *attributes associated with a named object* to refer to all of these cases.

XFN specifies no guarantees about the relationship between the attributes and the reference associated with a given name. Some naming systems may store the reference bound to a name in one or more attributes associated with a name. Attribute operations might affect the information used to construct a reference.

To avoid undefined results, programmers must use the operations in the context interface and not attribute operations when the intention is to manipulate a reference. Programmers should avoid the use of specific knowledge about how an XFN context implementation over a particular naming system constructs references.

**SEE ALSO**

- FN_attribute_t(3N), FN_attrvalue_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_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), xfn(3N)
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.

The details of the syntax and the semantics of these functions are described in

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 covert- ing 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)

modified 4 Nov 1994  3N-1235
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 (FN_ID_STRING format) 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);
```

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fn_syntax_type</td>
<td>Its value is the ASCII string &quot;standard&quot; if the context supports the XFN standard syntax model. Its value is an implementation-specific value if another syntax model is supported.</td>
</tr>
<tr>
<td>fn_std_syntax_direction</td>
<td>Its value is an ASCII string, one of &quot;left_to_right&quot;, &quot;right_to_left&quot;, or &quot;flat&quot;. 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 — all names are atomic).</td>
</tr>
<tr>
<td>fn_std_syntax_separator</td>
<td>Its value is the separator string for this name syntax. This attribute is required unless the fn_std_syntax_direction is &quot;flat&quot;.</td>
</tr>
<tr>
<td>fn_std_syntax_escape</td>
<td>If present, its value is the escape string for this name syntax.</td>
</tr>
<tr>
<td>fn_std_syntax_case_insensitive</td>
<td>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.</td>
</tr>
<tr>
<td>fn_std_syntax_begin_quote</td>
<td>If present, its value is the begin-quote string for this syntax. There can be multiple values for this attribute.</td>
</tr>
<tr>
<td>fn_std_syntax_end_quote</td>
<td>If present, its value is the end-quote string for this syntax. There can be multiple values for this attribute.</td>
</tr>
</tbody>
</table>
fn_std_syntax_ava_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**

**FN_E_ILLEGAL_NAME**
The name supplied to the operation was not a well-formed component according to the name syntax of the context.

**FN_E_INCOMPATIBLE_CODE_SETS**
Code set mismatches that occur during the construction of the compound name’s string form are resolved in an implementation-dependent way. When an implementation discovers that a compound name has components with incompatible code sets, it returns the error code **FN_E_INCOMPATIBLE_CODE_SETS**.

**FN_E_INVALID_SYNTAX_ATTRS**
The syntax attributes supplied are invalid or insufficient to fully specify the syntax.

**FN_E_SYNTAX_NOT_SUPPORTED**
The syntax specified is not supported.

**APPLICATION USAGE**
Most applications treat names as opaque data and 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_atrset_t(3N)**, **FN_compound_name_t(3N)**, **FN_identifier_t(3N)**, **FN_string_t(3N)**, **fn_ctx_get_syntax_attrs(3N)**, **xfn(3N)**

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:

```
#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
The link status fields can be retrieved using `fn_status_link_code()`, `fn_status_link_remaining_name()`, `fn_status_link_resolved_name()` and `fn_status_link_resolved_ref()`.

**ERRORS**

The following status codes are of special relevance when performing operations involving XFN links:

- **FN_E_LINK_ERROR**
  
  There was an error encountered resolving an XFN link encountered during resolution of the supplied name. Check the link part of the status object to determine cause of the link error.

- **FN_E_LINK_LOOP_LIMIT**
  
  A non-terminating loop (cycle) in the resolution can arise due to XFN links encountered during the resolution of a composite name. This code indicates either the definite detection of such a cycle, or that resolution exceeded an implementation-defined limit on the number of XFN links allowed for a single operation invoked by the caller.

- **FN_E_MALFORMED_LINK**
  
  A malformed link reference was encountered. For the `fn_ctx_lookup_link()` operation, the name supplied resolved to a reference that was not a link.

**APPLICATION USAGE**

For the `fn_ctx_bind()`, `fn_ctx_unbind()`, `fn_ctx_rename()`, `fn_ctx_lookup_link()`, `fn_ctx_create_subcontext()` and `fn_ctx_destroy_subcontext()` operations, resolution of the given name continues to the target context — that named by all but the terminal atomic part of the given name; the terminal atomic name is not resolved. Consequently, for operations that involve unbinding the terminal atomic part such as `fn_ctx_unbind()`, if the terminal atomic name is bound to a link, the link is not followed and the link itself is unbound from the terminal atomic name.

Many naming systems support a native notion of link that may be used within the naming system itself. XFN does not determine whether there is any relationship between such native links and XFN links.

**SEE ALSO**

`FN_composite_name_t(3N)`, `FN_ref_t(3N)`, `FN_status_t(3N)`, `fn_ctx_bind(3N)`, `fn_ctx_destroy_subcontext(3N)`, `fn_ctx_lookup(3N)`, `fn_ctx_lookup_link(3N)`, `fn_ctx_rename(3N)`, `fn_ctx_unbind(3N)`, `xfn_status_codes(3N)`, `xfn(3N)`
NAME  xfn_status_codes – descriptions of XFN status codes

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 link status code describes the error that occurred while resolving the XFN 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. Currently values and interpretations for the following codes are determined by XFN.

<table>
<thead>
<tr>
<th>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_PERMISSIONS</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 involved in the operation. This may be because the naming system is busy, or is not providing service.</td>
</tr>
</tbody>
</table>

modified 4 Nov 1994 3N-1241
<table>
<thead>
<tr>
<th>Error 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 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_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 contents 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 encountered 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 fn_ctx_lookup_link(), the name supplied resolved to a reference that was not a link.</td>
</tr>
<tr>
<td>FN_E_MALFORMED_REFERENCE</td>
<td></td>
</tr>
</tbody>
</table>
A context object could not be constructed from the supplied reference, because the reference was not properly formed.

**FN_E_NAME_IN_USE**
(Only for operations that bind names.) The supplied name was already in use.

**FN_E_NAME_NOT_FOUND**
Resolution of the supplied composite name proceeded to a context in which the next atomic component of the name was not bound.

**FN_E_NO_SUCH_ATTRIBUTE**
The object did not have an attribute with the given identifier.

**FN_E_NO_SUPPORTED_ADDRESS**
A context object could not be constructed from a particular reference. The reference contained no address type over which the context interface was supported.

**FN_E_NOT_A_CONTEXT**
Either one of the intermediate atomic names did not name a context, and resolution could not proceed beyond this point, or the operation required that the caller supply the name of a context, and the name did not resolve to a reference for a context.

**FN_E_OPERATION_NOT_SUPPORTED**
The operation attempted is not supported.

**FN_E_PARTIAL_RESULT**
The operation attempted is returning a partial result.

**FN_E_SYNTAX_NOT_SUPPORTED**
The syntax type specified is not supported.

**FN_E_TOO_MANY_ATTR_VALUES**
The operation attempted to associate more values with an attribute than the naming system supported.

**FN_E_UNSPECIFIED_ERROR**
An error occurred that could not be classified by any of the other error codes.

**FILES**

```c
#include <xfn/xfn.h>
```

**SEE ALSO**

FN_status_t(3N), xfn(3N)
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);

MT-LEVEL

Unsafe

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 database. The error YPERR_KEY is returned if the key already exists in the database. To store an item into the database without concern for whether it exists already or not, pass 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.

SEE ALSO

secure_rpc(3N), ypclnt(3N)

NOTES

This interface is unsafe in multithreaded applications. Unsafe interfaces should be called only from the main thread.
NAME
ypclnt, yp_get_default_domain, yp_bind, yp_unbind, yp_match, yp_first, yp_next,
yp_all, yp_order, yp_master, yperr_string, ypprot_err - NIS Version 2 client interface

SYNOPSIS
cc [flag ...] file ... -lnsl [ library ... ]
#include <rpcsvc/ypclnt.h>
#include <rpcsvc/yp_prot.h>

MT-LEVEL
Unsafe

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 ypser 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 outvalen. 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 ypserve process returns any answer (success or failure), the
ypclnt layer will return control to the user code, either with an error code, or a
success code and any results.

yp_get_default_domain (char **outdomain);

The NIS lookup calls require a map name and a domain name, at minimum. It is
assumed that the client process knows the name of the map of interest. Client
processes should fetch the node's default domain by calling
yp_get_default_domain(), and use the returned outdomain as the indomain
parameter to successive NIS name service calls. The domain thus returned is the
same as that returned using the SI_SRPC_DOMAIN command to the sysinfo(2)
system call.

yp_match(char *indomain, char *inmap, char *inkey, int inkeylen, char **outval,
int *outvallen);

yp_match() returns the value associated with a passed key. This key must be
exact; no pattern matching is available. yp_match() requires a full YP map
name; for example, hosts.byname instead of the nickname hosts.

yp_first(char *indomain, char *inmap, char **outkey, int *outkeylen, char **outval,
int *outvallen);

yp_first() returns the first key-value pair from the named map in the named
domain.

yp_next(char *indomain, char *inmap, char *inkey, int inkeylen, char **outkey,
int *outkeylen, char **outval, int *outvallen);
yp_next() returns the next key-value pair in a named map. The inkey parameter must be the outkey returned from an initial call to yp_first() (to get the second key-value pair) or the one returned from the \( n \)th call to yp_next() (to get the \( n \)th + second key-value pair). Similarly, the inkeylen parameter must be the outkeylen returned from the earlier yp_first() or yp_next() call.

The concept of first (and, for that matter, of next) is particular to the structure of the NIS map being processing; there is no relation in retrieval order to either the lexical order within any original (non-NIS name service) data base, or to any obvious numerical sorting order on the keys, values, or key-value pairs. The only ordering guarantee made is that if the yp_first() function is called on a particular map, and then the yp_next() function is repeatedly called on the same map at the same server until the call fails with a reason of YPERR_NOMORE, every entry in the data base will be seen exactly once. Further, if the same sequence of operations is performed on the same map at the same server, the entries will be seen in the same order.

Under conditions of heavy server load or server failure, it is possible for the domain to become unbound, then bound once again (perhaps to a different server) while a client is running. This can cause a break in one of the enumeration rules; specific entries may be seen twice by the client, or not at all. This approach protects the client from error messages that would otherwise be returned in the midst of the enumeration. The next paragraph describes a better solution to enumerating all entries in a map.

yp_all(char *indomain, char *inmap, struct ypall_callback *incallback);

yp_all() provides a way to transfer an entire map from server to client in a single request using TCP (rather than UDP as with other functions in this package). The entire transaction take place as a single RPC request and response. yp_all() can be used just like any other NIS name service procedure, identify the map in the normal manner, and supply the name of a function which will be called to process each key-value pair within the map. The call to yp_all() returns only when the transaction is completed (successfully or unsuccessfully), or the foreach() function decides that it does not want to see any more key-value pairs.

The third parameter to yp_all() is

```
struct ypall_callback *incallback {
    int (*foreach)( );
    char *data;
};
```

The function foreach() is called

```
foreach(int instatus, char *inkey, int inkeylen, char *inval, int invallen,
        char *indata);
```
The `instatus` parameter will hold one of the return status values defined in `<rpcsvc/yp_prot.h` — either `YP_TRUE` or an error code. (See `ypprot_err()`, below, for a function which converts an NIS name service protocol error code to a `ypclnt` layer error code.)

The key and value parameters are somewhat different than defined in the synopsis section above. First, the memory pointed to by the `inkey` and `inval` parameters is private to the `yp_all()` function, and is overwritten with the arrival of each new key-value pair. It is the responsibility of the `foreach()` function to do something useful with the contents of that memory, but it does not own the memory itself. Key and value objects presented to the `foreach()` function look exactly as they do in the server's map — if they were not NEWLINE-terminated or null-terminated in the map, they will not be here either.

The `indata` parameter is the contents of the `incallback→data` element passed to `yp_all()`. The `data` element of the callback structure may be used to share state information between the `foreach()` function and the mainline code. Its use is optional, and no part of the NIS client package inspects its contents — cast it to something useful, or ignore it.

The `foreach()` function is a Boolean. It should return 0 to indicate that it wants to be called again for further received key-value pairs, or non-zero to stop the flow of key-value pairs. If `foreach()` returns a non-zero value, it is not called again; the functional value of `yp_all()` is then 0.

```c
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".

```c
yp_master(char *indomain, char *inmap, char **outname);
```

`yp_master()` returns the machine name of the master NIS server for a map.

```c
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.

```c
ypprot_err (unsigned int incode);
```

`ypprot_err()` takes an NIS name service protocol error code as input, and returns a `ypclnt` layer error code, which may be used in turn as an input to `yperr_string()`.
RETURN VALUES  All integer functions return 0 if the requested operation is successful, or one of the following errors if the operation fails.

YPERR_ACCESS  15 /* access violation */
YPERR_BADARGS  1 /* args to function are bad */
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

SEE ALSO  nis+(1), ypcat(1), ypmatch(1), ypwhich(1), rpcbind(1M), rpc.nisd(1M), ypbind(1M),
            sysinfo(2), malloc(3C), yp®les(4)

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

Special Characters

 Islamist — non-local goto, 3B-1008
 NOTE — annotate source code with info for tools, 3X-116
 setjmp — non-local goto, 3B-1008

A

 abort — terminate the process abnormally, 3C-126
 abs — return absolute value of integer, 3C-127
 accept — accept a connection on a socket, 3N-128
 access utmpx file entry
 — endutxent, 3C-560
 — getutmp, 3C-560
 — getutmpx, 3C-560
 — getutxent, 3C-560
 — getutxid, 3C-560
 — getutxline, 3C-560
 — pututxline, 3C-560
 — setutxent, 3C-560
 — updutxline, 3C-560
 — updwtmp, 3C-560
 — updwtmpx, 3C-560
 — utmpxname, 3C-560

accounting
 — time accounting for current process — times, 3B-1159

 acos — trigonometric arccosine, 3M-1164
 acosh — inverse hyperbolic function, 3M-578
 acquire and release stream lock — flockfile, 3S-387
 funclockfile, 3S-387
 add a string of wchar_t characters to a curses window and advance cursor — curs_addwstr, 3X-218
 addnstr, 3X-218
 addwstr, 3X-218
 mvaddnstr, 3X-218
 mvaddwstr, 3X-218
 mvwaddnstr, 3X-218
 mvwaddwstr, 3X-218
 waddnstr, 3X-218
 waddwstr, 3X-218
 add a wchar_t character (with attributes) to a curses window and advance cursor —
 curs_addwch, 3X-214
 addwch, 3X-214
 echowchar, 3X-214
 mvaddwch, 3X-214
 mvwaddwch, 3X-214
 waddwch, 3X-214
 wechowchar, 3X-214
 add string of wchar_t characters (and attributes) to a curses window — curs_addwchstr, 3X-216
 addwchstr, 3X-216
 addwchstr, 3X-216
 mvaddwchstr, 3X-216
 mvaddwchstr, 3X-216
add string of wchar_t characters (and attributes) to a curses window —
curs_addwchstr, continued
mvwaddwchnstr, 3X-216
mvwaddwchnstr, 3X-216
waddwchnstr, 3X-216
waddwchnstr, 3X-216
additional severities
define — addsev, 3C-137
addnwstr — add a string of wchar_t characters to a curses window and advance cursor, 3X-218
addsev — define additional severities, 3C-137
addseverity — build a list of severity levels for an application for use with fmtmsg, 3C-138
addwch — add a wchar_t character (with attributes) to a curses window and advance cursor, 3X-214
addwchnstr — add string of wchar_t characters (and attributes) to a curses window, 3X-216
addwchstr — add string of wchar_t characters (and attributes) to a curses window, 3X-216
addwstr — add a string of wchar_t characters to a curses window and advance cursor, 3X-218
adjcurspos — moving the cursor by character, 3X-219
advance — regular expression compile and match routines, 3G-909
aio_cancel — cancel asynchronous I/O request, 3R-140
aio_error — retrieve error status of asynchronous I/O operation, 3R-146
aio_fsync — asynchronous file synchronization, 3R-142
aio_read — asynchronous read and write operations, 3R-144
aio_return — retrieve return status of asynchronous I/O operation, 3R-146
aio_suspend — wait for asynchronous I/O request, 3R-149
aio_write — asynchronous read and write operations, 3R-144
aiocancel — cancel an asynchronous operation, 3-151
aioread — initiate asynchronous read, 3-152
aiowait — wait for completion of asynchronous I/O operation, 3-154
aioread — initiate asynchronous write, 3-152
alarm
schedule signal after interval in microseconds — ualarm, 3C-1172
ALE curses library, See curses library
alphasort — scan a directory, 3B-971
annotate source code with info for tools
— _NOTE, 3X-116
— NOTE, 3X-116
applications
build a list of severity levels for use with fmtmsg — addseverity, 3C-138
display a message on stderr or system console — fmtmsg, 3C-390
get entries from symbol table — nlist, 3B-791
arc — graphics interface, 3-817
arithmetic
compute the quotient and remainder — div, 3C-310
arithmetic, 48-bit integer
generate uniformly distributed pseudo-random numbers — drand48, 3C-321
ascftime — convert date and time to string, 3C-1064
asin — trigonometric arcsine, 3M-1164
asinh — inverse hyperbolic function, 3M-578
assert — verify program assertion, 3C-155
asynchronous file synchronization
— aio_sync, 3R-142
asynchronous I/O
— aio_cancel, 3R-140
— aiocancel, 3-151
— aioread, 3-152
— aiowait, 3-154
— aiowrite, 3-152
retrieve error status — aio_error, 3R-146
retrieve return status — aio_return, 3R-146
wait for request — aio_suspend, 3R-149
asynchronous read and write operations
— aio_read, aio_write, 3R-144
asysmem — return physical memory information,
atan — trigonometric arctangent, 3M-1164
atan2 — trigonometric arctangent, 3M-1164
atanh — inverse hyperbolic function, 3M-578
atexit — add program termination routine, 3C-156
attroff — curses character and window attribute control routines, 3X-220
attron — curses character and window attribute control routines, 3X-220
attrset — curses character and window attribute control routines, 3X-220
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_in_ipc — creating audit record tokens, 3-160
au_to_in_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, 3-160
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_subject — creating audit record tokens, 3-160
au_to_socket — creating audit record tokens, 3-160
au_to_text — creating audit record tokens, 3-160
auth_destroy — library routines for client side remote procedure call authentication, 3N-925
authnone_create — library routines for client side remote procedure call authentication, 3N-925
authsys_create — library routines for client side remote procedure call authentication, 3N-925
authsys_create_default — library routines for client side remote procedure call authentication, 3N-925

B
base-64 ASCII characters
convert from long integer — l64a, 3C-125
convert to long integer — a64l, 3C-125
basename — return the last element of path name, 3G-164

audit control file information, continued
— getacflag, 3-469
— getacinfo, 3-469
— getacmin, 3-469
— getacna, 3-469
— setac, 3-469

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_write, 3-157

auth_destroy — library routines for client side remote procedure call authentication, 3N-925
authnone_create — library routines for client side remote procedure call authentication, 3N-925
authsys_create — library routines for client side remote procedure call authentication, 3N-925
authsys_create_default — library routines for client side remote procedure call authentication, 3N-925

B
base-64 ASCII characters
convert from long integer — l64a, 3C-125
convert to long integer — a64l, 3C-125
basename — return the last element of path name, 3G-164

Index—3
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_ipc, 3-160
- au_to_ipc_perm, 3-160
- au_to_iport, 3-160
- au_to_me, 3-160
- au_toOpaque, 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

bcmp — operates on variable length strings of bytes, 3C-173
bcopy — operates on variable length strings of bytes, 3C-173

Bessel functions
- j0, 3M-165
- j1, 3M-165
- jn, 3M-165
- y0, 3M-165
- yi, 3M-165
- yn, 3M-165

cbrt — cube root function, 3M-1048

bzero — operates on variable length strings of bytes, 3C-173

Index—4
ctime — convert date and time to string, 3C-1064
character based forms package
— forms, 3X-454
character based menus package
— menus, 3X-707
character based panels package
— panels, 3X-808
character classification and conversion
— _tolower, 3C-199
— _toupper, 3C-199
— conv, 3C-199
— toascii, 3C-199
— tolower, 3C-199
— toupper, 3C-199
character handling
— ctype, 3C-208
— isalnum, 3C-208
— isalpha, 3C-208
— isascii, 3C-208
— iscntrl, 3C-208
— isdigit, 3C-208
— isgraph, 3C-208
— islower, 3C-208
— isprint, 3C-208
— ispunct, 3C-208
— isspace, 3C-208
— isupper, 3C-208
— isxdigit, 3C-208
check whether or not Volume Management is managing a pathname — volmgt_inuse, 3X-1180
circle — graphics interface, 3-817
client side remote procedure call authentication, library routines for
— auth_destroy, 3N-925
— authnone_create, 3N-925
— authsys_create, 3N-925
— authsys_create_default, 3N-925
— rpc_clnt_auth, 3N-925
cIClient_call — library routines for client side calls, 3N-927
cIClient_control — library routines for dealing with creation and manipulation of CLIENT handles, 3N-930
cIClient_create — library routines for dealing with creation and manipulation of CLIENT handles, 3N-930
cIClient_create_timed — library routines for dealing with creation and manipulation of CLIENT handles, 3N-930
cIClient_create_vers — library routines for dealing with creation and manipulation of CLIENT handles, 3N-930
cIClient_destroy — library routines for dealing with creation and manipulation of CLIENT handles, 3N-930
cIClient_dg_create — library routines for dealing with creation and manipulation of CLIENT handles, 3N-930
cIClient_freereres — library routines for client side calls, 3N-927
cIClient_geterr — library routines for client side calls, 3N-927
cIClient_pcreateerror — library routines for dealing with creation and manipulation of CLIENT handles, 3N-930
cIClient_perrno — library routines for client side calls, 3N-927
cIClient_perror — library routines for client side calls, 3N-927
cIClient_raw_create — library routines for dealing with creation and manipulation of CLIENT handles, 3N-930
cIClient_spcreateerror — library routines for dealing with creation and manipulation of CLIENT handles, 3N-930
cIClient_sperrno — library routines for client side calls, 3N-927
cIClient_sperror — library routines for client side calls, 3N-927
cIClient_tli_create — library routines for dealing with creation and manipulation of CLIENT handles, 3N-930
cIClient_tlitaire — library routines for dealing with creation and manipulation of CLIENT handles, 3N-930
cIClient_tp_create — library routines for dealing with creation and manipulation of CLIENT handles, 3N-930

Index−5
clnt_tp_create_timed — library routines for dealing with creation and manipulation of CLIENT handles, 3N-930
clnt_vc_create — library routines for dealing with creation and manipulation of CLIENT handles, 3N-930
clock — report CPU time used, 3C-186
clock_getres — high-resolution clock operations, 3R-187
clock_gettime — high-resolution clock operations, 3R-187
clock_settime — high-resolution clock operations, 3R-187
closelog — close system log file, 3-1086
closepl — graphics interface, 3-817
closevt — graphics interface, 3-817
code conversion allocation function — iconv_open, 3-583
code conversion deallocation function — iconv_close, 3-582
code conversion function — iconv, 3-580
command options
  get option letter from argument vector — getopt, 3C-512
command suboptions
  parse suboptions from a string — getsubopt, 3C-547
commands
  open, close to and from a command — p2open, p2close, 3G-798
  return stream to remote — rcmd, 3N-892
communications
  accept a connect request — t_accept, 3N-1091
  accept a connection on a socket — accept, 3N-128
  acknowledge receipt of an orderly release indication — t_sndrel, 3N-1120
  allocate memory for argument structures — t_alloc, 3N-1093
  bind a name to a socket — bind, 3N-167
  bind an address to a transport endpoint — t_bind, 3N-1095
communications, continued
  close a transport endpoint — t_close, 3N-1098
  create a pair of connected sockets — socketpair, 3N-1045
  create an endpoint for communication — socket, 3N-1042
  disable a transport endpoint — t_unbind, 3N-1135
  establish a connection with another transport user — t_connect, 3N-1099
  establish a transport endpoint — t_open, 3N-1111
  free allocated memory — t_free, 3N-1103
  get name of peer connected to socket — getpeername, 3N-517
  get protocol-specific service information — t_getinfo, 3N-1104
  get socket name — getsockname, 3N-539
  get the current state — t_getstate, 3N-1107
  initiate a connection on a socket — connect, 3N-197
  initiate an orderly release of connection — t_sndrel, 3N-1129
  listen for a connect request — t_listen, 3N-1108
  listen for connections on a socket — listen, 3N-648
  look at the current event on a transport endpoint — t_look, 3N-1110
  manage options for a transport endpoint — t_optmgmt, 3N-1112
  produce error message — t_error, 3N-1102
  receive a data unit — t_rcvudata, 3N-1121
  receive a unit data error indication — t_rcvuderr, 3N-1123
  receive data or expedited data sent over a connection — t_rcv, 3N-1114
  receive the confirmation from a connect request — t_rcvconnect, 3N-1116
  retrieve information from disconnect — t_rcvdis, 3N-1118
  scatter data in order to test the network — spray, 3N-1046
  send a data unit — t_sndudata, 3N-1130

Index—6
communications, continued
send a message from a socket — send, sendto, sendmsg, 3N-1003
send data or expedited data over a connection — t_snd, 3N-1125
send user-initiated disconnect request — t_snddis, 3N-1127
shut down part of a full-duplex connection — shutdown, 3N-1020
synchronize transport library — t_sync, 3N-1133
compile — regular expression compile and match routines, 3G-909
cond_attr — condition variable initialization attributes, 3T-842
cond_broadcast() — signal a condition variable, 3T-189
cond_destroy() — destroy a condition variable, 3T-189
cond_init() — initialize a condition variable, 3T-189
cond_signal() — signal a condition variable, 3T-189
cond_wait() — wait for a condition variable, 3T-189
cond_wait() — wait for a condition variable, 3T-189
condition variable initialization attributes —
cond_attr, 3T-842
configuration script
execute — doconfig, 3N-319
confstr — get configurable variables, 3C-196
connect — initiate a connection on socket, 3N-197
cont — graphics interface, 3-817
control system log
close system log — closelog, 3-1086
set log priority mask — setlogmask, 3-1086
start system log — openlog, 3-1086
write to system log — syslog, 3-1086
convert a supplied name into an absolute pathname that can be used to access removable media —
media_findname, 3X-680
convert between Volume Management symbolic
names, and the devices that correspond to them
— volmgt_symdev, 3X-1183
— volmgt_symname, 3X-1183
convert date and time to string — strftime, 3C-1064
asctime, 3C-1064
cftime, 3C-1064
convert date and time to wide character string —
wcsftime, 3I-1199
convert floating-point number to string
— econvert, 3C-324
— ecvt, 3C-324
— fconvert, 3C-324
— fcvt, 3C-324
— fprintf, 3B-821
— gcov, 3-324
— gcvt, 3-324
— printf, 3B-821
— qeconvert, 3C-324
— qecvt, 3C-324
— qfconvert, 3C-324
— qfcvt, 3C-324
— qprintf, 3B-821
— qvfprintf, 3B-821
— qvprintf, 3B-821
— qvsprintf, 3B-821
convert to wchar_t strings
wsprintf, 3I-1215
convert wide character string to double-precision number — wcstod, 3I-1200
watof, 3I-1200
wstod, 3I-1200
convert wide character string to unsigned long —
wcsftoul, 3I-1204

copysign() function, 3M-584

cos — trigonometric cosine, 3M-1164
cosh — hyperbolic cosine, 3M-578

CPU time
  report for calling process — clock, 3C-186

CPU-use
  prepare execution profile — monitor, 3C-718

CRT handling and optimization package
  — curses, 3X-286

cset — get information on EUC codesets, 3I-202

csetcol — get information on EUC codesets, 3I-202

csetlen — get information on EUC codesets, 3I-202

csetno — get information on EUC codesets, 3I-202

ctermid — generate path name for controlling terminal, 3S-203

ctermid_r — generate path name for controlling terminal, 3S-203

cctype — character handling, 3C-208

cube root — cbrt, 3M-1048

current working directory
  get pathname — getcwd, 3C-480

curs_addwch — add a wchar_t character (with attributes) to a curses window and advance
cursor, 3X-214

curs_addwchstr — add string of wchar_t characters (and attributes) to a curses window, 3X-216

curs_addwstr — add a string of wchar_t characters to a curses window and advance cursor, 3X-218

curs_ailecompat — moving the cursor by character, 3X-219

curs_attr — curses character and window attribute control routines, 3X-220
  Attributes, 3X-220

curs_getwch — get (or push back) wchar_t characters from curses terminal keyboard, 3X-237
  Function Keys, 3X-237

curs_getwstr — get wchar_t character strings from curses terminal keyboard, 3X-241

curs_inswch — insert a wchar_t character before the character under the cursor in a curses window, 3X-253

curs_inswstr — insert wchar_t string before character under the cursor in a curses window, 3X-254

curs_inwch — get a wchar_t character and its attributes from a curses window, 3X-256

curs_inwchstr — get a string of wchar_t characters (and attributes) from a curses window, 3X-257

curs_inwstr — get a string of wchar_t characters from a curses window, 3X-258

curs_pad — create and display curses pads, 3X-265

curses — CRT handling and optimization package, 3X-286

curses bell and screen flash routines
  — beep, 3X-222
  — curs_beep, 3X-222
  — flash, 3X-222

curses borders, horizontal and vertical lines, create
  — border, 3X-224
  — box, 3X-224
  — curs_border, 3X-224
  — wborder, 3X-224
  — whline, 3X-224
  — wvline, 3X-224

curses character and window attribute control routines
  — attroff, 3X-220
  — attron, 3X-220
  — attrset, 3X-220
  — curs_attr, 3X-220
  — standend, 3X-220
  — standout, 3X-220
  — wattroff, 3X-220
  — wattroff, 3X-220
  — wattroff, 3X-220
  — wstandend, 3X-220
  — wstandout, 3X-220

curses color manipulation routines
  — can_change_colors, 3X-227
curses color manipulation routines, continued
   — color_content, 3X-227
   — curs_color, 3X-227
   — has_colors, 3X-227
   — init_color, 3X-227
   — init_pair, 3X-227
   — pair_content, 3X-227
   — start_color, 3X-227

Curses cursor and window coordinates
   — curs_getyx, 3X-242
   — getbegyx, 3X-242
   — getmaxyx, 3X-242
   — getparyx, 3X-242
   — getyx, 3X-242

Curses environment query routines
   — baudrate, 3X-275
   — curs_termattrs, 3X-275
   — erasechar, 3X-275
   — has_ic, 3X-275
   — has_il, 3X-275
   — killchar, 3X-275
   — longname, 3X-275
   — termattrs, 3X-275
   — termname, 3X-275

Curses interfaces to termcap library
   — curs_termcap, 3X-277
   — tgetent, 3X-277
   — tgetflag, 3X-277
   — tgetnum, 3X-277
   — tgetstr, 3X-277
   — tgoto, 3X-277
   — tputs, 3X-277

Curses interfaces to terminfo database
   — curs_terminfo, 3X-278
   — del_curterm, 3X-278
   — mvcur, 3X-278
   — putp, 3X-278
   — restartterm, 3X-278
   — set_curterm, 3X-278
   — setterm, 3X-278
   — setupterm, 3X-278
   — tigetflag, 3X-278
   — tigetnum, 3X-278
   — tigetstr, 3X-278
   — tparm, 3X-278

Curses interfaces to terminfo database, continued
   — tputs, 3X-278
   — vidattr, 3X-278
   — vidputs, 3X-278

Curses library, See also form library, menu library, or panel library
   — adjcurspos, 3X-219
   — curs_alecompat, 3X-219
   — moveendch, 3X-219
   — moveprevch, 3X-219
   — wadjcurspos, 3X-219
   — wmoveendch, 3X-219
   — wmoveprevch, 3X-219

Curses miscellaneous utility routines
   — curs_util, 3X-282
   — delay_output, 3X-282
   — filter, 3X-282
   — flushingp, 3X-282
   — getwin, 3X-282
   — keyname, 3X-282
   — putwin, 3X-282
   — unctrl, 3X-282
   — use_env, 3X-282

Curses pads, create and display — curs_pad, 3X-265
   — newpad, 3X-265
   — pechochar, 3X-265
   — pechowchar, 3X-265
   — pnoutrefresh, 3X-265
   — refresh, 3X-265
   — subpad, 3X-265

Curses refresh control routines
   — curs_touch, 3X-281
   — is_linetouched, 3X-281
   — is_wintouched, 3X-281
   — touchline, 3X-281
   — touchwin, 3X-281
   — untouchwin, 3X-281
   — wtouchln, 3X-281

Curses screen initialization and manipulation routines
   — curs_initscr, 3X-245
   — delscreen, 3X-245
   — endwin, 3X-245
   — initscr, 3X-245
curses screen initialization and manipulation routines, continued
   — isendwin, 3X-245
   — newterm, 3X-245
   — set_term, 3X-245

curses screen, read/write from/to file
   — curs_scr_dump, 3X-271
   — scr_dump, 3X-271
   — scr_init, 3X-271
   — scr_restore, 3X-271
   — scr_set, 3X-271

curses soft label routines
   — curs_slk, 3X-273
   — slik_attroff, 3X-273
   — slik_attron, 3X-273
   — slik_attrset, 3X-273
   — slik_clear, 3X-273
   — slik_init, 3X-273
   — slik_label, 3X-273
   — slik_noutrefresh, 3X-273
   — slik_refresh, 3X-273
   — slik_restore, 3X-273
   — slik_set, 3X-273
   — slik_touch, 3X-273

curses terminal input option control routines
   — cbreak, 3X-247
   — curs_inopts, 3X-247
   — echo, 3X-247
   — halfdelay, 3X-247
   — intrflush, 3X-247
   — keypad, 3X-247
   — meta, 3X-247
   — nocbreak, 3X-247
   — nodelay, 3X-247
   — noecho, 3X-247
   — noqiflush, 3X-247
   — noraw, 3X-247
   — notimeout, 3X-247
   — qiflush, 3X-247
   — raw, 3X-247
   — timeout, 3X-247
   — typeahead, 3X-247
   — wtimeout, 3X-247

curses terminal keyboard
   — curs_getstr, 3X-236

curses terminal keyboard, continued
   — getstr, 3X-236
   — mvgetstr, 3X-236
   — mvwgetstr, 3X-236
   — wgetnstr, 3X-236
   — wgetstr, 3X-236

curses terminal keyboard, get characters
   — curs_getch, 3X-232
   — getch, 3X-232
   — mvgetch, 3X-232
   — mvwgetch, 3X-232
   — ungetch, 3X-232
   — wgetch, 3X-232

curses terminal output option control routines
   — clearok, 3X-262
   — curs_outopts, 3X-262
   — idcok, 3X-262
   — idlok, 3X-262
   — immedok, 3X-262
   — leaveok, 3X-262
   — nl, 3X-262
   — nonl, 3X-262
   — scrollok, 3X-262
   — setscren, 3X-262
   — wsetscren, 3X-262

curses window background manipulation routines
   — bkgd, 3X-223
   — bkgdset, 3X-223
   — curs_bkgd, 3X-223
   — wbkgd, 3X-223
   — wbkgdset, 3X-223

curses window cursor
   — curs_move, 3X-261
   — move, 3X-261
   — wmove, 3X-261

curses window, add character and advance cursor
   — addch, 3X-210
   — curs_addch, 3X-210
   — echochar, 3X-210
   — mvwaddch, 3X-210
   — mvwaddch, 3X-210
   — waddch, 3X-210
   — waddch, 3X-210
   — wechochar, 3X-210

curses window, add string of characters
   — addchnstr, 3X-212

Index−10
curses window, add string of characters, continued
  — addchstr, 3X-212
  — curs_addchstr, 3X-212
  — mvaddchnstr, 3X-212
  — mvaddchstr, 3X-212
  — mvwaddchnstr, 3X-212
  — mvwaddchstr, 3X-212
  — waddchnstr, 3X-212
  — waddchstr, 3X-212

curses window, add string of characters and advance cursor
  — addnstr, 3X-213
  — addstr, 3X-213
  — curs_addstr, 3X-213
  — mvaddnstr, 3X-213
  — mvaddstr, 3X-213
  — mvwaddnstr, 3X-213
  — mvwaddstr, 3X-213
  — waddnstr, 3X-213
  — waddstr, 3X-213

curses window, clear all or part
  — clear, 3X-226
  — clrtobot, 3X-226
  — clrtoeol, 3X-226
  — curs_clear, 3X-226
  — erase, 3X-226
  — wclear, 3X-226
  — wclrtobot, 3X-226
  — wclrtoeol, 3X-226
  — werase, 3X-226

curses window, convert formatted input
  — curs_scanw, 3X-270
  — mvscanw, 3X-270
  — mvwscanw, 3X-270
  — scanw, 3X-270
  — wscanw, 3X-270
  — wscantw, 3X-270

curses window, delete and insert lines
  — curs_deleteln, 3X-231
  — deleteln, 3X-231
  — insdelln, 3X-231
  — insertln, 3X-231
  — wdeleteln, 3X-231
  — winsdelln, 3X-231
  — winsertln, 3X-231

curses window, delete character under cursor

  — curs_delch, 3X-230
  — delch, 3X-230
  — mvdelch, 3X-230
  — mvwdelch, 3X-230
  — wdelch, 3X-230

curses window, get character and its attributes
  — curs_inch, 3X-243
  — inch, 3X-243
  — mvinch, 3X-243
  — mvwinch, 3X-243
  — winch, 3X-243

curses window, get string of characters
  — curs_inchstr, 3X-244
  — curs_instr, 3X-252
  — inchstr, 3X-244
  — innstr, 3X-252
  — instr, 3X-252
  — mvinchstr, 3X-244
  — mvinnstr, 3X-252
  — mvinst, 3X-252
  — mvwinchstr, 3X-244
  — mvwinnstr, 3X-252
  — mvwinsch, 3X-250
  — winchstr, 3X-244
  — winchstr, 3X-244
  — winnstr, 3X-252
  — winstr, 3X-252

curses window, insert character before character under cursor
  — curs_insch, 3X-250
  — insch, 3X-250
  — mvinsch, 3X-250
  — mvwinsch, 3X-250
  — winsch, 3X-250

curses window, insert string before character under cursor
  — curs_instr, 3X-251
  — instr, 3X-251
  — insnstr, 3X-251
  — mvinsnstr, 3X-251

Index—11
curses window, insert string before character under cursor, continued
  — mvinsstr, 3X-251
  — mvwinsstr, 3X-251
  — mvwinsstr, 3X-251
  — winsstr, 3X-251
  — winstr, 3X-251

curses window, scroll
  — curs_scroll, 3X-272
  — scrl, 3X-272
  — scroll, 3X-272
  — wscrl, 3X-272

curses windows and lines, refresh
  — curs_refresh, 3X-268
  — doupdate, 3X-268
  — refresh, 3X-268
  — wnoutrefresh, 3X-268
  — wrefresh, 3X-268
  — wredrawln, 3X-268
  — wrefresh, 3X-268

curses windows, create
  — curs_window, 3X-284
  — delwin, 3X-284
  — derwin, 3X-284
  — dupwin, 3X-284
  — mvderwin, 3X-284
  — mvwin, 3X-284
  — newwin, 3X-284
  — subwin, 3X-284
  — syncok, 3X-284
  — wcursyncup, 3X-284
  — wsyncdown, 3X-284
  — wsycup, 3X-284

curses windows, overlap and manipulate
  — copywin, 3X-264
  — curs_overlay, 3X-264
  — overlay, 3X-264
  — overwrite, 3X-264

curses windows, print formatted output
  — curs_printw, 3X-267
  — mvprintw, 3X-267
  — mvwprintw, 3X-267
  — printw, 3X-267
  — vwpwprintw, 3X-267
  — wprintw, 3X-267

curses, low-level routines
  — curs_kernel, 3X-259
  — curs_set, 3X-259
  — def_prog_mode, 3X-259
  — def_shell_mode, 3X-259
  — getsyx, 3X-259
  — napms, 3X-259
  — reset_prog_mode, 3X-259
  — reset_shell_mode, 3X-259
  — resettty, 3X-259
  — ripoffline, 3X-259
  — savetty, 3X-259
  — setsyx, 3X-259

cuserid — get character-string representation of login name of user, 3S-297

D

data base subroutines — dbm, 3B-298
  — dbmclose, 3B-298
  — dbmclose, 3B-298
  — dbminit, 3B-298
  — delete, 3B-298
  — fetch, 3B-298
  — firstkey, 3B-298
  — nextkey, 3B-298
  — store, 3B-298

database subroutines
  — ndbm, 3-746

date and time
  convert to string — asctime, 3C-204
  convert user format date and time — getdate, 3C-481
  — gettimeofday, 3C-554

date and time conversion — strftime, 3C-1073

dbm — data base subroutines, 3B-298
  — dbm_clearerr — reset the error condition, 3-746
  — dbm_close — close database, 3-746
  — dbm_delete — delete a key and its associated contents, 3-746
  — dbm_error — return an error condition, 3-746
  — dbm_fetch — access data stored under a key, 3-746
  — dbm_firstkey — return first key in the database, 3-746
  — dbm_nextkey — return next key in the database,
dbm_open — open database, 3-746
dbm_store — place data under a key, 3-746
dbmclose — data base subroutines, 3B-298
dbminit — data base subroutines, 3B-298
decimal record from double-precision floating —
double_to_decimal, 3-383
decimal record from extended-precision floating —
extended_to_decimal, 3-383
decimal record from quadruple-precision floating —
quadruple_to_decimal, 3-383
decimal record from single-precision floating —
single_to_decimal, 3-383
decimal record to double-precision floating —
decimal_to_double, 3-300
decimal record to extended-precision floating —
decimal_to_extended, 3-300
decimal record to quadruple-precision floating —
decimal_to_quadruple, 3-300
decimal record to single-precision floating —
decimal_to_single, 3-300
decimal_to_double — decimal record to
double-precision floating, 3-300
decimal_to_extended — decimal record to
extended-precision floating, 3-300
decimal_to_quadruple — decimal record to
quadruple-precision floating, 3-300
decimal_to_single — decimal record to single-
precision floating, 3-300
define character class — wchar, 3I-1211
define default catalog — setcat, 3C-1007
define the label for pfmt() and lfmt() —
setlabel, 3C-1014
delete — data base subroutines, 3B-298
C++, decode encoded symbol name (em demangle, 3-301
device number
manage — makedev, major, minor, 3C-662
dgettext — message handling function, 3I-550
dial — establish an outgoing terminal line connection, 3N-302
difftime — computes the difference between two
calendar times, 3C-304
directories
create, remove them in a path — mkdirp,
rmdirp, 3G-710
get current working directory pathname —
getwd, 3C-566
get pathname of current working directory —
getcwd, 3C-480
directory operations
— alphasort, 3B-971
— closedir, 3C-305
— directory, 3C-305
— opendir, 3C-305
— readdir, 3C-305
— readdir_r, 3C-305
— rewinddir, 3C-305
— scandir, 3B-971
— seekdir, 3C-305
— telldir, 3C-305
dirname — report parent directory name of file
path name, 3G-309
display error message in standard format —
pfmt, 3C-813
display error message in standard format and pass
to logging and monitoring services —
1fmt, 3C-631, 3C-1177, 3C-1185
div — compute quotient and remainder, 3C-310
division and remainder operations
— div, 3C-310
— ld, 3C-310
dladdr — translate address to symbolic information,
3X-311
dlclose — close a shared object, 3X-313
dlerror — get diagnostic information, 3X-314
dlopen — open a shared object, 3X-315
dlsym — get address of symbol in shared object,
3X-317
dn_comp — resolver routines, 3N-913
dn_expand — resolver routines, 3N-913
doconfig — execute a configuration script, 3N-319
double_to_decimal — decimal record from
double-precision floating, 3-383

Index—13
dup2 — duplicate an open file descriptor, 3C-323
duplicate an open file descriptor — dup2, 3C-323
dynamic linking
close a shared object — dlclose, 3X-313
get address of symbol in shared object —
dlsym, 3X-317
get diagnostic information — dlerror, 3X-314
open a shared object — dlopen, 3X-315

dl

echowchar — add a wchar_t character (with attributes) to a curses window and advance cursor,
3X-214
econvert — convert number to ASCII, 3-324
ecvt — convert number to ASCII, 3-324
data — last location in program, 3C-369
elf — object file access library, 3E-327
get entries from name list — nlist, 3E-792
elf_begin — process ELF object files, 3E-339
elf_cntl — control an elf file descriptor, 3E-345
elf_end — process ELF object files, 3E-339
elf_errmsg — error handling, 3E-346
elf_errno — error handling, 3E-346
elf_fill — set fill byte, 3E-347
elf_flagdata — manipulate flags, 3E-348
elf_flagehdr — manipulate flags, 3E-348
elf_flagelf — manipulate flags, 3E-348
elf_flagphdr — manipulate flags, 3E-348
elf_flagshdr — manipulate flags, 3E-348
elf_getarhdr — retrieve archive member header,
3E-350
elf_getarsym — retrieve archive symbol table,
3E-351
elf_getbase — get the base offset for an object file, 3E-352
elf_getdata — get section data, 3E-353
elf_getident — retrieve file identification data,
3E-357
elf_getscn — get section information, 3E-359
elf_hash — compute hash value, 3E-361
elf_kind — determine file type, 3E-362
elf_memory — process ELF object files, 3E-339
elf_ndxscn — get section information, 3E-359
elf_newdata — get section data, 3E-353
elf_newscn — get section information, 3E-359
elf_next — process ELF object files, 3E-339
elf_nextscn — get section information, 3E-359
elf_rand — process ELF object files, 3E-339
elf_rawdata — get section data, 3E-353
elf_rawfile — retrieve uninterpreted file contents, 3E-363
elf_strptr — make a string pointer, 3E-364
elf_update — update an ELF descriptor, 3E-365
elf_version — coordinate ELF library and application versions, 3E-368
elf_fsize — return the size of an object file type,
3E-333
elf32_getehdr — retrieve class-dependent object file header, 3E-334
elf_getphdr — retrieve class-dependent program header table, 3E-335
elf32_getshdr — retrieve class-dependent section header, 3E-336
elf32_newehdr — retrieve class-dependent object file header, 3E-334
elf_newphdr — retrieve class-dependent program header table, 3E-335
elf32_xlatetof — class-dependent data translation, 3E-337
elf32_xlatetom — class-dependent data translation, 3E-337
encryption
determine whether a buffer of characters is encrypted — isencrypt, 3G-593
encryption, password
— crypt, 3C-201
— encrypt, 3C-201
— setkey, 3C-201
end — last location in program, 3C-369
endac — get audit control file information, 3-469
endauclass — close audit_class database file, 3-469
endauevent — close audit_event database file,
EUC characters, continued
    convert a string of EUC characters from the stream to Process Code — getwfs, 3I-568
    convert a string of Process Code characters to EUC characters and put it on a stream — fputws, 3I-884

EUC codeset, get information
    — getwidth, 3I-567

EUC codesets, get information
    — cset, 3I-202
    — csetcol, 3I-202
    — csetlen, 3I-202
    — csetno, 3I-202
    — wcsetno, 3I-202

EUC display width
    — euccol, 3I-373
    — eucscol, 3I-373
euccol — get EUC character display width, 3I-373
eucscol — get EUC string display width, 3I-373

Euclidean distance function — hypot, 3M-579
eucscol — get EUC string display width, 3I-373

Executable and Linking Format, See elf
exit — terminate process, 3C-374
exit program
    add routine — atexit, 3C-156
exp — exponential function, 3M-375
expm1 — exponential minus 1 function, 3M-375
exponential function — exp, 3M-375
exponential minus 1 function — expm1, 3M-375

Extended Unix Code, See EUC
extended_to_decimal — decimal record from extended-precision floating, 3-383

external data representation
    See XDR, 3N-1218

F
fabs() function, 3M-584
fattach — attach a STREAMS-based file descriptor to an object in the file system name space, 3C-376
fconvert — convert number to ASCII, 3-324
fcvt — convert number to ASCII, 3-324

Index—15
fdatasync — synchronize a file’s data, 3R-379
fdetach — detach a name from a STREAMS-based file descriptor, 3C-380
fetch — data base subroutines, 3B-298
ffs — find first set bit, 3C-382
fgetgrent — get group entry from file, 3C-489
fgetgrent_r — get group entry from file, 3C-489
fgetpos — reposition a file pointer in a stream, 3C-464
fgetpwent — get password entry from a file, 3C-524
fgetpwent_r — get password entry from a file, 3C-524
fgetspent — get shadow password database entry, 3C-543
fgetspent_r — get shadow password database entry(reentrant), 3C-543
fgetwc — get EUC character from stream, 3I-565
fgetws — em convert a string of EUC characters from the stream to Process Code, 3I-568

FIFO
create a new one — mkfifo, 3C-711
file descriptor
duplicate an open one — dup2, 3C-323
get table size — getdtablesize, 3C-486
STREAMS-based, attach to an object in file system name space — fattach, 3C-376
STREAMS-based, detach a name — fdetach, 3C-380
test for a STREAMS file — isastream, 3C-592

file descriptors
apply or remove advisory lock on open file — flock, 3B-385

file name
make a unique one — mktemp, 3C-712

file path names
returns the real file name — realpath, 3C-897

file pointer in a stream
reposition — fsetpos, fgetpos, 3C-464

file tree
recursively descend — ftw, 3C-467

file_to_decimal — decimal record from character stream, 3-1070

files
allows sections of file to be locked — lockf, 3C-653
— remove, 3C-912
report parent directory of file path name — dirname, 3G-309
search for named file in named directories — pathfind, 3G-810
set a file to a specified length — truncate, 3C-1165
synchronize a file’s in-memory state with that on the physical medium — fsync, 3C-465

firstkey — data base subroutines, 3B-298
floating-point number
convert to string — ecvt, 3C-326
floating-point number, determine type
— finite, 3C-594
— fpclass, 3C-594
— isnan, 3C-594
— isnand, 3C-594
— isnanf, 3C-594
— unordered, 3C-594
floating-point number, double-precision
convert string to — atof, 3C-1077
floating-point numbers, manipulate
— frexp, 3C-460
— ldexp, 3C-460
— logb, 3C-460
— modf, 3C-460
— modff, 3C-460
— nextafter, 3C-460
floating-point, manipulate
— scalb, 3C-460
flock — apply or remove an advisory lock on an open file, 3B-385
flockfile — acquire and release stream lock, 3S-387
floor — round to integral value, 3M-389
fmod() function, 3M-584
fmtmsg — display a message on stderr or system console, 3C-390
fnmatch — match filename or path name, 3C-422
fopen — open stream, 3B-424
form library, See also curses library
formatted input conversion — wscanf, 3I-1216
formatted output conversion
 — fprintf, 3B-821
 — printf, 3B-821
 — vfprintf, 3B-821
 — vprintf, 3B-821
 — vsprintf, 3B-821
forms — character based forms package, 3X-454
forms field attributes, set and get
 — field_buffer, 3X-435
 — field_status, 3X-435
 — form_field_buffer, 3X-435
 — set_field_buffer, 3X-435
 — set_field_status, 3X-435
 — set_max_field, 3X-435
forms field characteristics
 — dynamic_field_info, 3X-436
 — field_info, 3X-436
 — form_field_info, 3X-436
forms field data type validation
 — field_arg, 3X-442
 — field_type, 3X-442
 — form_field_validation, 3X-442
 — set_field_type, 3X-442
forms field option routines
 — field_opts, 3X-439
 — field_opts_off, 3X-439
 — field_opts_on, 3X-439
 — form_field_opts, 3X-439
 — set_field_opts, 3X-439
forms field, off-screen data ahead or behind
 — data_ahead, 3X-429
 — data_behind, 3X-429
 — form_data, 3X-429
forms fields, create and destroy
 — dup_field, 3X-438
 — form_field_new, 3X-438
 — free_field, 3X-438
 — link_field, 3X-438
 — new_field, 3X-438
forms fieldtype routines
 — form_fieldtype, 3X-443
 — free_fieldtype, 3X-443
 — link_fieldtype, 3X-443
 — new_fieldtype, 3X-443
 — set_fieldtype_arg, 3X-443
 — set_fieldtype_choice, 3X-443
forms option routines
 — form_opts, 3X-449
 — form_opts_off, 3X-449
 — form_opts_on, 3X-449
 — set_form_opts, 3X-449
forms pagination
 — form_new_page, 3X-448
 — new_page, 3X-448
 — set_new_page, 3X-448
forms window and subwindow association routines
 — form_sub, 3X-453
 — form_win, 3X-453
 — scale_form, 3X-453
 — set_form_sub, 3X-453
 — set_form_win, 3X-453
forms window cursor, position
 — form_cursor, 3X-428
 — pos_form_cursor, 3X-428
forms, application-specific routines
 — field_init, 3X-445
 — field_term, 3X-445
 — form_hook, 3X-445
 — form_init, 3X-445
 — form_term, 3X-445
 — set_field_init, 3X-445
 — set_field_term, 3X-445
 — set_form_init, 3X-445
 — set_form_term, 3X-445
forms, associate application data
 — field_userptr, 3X-441
 — form_field_userptr, 3X-441
 — form_userptr, 3X-452
 — set_field_userptr, 3X-441
 — set_form_userptr, 3X-452
forms, command processor
 — form_driver, 3X-430
forms, connect fields
 — field_driver, 3X-430
 — field_count, 3X-433

Index—17
forms, connect fields, continued
- form_field, 3X-433
- form_fields, 3X-433
- move_field, 3X-433
- set_form_fields, 3X-433
forms, create and destroy
- form_new, 3X-447
- free_form, 3X-447
- new_form, 3X-447
forms, format general appearance
- field_just, 3X-437
- form_field_just, 3X-437
- set_field_just, 3X-437
forms, format general display attributes
- field_back, 3X-434
- field_fore, 3X-434
- field_pad, 3X-434
- form_field_attributes, 3X-434
- set_field_back, 3X-434
- set_field_fore, 3X-434
- set_field_pad, 3X-434
forms, set current page and field
- current_field, 3X-450
- field_index, 3X-450
- form_page, 3X-450
- set_current_field, 3X-450
- set_form_page, 3X-450
forms, write/erase from associated subwindows
- form_post, 3X-451
- post_form, 3X-451
- unpost_form, 3X-451
fpgetmask — IEEE floating-point environment control, 3C-457
fpgetround — IEEE floating-point environment control, 3C-457
fpgetsticky — IEEE floating-point environment control, 3C-457
fprintf — formatted output conversion, 3B-821, 3S-825
fpsetmask — IEEE floating-point environment control, 3C-457
fpsetround — IEEE floating-point environment control, 3C-457
fpsetsticky — IEEE floating-point environment control, 3C-457
fputwc — put wide character on stream, 3I-883
fputws — convert a string of Process Code characters to EUC characters and put it on a stream, 3I-884
free — memory allocator, 3X-169
freopen — open stream, 3B-424
fscanf — convert formatted input, 3S-972
fsetpos — reposition a file pointer in a stream, 3C-464
fsync — synchronize a file’s in-memory state with that on the physical medium, 3C-465
ftime — get date and time, 3C-466
ftruncate — set a file to a specified length, 3C-1165
ftw — walk a file tree, 3C-467
func_to_decimal — decimal record from character function, 3-1070
funlockfile — acquire and release stream lock, 3S-387
G
gamma — log gamma function, 3M-635
gamma_r — log gamma function, 3M-635
gconvert — convert number to ASCII, 3-324
gcvt — convert number to ASCII, 3-324
general terminal interface
- cfgetispeed, 3-1137
- cfgetospeed, 3-1137
- cfsetispeed, 3-1137
- cfsetospeed, 3-1137
- tcdrain, 3-1137
- tcflow, 3-1137
- tcflush, 3-1137
- tcgetattr, 3-1137
- tcgetpgrp, 3-1137
- tcgetsid, 3-1137
- tcsetpgrp, 3-1137
- tcsetattr, 3-1137
- tcsetbreak, 3-1137
- tcsetattr, 3-1137
- tcsetpgrp, 3-1137
generate path name for controlling terminal
- ctermid, 3S-203
generate path name for controlling terminal, continued
  — ctermid_r, 3S-203
generate path names matching a pattern
  — glob, 3C-569
  — globfree, 3C-569
generic transport name-to-address translation
  — netdir, 3N-749
  — netdir_free, 3N-749
  — netdir_getbyaddr, 3N-749
  — netdir_getbyname, 3N-749
  — netdir_mergeaddr, 3N-749
  — netdir_options, 3N-749
  — netdir_strerror, 3N-749
  — netdir_strerror_r, 3N-749
get and set media attributes
  — media_getattr, 3X-682
  — media_setattr, 3X-682
get configurable variables — confstr, 3C-196
get error message string — t_strerror, 3N-1132
get service entry — getservbyname, 3N-535
  — endservent, 3N-535
  — getservbyname_r, 3N-535
  — getservbyport, 3N-535
  — getservbyport_r, 3N-535
  — getservent, 3N-535
  — getservent_r, 3N-535
  — setservent, 3N-535
get wchar_t character strings from curses terminal keyboard —
  — curs_getwstr, 3X-241
  — getwstr, 3X-241
  — mvgetwstr, 3X-241
  — mvwgetwstr, 3X-241
  — wgetwstr, 3X-241
  — wgetwstr, 3X-241
get a string of wchar_t characters from a curses window —
  — curs_inwstr, 3X-258
  — winswstr, 3X-258
get a wchar_t character and its attributes from a curses window —
  — curs_inwch, 3X-256
  — inwch, 3X-256
  — mvinwch, 3X-256
  — mvwvch, 3X-256
  — winwch, 3X-256
get a string of wchar_t characters from a curses window —
  — curs_inwstr, continued
  — winswstr, 3X-258
get a wchar_t character strings from curses terminal keyboard —
  — curs_getwstr, 3X-241
  — getnswstr, 3X-241
  — getwstr, 3X-241
  — mvgetwstr, 3X-241
  — mvwvch, 3X-256
  — mvwgetwstr, 3X-241
  — mvwvch, 3X-256
  — wgetnswstr, 3X-241
  — wgetwstr, 3X-241
getacdir — get audit control file information, 3-469
getacconfig — get audit control file information, 3-469
getacinfo — get audit control file information, 3-469
getacmin — get audit control file information, 3-469
getacna — get audit control file information, 3-469
getauclassent — get audit_class database entry, 3-471
getauclassent_r — get audit_class database
entry, 3-471
getaulearnam — get audit_class database entry, 3-471
getaulearnam_r — get audit_class database entry, 3-471
getauditflags() — generate process audit state, 3-488
getauditflagsbin() — convert audit flag specifications, 3-473
getauditflagschar() — convert audit flag specifications, 3-473
getauevent — get audit_event database entry, 3-474
getauevent_r — get audit_event database entry, 3-474
getauevnam — get audit_event database entry, 3-474
getauevnam_r — get audit_event database entry, 3-474
getauevnonam — get audit_event database entry, 3-474
getauevnum — get audit_event database entry, 3-474
getauevnum_r — get audit_event database entry, 3-474
getauuserent — get audit_user database entry, 3-476
getauserent_r — get audit_user database entry, 3-476
getausernam — get audit_user database entry, 3-476
getausernam_r — get audit_user database entry, 3-476
getcw — get pathname of current working directory, 3C-480
getdate — convert user format date and time, 3C-481
General Specifications, 3C-483
Internal Format Conversion, 3C-482
Modified Conversion Specifications, 3C-482
getdtablesize — get file descriptor table size, 3C-486
getenv — return value for environment name, 3C-487
getgrent — get group entry from database, 3C-489
getgrent_r — get group entry from database, 3C-489
getggrid — get group entry from database, 3C-489
getgrgrid_r — get group entry from database, 3C-489
getgrnam — get group entry from database, 3C-489
getgrnam_r — get group entry from database, 3C-489
gethostid — get unique identifier of current host, 3C-497
gethostname — get name of current host, 3C-498
gethrtime — get high resolution real time, 3C-499
gethrvtime — get high resolution virtual time, 3C-499
getlogin — get login name, 3C-500
getlogin_r — get login name, 3C-500
getmntany — get mnttab file information, 3C-501
getmntent — get mnttab file information, 3C-501
getnwstr — get wchar_t character strings from curses terminal keyboard, 3X-241
getopt — get option letter from argument vector, 3C-512
getpagesize — get system page size, 3C-515
getpass — read a password, 3C-516
getpeername — get name of peer connected to socket, 3N-517
getpriority — get scheduling priority for process, process group or user, 3C-518
getpublickey — retrieve public or secret key, 3N-522
getpw — get passwd entry from UID, 3C-523
getpwent — get password entry from user database, 3C-524
getpwent_r — get password entry from user database, 3C-524
getpwname — get password entry from user data-
base, 3C-524
getpwnam_r — get password entry from user database, 3C-524
getpwuid — get password entry from user database, 3C-524
getpwuid_r — get password entry from user database, 3C-524
getrusage — get information about resource utilization, 3C-531
getsecretkey — retrieve public or secret key, 3N-522
getservbyname — get service entry, 3N-535
getservbyname_r — get service entry, 3N-535
getservbyport — get service entry, 3N-535
getservbyport_r — get service entry, 3N-535
getservent — get service entry, 3N-535
getservent_r — get service entry, 3N-535
getspent — get shadow password database entry, 3C-543
getspent_r — get shadow password database entry (reentrant), 3C-543
getspnam — get shadow password database entry, 3C-543
getspnam_r — get shadow password database entry (reentrant), 3C-543
getsubopt — parse suboptions from a string, 3C-547
gettext — message handling function, 3I-550
gettimeofday — get date and time, 3C-554, 3B-553
gettxt — retrieve a text string, 3C-555
getusershell() — get legal user shells, 3C-557
getutent — access utmp file entry, 3C-558
getutid — access utmp file entry, 3C-558
getutline — access utmp file entry, 3C-558
getutmp — access utmpx file entry, 3C-560
getutmph — access utmpx file entry, 3C-560
getutxent — access utmpx file entry, continued
getutxid(), 3C-560
getutxline(), 3C-560
pututxline(), 3C-560
setutxent(), 3C-560
updutmp(), 3C-560
updutmph(), 3C-560
updutmpx(), 3C-560
utmpxname(), 3C-560
getutxid — access utmpx file entry, 3C-560
getutxline — access utmpx file entry, 3C-560
getvfsany — get vfstab file entry, 3C-563
getvfsent — get vfstab file entry, 3C-563
getvfsfile — get vfstab file entry, 3C-563
getvfsaspec — get vfstab file entry, 3C-563
getwc — get EUC character from stream, 3I-565
getwchar — get (or push back) wchar_t characters from curses terminal keyboard, 3X-237
getwchar — get EUC character from stdin, 3I-565
getwd — get current working directory pathname, 3C-566
getwidth — get codeset information, 3I-567
gettext — convert a string of EUC characters from the stream to Process Code, 3I-568
getwstr — get wchar_t character strings from curses terminal keyboard, 3X-241
glob — generate path names matching a pattern, 3C-569
globfree — generate path names matching a pattern, 3C-569
gmatch — shell global pattern matching, 3G-573
grantpt — grant access to the slave pseudo-terminal device, 3C-574

graphics interface
    — arc, 3-817
    — box, 3-817
    — circle, 3-817
    — closepl, 3-817
    — closevt, 3-817
    — cont, 3-817
    — erase, 3-817
    — label, 3-817
    — line, 3-817
    — linmod, 3-817

Index—21
graphics interface, continued
   — move, 3-817
   — openpl, 3-817
   — openvt, 3-817
   — plot, 3-817
   — point, 3-817
   — space, 3-817

group IDs
   set terminal foreground process group id —
      tcsetpgrp, 3C-1136

group IDs, supplementary
   initialize — initgroups, 3C-590

groups
   — endgrent, 3C-489
   — fgetgrent, 3C-489
   — fgetgrent_r, 3C-489
   — getgrent, 3C-489
   — getgrent_r, 3C-489
   — getgrgid, 3C-489
   — getgrgid_r, 3C-489
   — getgrnam, 3C-489
   — getgrnam_r, 3C-489
   — setgrent, 3C-489

H

halt system processor
   — reboot, 3C-898

hash-table search routine
   — hsearch, 3C-575

hasmntopt — get mnttab file information, 3C-501

have Volume Management check for media —
   volmgt_check, 3X-1179

hcreate — create hash table, 3C-575

hdestroy — destroy hash table, 3C-575

host ID
   get unique identifier of current host —
      gethostid, 3C-497

host machines, remote
   return information about users — rusers,
      rnusers, 3N-966

host name
   get name of current host — gethostname,
      3C-498
   set name of current host — sethostname,
      3C-498

host name, continued

hsearch — hash-table search routine, 3C-575

hyperbolic functions
   — acosh, 3M-578
   — asinh, 3M-578
   — atanh, 3M-578
   — cosh, 3M-578
   — sinh, 3M-578
   — tanh, 3M-578

hypot — Euclidean distance function, 3M-579

I

I/O asynchronous
   read and write operations — aio_read,
      aio_write, 3R-144

I/O multiplexing, synchronous
   — select, 3C-987

I/O package
   standard buffered I/O — stdio, 3S-1050

I/O, asynchronous
   cancel request — aio_cancel, 3R-140
   file synchronization — aio_sync, 3R-142
   retrieve error status — aio_error, 3R-146
   retrieve return status — aio_return, 3R-146
   wait for request — aio_suspend, 3R-149

I/O, requests
   list — lio_listio, 3R-645

iconv — code conversion function, 3-580

iconv_close — code conversion deallocation
   function, 3-582

iconv_open — code conversion allocation func-
   tion, 3-583

IEEE arithmetic
   convert floating-point number to string —
      ecvt, 3C-326

IEEE floating-point environment control
   — fpgetmks, 3C-457
   — fpgetround, 3C-457
   — fpgetsticky, 3C-457
   — fpsetmask, 3C-457
   — fpsetround, 3C-457
   — fpsetsticky, 3C-457

ilogb() function, 3M-584
index — string operations, 3C-587
inet — Internet address manipulation, 3N-588
inet_addr — Internet address manipulation, 3N-588
inet_lnaof — Internet address manipulation, 3N-588
inet_makeaddr — Internet address manipulation, 3N-588
inet_neto — Internet address manipulation, 3N-588
inet_network — Internet address manipulation, 3N-588
inet_ntoa — Internet address manipulation, 3N-588
initgroups — initialize the supplementary group access list, 3C-590
initialize kernel statistics facility
— kstat_close, 3K-620
— kstat_open, 3K-620
initstate — routines for changing random number generators, 3C-890
innwstr — get a string of wchar_t characters from a curses window, 3X-258
input conversion
— convert from wchar_t string — wscanf, 3I-1216
input/output package
standard buffered I/O — stdio, 3S-1050
insert a wchar_t character before the character under the cursor in a curses window —
curs_inschwch, 3X-253
inschwch, 3X-253
mvinschwch, 3X-253
mvwinschwch, 3X-253
winschwch, 3X-253
insert wchar_t string before character under the cursor in a curses window —
curs_inswwstr, continued
mvwwinswwstr, 3X-254
wwinswwstr, 3X-254
wwinswwstr, 3X-254
inswwstr — insert wchar_t string before character under the cursor in a curses window, 3X-254
insque — insert element to a queue, 3C-591
inswwch — insert a wchar_t character before the character under the cursor in a curses window, 3X-253
inswwstr — insert wchar_t string before character under the cursor in a curses window, 3X-254
interface to libthread threads information
— libthread_db, 3T-637
— td_init, 3T-637
— td_log, 3T-637
— td_ta_delete, 3T-637
— td_ta_get_nthreads, 3T-637
— td_ta_get_ph, 3T-637
— td_ta_map_id2thr, 3T-637
— td_ta_map_lwp2thr, 3T-637
— td_ta_new, 3T-637
— td_ta_thr_iter, 3T-637
— td_ta_tsd_iter, 3T-637
— td_thr_get_info, 3T-637
— td_thr_getfpregs, 3T-637
— td_thr_getgregs, 3T-637
— td_thr_getxregs, 3T-637
— td_thr_getxregsize, 3T-637
— td_thr_setfpregs, 3T-637
— td_thr_setgregs, 3T-637
— td_thr_setxregs, 3T-637
— td_thr_setxregsize, 3T-637
— td_thr_setprio, 3T-637
— td_thr_setsigpending, 3T-637
— td_thr_setxregs, 3T-637
— td_thr_sigsetmask, 3T-637
— td_thr_tsd, 3T-637
— td_thr_validate, 3T-637
Internet address manipulation — inet, 3N-588
inet_addr, 3N-588
inet_lnaof, 3N-588
inet_makeaddr, 3N-588
inet_neto, 3N-588
inet_network, 3N-588
Internet address manipulation — inet, continued
  inet_network, 3N-588
  inet_ntoa, 3N-588

Interprocess Communication
  create a new FIFO — mkfifo, 3C-711
interprocess communication
  standard package — ftok, 3C-1054
inwch — get a wchar_t character and its attributes
  from a curses window, 3X-256
inwchnstr — get a string of wchar_t characters
  (and attributes) from a curses window, 3X-257
inwchstr — get a string of wchar_t characters
  (and attributes) from a curses window, 3X-257
inwchar — get a string of wchar_t characters from a
curses window, 3X-258
isalnum — character handling, 3C-208
isalpha — character handling, 3C-208
isascii — character handling, 3C-208
isatty — find name of a terminal, 3C-1170
isdigit — character handling, 3C-208
isencrypt — determine whether a buffer of char-
  acters is encrypted, 3G-593
isenglish — Process Code character classification
  macros and functions, 3I-595
isgraph — character handling, 3C-208
isideogram — Process Code character
  classification macros and functions, 3I-595
islower — character handling, 3C-208
isnan() function, 3M-584
isnumber — Process Code character classification
  macros and functions, 3I-595
isphonogram — Process Code character
  classification macros and functions, 3I-595
isprint — character handling, 3C-208
ispunct — character handling, 3C-208
isspace — character handling, 3C-208
isspecial — Process Code character classification
  macros and functions, 3I-595
issupper — character handling, 3C-208
iswalnum — Process Code character classification
  macros and functions, 3I-595
iswalpha — Process Code character classification
  macros and functions, 3I-595
iswascii — Process Code character classification
  macros and functions, 3I-595
iswcntrol — Process Code character classification
  macros and functions, 3I-595
iswctype — test character for specified class,
  3I-597
iswddigit — Process Code character classification
  macros and functions, 3I-595
iswgraph — Process Code character classification
  macros and functions, 3I-595
iswlower — Process Code character classification
  macros and functions, 3I-595
iswprint — Process Code character classification
  macros and functions, 3I-595
iswpunct — Process Code character classification
  macros and functions, 3I-595
iswspace — Process Code character classification
  macros and functions, 3I-595
iswupper — Process Code character classification
  macros and functions, 3I-595
iswxdigit — Process Code character classification
  macros and functions, 3I-595
isxdigit — character handling, 3C-208

J
j0 — Bessel function, 3M-165
j1 — Bessel function, 3M-165
jn — Bessel function, 3M-165

K
Kerberos authentication library
  — kerberos, 3N-598
    — krb_get_cred, 3N-598
    — krb_kntoln, 3N-598
    — krb_mk_err, 3N-598
    — krb_mk_req, 3N-598
    — krb_mk_safe, 3N-598
    — krb_rd_err, 3N-598
    — krb_rd_req, 3N-598
    — krb_rd_safe, 3N-598
    — krb_set_key, 3N-598
Kerberos authentication routines for RPC
Kerberos authentication routines for RPC, continued
  — authkerb_getucred, 3N-602
  — authkerb_seccreate, 3N-602
  — kerberos_rpc, 3N-602
  — svc_kerb_reg, 3N-602
Kerberos authentication routines via network stream sockets
  — krb_net_read, 3N-608
  — krb_net_write, 3N-608
  — krb_recauth, 3N-608
  — krb_sendauth, 3N-608
Kerberos ticket cache filename
  — krb_set_tkt_string, 3N-611
Kerberos utility routines
  — krb_get_admhst, 3N-606
  — krb_get_krbhst, 3N-606
  — krb_get_lrealm, 3N-606
  — krb_get_phost, 3N-606
  — krb_realmofhost, 3N-606
kernel virtual memory functions
  copy data from kernel image or running system
    — kvm_read, kvm_kread, kvm_uread, 3K-628
  copy data to kernel image or running system —
    kvm_write, kvm_kwrite, kvm_uwrite, 3K-628
  get entries from kernel symbol table —
    kvm_nlist, 3K-625
  get invocation argument for process —
    kvm_getcmd, 3K-622
  get u-area for process —
    kvm_getu, 3K-622
kstat — kernel statistics facility, 3K-612
kstat_chain_update — update the kstat header chain, 3K-618
kstat_close — initialize kernel statistics facility, 3K-620
kstat_data_lookup — find a kstat by name, 3K-619
kstat_lookup — find a kstat by name, 3K-619
kstat_open — initialize kernel statistics facility, 3K-620
kstat_read — read or write kstat data, 3K-621
kstat_write — read or write kstat data, 3K-621
kstat — kernel statistics facility, continued
  specify a kernel to examine —
    kvm_open, kvm_close, 3K-626
killpg — send signal to a process group, 3C-605
kstat — kernel statistics facility, 3K-612
kstat_chain_update — update the kstat header chain, 3K-618
kstat_close — initialize kernel statistics facility, 3K-620
kstat_data_lookup — find a kstat by name, 3K-619
kstat_lookup — find a kstat by name, 3K-619
kstat_open — initialize kernel statistics facility, 3K-620
kstat_read — read or write kstat data, 3K-621
kstat_write — read or write kstat data, 3K-621
kvm_close — specify kernel to examine, 3K-626
kvm_getcmd — get invocation arguments for process, 3K-622
kvm_getproc — read system process structures, 3K-624
kvm_getu — get u-area for process, 3K-622
kvm_kread — copy data from a kernel image or running system, 3K-628
kvm_write — copy data to a kernel image or running system, 3K-628
kvm_nlist — get entries from kernel symbol table, 3K-625
kvm_nextproc — read system process structures, 3K-624
kvm_open — specify kernel to examine, 3K-626
kvm_read — copy data from kernel image or running system, 3K-628
kvm_setproc — read system process structures, 3K-624
kvm_uread — copy data from a kernel image or running system, 3K-628
kvm_uwrite — copy data to a kernel image or running system, 3K-628
kvm_write — copy data to kernel image or running system, 3K-628

Index—25
library routines for dealing with creation and manipulation of CLIENT handles, continued
    — rpc_createerr, 3N-930
libthread_db — interface to libthread threads
    information, 3T-637
    Support Level, 3T-637
line — graphics interface, 3-817
linear search and update routine
    — lfind, 3C-655
    — lsearch, 3C-655
linmod — graphics interface, 3-817
lio_listio — list directed I/O, 3R-645
list directed I/O — lio_listio, 3R-645
listen — listen for connections on a socket,
    3N-648
llabs — return absolute value of long long integer,
    3C-127
lldiv — compute quotient and remainder, 3C-310
lfmt — display error message in standard format
    and pass to logging and monitoring services,
    3C-631
lgamma — log gamma function, 3M-635
lgamma_r — log gamma function, 3M-635
library routines for dealing with creation and manipulation of CLIENT handles
    — cint_call, 3N-927
    — cint_freeres, 3N-927
    — cint_geterr, 3N-927
    — cint_perrorno, 3N-927
    — cint_perror, 3N-927
    — cint_serrorno, 3N-927
    — cint_serror, 3N-927
    — rpc_broadcast, 3N-927
    — rpc_broadcast_exp, 3N-927
    — rpc_call, 3N-927
    — rpc_cint_calls, 3N-927
library routines for dealing with creation and manipulation of CLIENT handles
    — cint_control, 3N-930
    — cint_create, 3N-930
    — cint_create_timed, 3N-930
    — cint_create_vers, 3N-930
    — cint_destroy, 3N-930
    — cint_dg_create, 3N-930
    — cint_pcreateerror, 3N-930
    — cint_raw_create, 3N-930
    — cint_spcreateerror, 3N-930
    — cint_tli_create, 3N-930
    — cint_tp_create, 3N-930
    — cint_tp_create_timed, 3N-930
    — cint_vc_create, 3N-930
    — rpc_cint_create, 3N-930
lock
    apply or remove advisory lock on open file — flock, 3B-385
lock address space
    — mlockall, 3C-717
lock memory pages
    — mlock, 3C-715
lockfile for user’s mailbox
    — maillock, 3X-659
log — natural logarithm, 3M-375
log gamma function
    — gamma, 3M-635
    — gamma_r, 3M-635
    — lgamma, 3M-635
    — lgamma_r, 3M-635
log10 — logarithm, base 10, 3M-375
log1p — natural logarithm of 1 plus argument, 3M-375
logarithm, base 10 — log10, 3M-375

Index—26
logarithm, natural — log, 3M-375
logb() function, 3M-586
login name
— getlogin, 3C-500
— getlogin_r, 3C-500
longjmp — non-local goto, 3B-1008, 3C-1011

M
madvise — provide advice to VM system, 3-657
maillock — manage lockfile for user’s mailbox, 3X-659
makecontext — manipulate user contexts, 3C-661
malloc — memory allocator, 3X-169
manipulate sets of signals — sigsetops, 3C-1031
    sigaddset, 3C-1031
    sigdelset, 3C-1031
    sigemptyset, 3C-1031
    sigfillset, 3C-1031
    sigismember, 3C-1031
match filename or path name — fnmatch, 3C-422
math library exception-handling — matherr, 3M-670
mathematical functions
    — acos, 3M-1164
    — acosh, 3M-578
    — asin, 3M-1164
    — asinh, 3M-578
    — atan, 3M-1164
    — atan2, 3M-1164
    — atanh, 3M-578
    — cbrt, 3M-1048
    — ceil, 3M-389
    — cos, 3M-1164
    — cosh, 3M-578
    — exp, 3M-375
    — expm1, 3M-375
    — floor, 3M-389
    — gamma, 3M-635
    — gamma_r, 3M-635
    — hypot, 3M-579
    — j0, 3M-165
    — j1, 3M-165
    — jn, 3M-165
    — lgamma, 3M-635
mathematical functions, continued
    — lgamma_r, 3M-635
    — log, 3M-375
    — log10, 3M-375
    — loglp, 3M-375
    — pow, 3M-375
    — rint, 3M-389
    — sin, 3M-1164
    — sinh, 3M-578
    — sqrt, 3M-1048
    — tan, 3M-1164
    — tanh, 3M-578
    — y0, 3M-165
    — y1, 3M-165
    — yn, 3M-165
matherr — math library exception-handling, 3M-670
mbstowcs — multibyte string functions, 3C-677
mctl — memory management control, 3B-678
media_findname — convert a supplied name into
    an absolute pathname that can be used to
    access removable media, 3X-680
media_getattr — get and set media attributes, 3X-682
media_setattr — get and set media attributes, 3X-682
memory — memory operations, 3C-684
    optimizing usage of user mapped memory —
    madvise, 3-657
memory allocator — bsdmalloc, 3X-169
    — alloca, 3C-663
    — calloc, 3C-663, 3X-665, 3X-668
    — free, 3C-663, 3X-665, 3X-668, 3X-169
    — mallinfo, 3X-665
    — malloc, 3C-663, 3X-665, 3X-668, 3X-169
    — mallocpt, 3X-665
    — memalign, 3C-663
    — realloc, 3C-663, 3X-665, 3X-668, 3X-169
    — valloc, 3C-663
memory lock or unlock
    calling process — plock, 3C-816
memory management — mctl, 3B-678
    copy a file into memory — copylist, 3G-200
    get system page size — getpagesize, 3C-515

Index–27
memory management — mctl, continued
lock address space — mlockall, 3C-717
lock pages in memory — mlock, 3C-715
synchronize memory with physical storage —
msync, 3C-733
unlock address space — munlockall, 3C-717
unlock pages in memory — munlock, 3C-715
memory object, shared
open — shm_open, 3R-1017
remove — shm_unlink, 3R-1019
memory operations
— memccpy, 3C-684
— memchr, 3C-684
— memcmp, 3C-684
— memcpy, 3C-684
— memmove, 3C-684
— memset, 3C-684
menu library, See also curses library
menus — character based menus package, 3X-707
menus cursor
— menu_cursor, 3X-686
— pos_menu_cursor, 3X-686
menus display attributes
— menu_attributes, 3X-685
— menu_back, 3X-685
— menu_fore, 3X-685
— menu_grey, 3X-685
— menu_pad, 3X-685
— set_menu_back, 3X-685
— set_menu_fore, 3X-685
— set_menu_grey, 3X-685
— set_menu_pad, 3X-685
menus from associated subwindows, write/erase
— menu_post, 3X-704
— post_menu, 3X-704
— unpost_menu, 3X-704
menus item name and description
— item_description, 3X-693
— item_name, 3X-693
— menu_item_name, 3X-693
menus item options routines
— item_opts, 3X-695
— item_opts_off, 3X-695
— item_opts_on, 3X-695
menus item options routines, continued
— menu_item_opts, 3X-695
— set_item_opts, 3X-695
menus item values, set and get
— item_value, 3X-697
— menu_item_value, 3X-697
— set_item_value, 3X-697
menus item, visibility
— item_visible, 3X-698
— menu_item_visible, 3X-698
menus items, associate application data
— item_userptr, 3X-696
— menu_item_userptr, 3X-696
— set_item_userptr, 3X-696
menus items, connect and disconnect
— item_count, 3X-699
— menu_items, 3X-699
— set_menu_items, 3X-699
menus items, create and destroy
— free_item, 3X-694
— menu_item_new, 3X-694
— new_item, 3X-694
menus items, get and set
— current_item, 3X-692
— item_index, 3X-692
— menu_item_current, 3X-692
— set_current_item, 3X-692
— set_top_row, 3X-692
— top_row, 3X-692
menus mark string routines
— menu_mark, 3X-700
— set_menu_mark, 3X-700
menus options routines
— menu_opts, 3X-702
— menu_opts_off, 3X-702
— menu_opts_on, 3X-702
— set_menu_opts, 3X-702
menus pattern match buffer
— menu_pattern, 3X-703
— set_menu_pattern, 3X-703
menus subsystem, command processor
— menu_driver, 3X-687
menus window and subwindow association rou-
tines
Index—28
menus, window and subwindow association routines, continued

- menu_sub, 3X-706
- menu_win, 3X-706
- scale_menu, 3X-706
- set_menu_sub, 3X-706
- set_menu_win, 3X-706

menus, application-specific routines

- item_init, 3X-690
- item_term, 3X-690
- menu_hook, 3X-690
- menu_init, 3X-690
- menu_term, 3X-690
- set_item_init, 3X-690
- set_item_term, 3X-690
- set_menu_init, 3X-690
- set_menu_term, 3X-690

menus, associate application data

- menu_userptr, 3X-705
- set_menu_userptr, 3X-705

menus, create and destroy

- free_menu, 3X-701
- menu_new, 3X-701
- new_menu, 3X-701

menus, rows and columns

- menu_format, 3X-689
- set_menu_format, 3X-689

message catalog

- open/catalog — catopen, catclose, 3C-184
- read a program message — catgets, 3C-183

message handling functions

- bindtextdomain, 3I-550
- dcgettext, 3I-550
- dgettext, 3I-550
- gettext, 3I-550
- textdomain, 3I-550

message queue

- close — mq_close, 3R-720
- get attributes — mq_getattr, 3R-730
- notify process (or thread) — mq_notify, 3R-721
- open — mq_open, 3R-723
- receive a message from — mq_receive, 3R-726

message queue, continued

- remove — mq_unlink, 3R-732
- send message to — mq_send, 3R-728
- set attributes — mq_setattr, 3R-730

messages

- display a message on stderr or system console
  — fmtmsg, 3C-390
- print system error messages — perror, 3C-812

system signal messages — psignal, 3C-834

mkdirp — create directories in a path, 3G-710
mkfifo — create a new FIFO, 3C-711
mktemp — make a unique file name, 3C-712
mkt ime — converts a tm structure to a calendar time, 3C-713

mnttab file

- getmntany, 3C-501
- getmntent, 3C-501
- hasmntopt, 3C-501
- putmntent, 3C-501

monitor — prepare process execution profile, 3C-718

move — graphics interface, 3-817
movenextch — moving the cursor by character, 3X-219
moveprevch — moving the cursor by character, 3X-219

mkdirp — create directories in a path, 3G-710
mkfifo — create a new FIFO, 3C-711
mktemp — make a unique file name, 3C-712
mkt ime — converts a tm structure to a calendar time, 3C-713

msync — synchronize memory with physical storage, 3C-733

multibyte character handling

Index—29
multibyte character handling, continued  
— mbchar, 3C-675  
— mblen, 3C-675  
— mbtowc, 3C-675  
— wctomb, 3C-675  

multibyte string functions — mbstring, 3C-677  
mbstowcs, 3C-677  
wstombs, 3C-677  

mvaddnwstr — add a string of wchar_t characters to a curses window and advance cursor, 3X-218  
mvaddwch — add a wchar_t character (with attributes) to a curses window and advance cursor, 3X-214  
mvaddwchnstr — add string of wchar_t characters (and attributes) to a curses window, 3X-216  
mvaddwchstr — add string of wchar_t characters (and attributes) to a curses window, 3X-216  
mvaddwstr — add a string of wchar_t characters to a curses window and advance cursor, 3X-218  
mvgetnwstr — get wchar_t character strings from curses terminal keyboard, 3X-241  
mvwgetnwstr — get wchar_t character strings from a curses window, 3X-258  
mvwaddnwstr — add a string of wchar_t characters from a curses window and advance cursor, 3X-218  
mvwaddwch — add a wchar_t character (with attributes) to a curses window and advance cursor, 3X-214  
mvwaddwchnstr — add string of wchar_t characters (and attributes) to a curses window, 3X-216  
mvwaddwchstr — add string of wchar_t characters (and attributes) to a curses window, 3X-216  
mvwaddwstr — add a string of wchar_t characters to a curses window and advance cursor, 3X-218  
mvwgetnwstr — get wchar_t character strings from curses terminal keyboard, 3X-241  
mvwgetwch — get (or push back) wchar_t characters from curses terminal keyboard, 3X-237  
mvwgetwstr — get wchar_t character strings from curses terminal keyboard, 3X-241  
mvwinnwstr — get a string of wchar_t characters from a curses window, 3X-258  
mvwinsnwstr — insert wchar_t string before character under the cursor in a curses window, 3X-254  
mvwinswch — insert a wchar_t character before the character under the cursor in a curses window, 3X-253  
mvwinswstr — insert wchar_t string before character under the cursor in a curses window, 3X-254  
mvinswchnstr — get a string of wchar_t characters (and attributes) from a curses window, 3X-256  
mvinswchstr — get a string of wchar_t characters (and attributes) from a curses window, 3X-257  
mvinswstr — get a string of wchar_t characters (and attributes) from a curses window, 3X-257  
mvwinwchnstr — get a string of wchar_t characters (and attributes) from a curses window, 3X-257  
mvwinwstr — get a string of wchar_t characters from a curses window, 3X-258  
mvwinwch — get a wchar_t character and its attributes from a curses window, 3X-256  
mvwinwchstr — get a string of wchar_t characters (and attributes) from a curses window, 3X-257  
mvwinwstr — get a string of wchar_t characters (and attributes) from a curses window, 3X-257  
mvwinwstr — get a string of wchar_t characters from a curses window, 3X-258
named pipe
   create a new one — mkfifo, 3C-711
nanosleep — high resolution sleep, 3R-745
natural logarithm — log, 3M-375
natural logarithm of 1 plus argument — log1p, 3M-375
ndbm — database subroutines, 3-746
netdir — generic transport name-to-address translation, 3N-749
netdir_free — generic transport name-to-address translation, 3N-749
netdir_getbyaddr — generic transport name-to-address translation, 3N-749
netdir_getbyname — generic transport name-to-address translation, 3N-749
netdir_mergeaddr — generic transport name-to-address translation, 3N-749
netdir_options — generic transport name-to-address translation, 3N-749
netdir_perror — generic transport name-to-address translation, 3N-749
netdir_sperror — generic transport name-to-address translation, 3N-749
network configuration database entry
   — endnetconfig, 3N-506
   — freenetconfig, 3N-506
   — getnetconfig, 3N-506
   — getnetconfig, 3N-506
   — nc_perror, 3N-506
   — nc_sperror, 3N-506
   — setnetconfig, 3N-506
network configuration entry corresponding to NET-PATH
   — endnetpath, 3N-510
   — getnetpath, 3N-510
   — setnetpath, 3N-510
network entry
   — endnetent, 3N-503
   — getnetentaddr, 3N-503
   — getnetentaddr_r, 3N-503
   — getnetbyname, 3N-503
   — getnetbyname_r, 3N-503
network entry, continued
   — getnetent, 3N-503
   — getnetent_r, 3N-503
   — setnetent, 3N-503
network group entry
   — endnetgrent, 3N-508
   — getnetgrent, 3N-508
   — getnetgrent_r, 3N-508
   — innetgr, 3N-508
   — setnetgrent, 3N-508
network host entry
   — endhostent, 3N-492
   — gethostbyaddr, 3N-492
   — gethostbyaddr_r, 3N-492
   — gethostbyname, 3N-492
   — gethostbyname_r, 3N-492
   — gethostent, 3N-492
   — gethostent_r, 3N-492
   — sethostent, 3N-492
network listener service
   format and send listener service request message — nlsrequest, 3N-795
   get client’s data passed via the listener —
      nlsgetcall, 3N-793
   get name of transport provider — nlsprovider, 3N-794
network protocol entry
   — endprotoent, 3N-519
   — getprotobyname, 3N-519
   — getprotobyname_r, 3N-519
   — getprotobynumber, 3N-519
   — getprotobynumber_r, 3N-519
   — getprotoent, 3N-519
   — getprotoent_r, 3N-519
   — setprotoent, 3N-519
newpad — create and display curses pads, 3X-265
nextafter() function, 3M-584
nextkey — database subroutines, 3B-298
nftw — walk a file tree, 3C-467
nice — change priority of a process, 3B-753
NIS client interface
   — yp_all, 3N-1245
   — yp_bind, 3N-1245
   — yp_first, 3N-1245
NIS client interface, continued
  — yp_get_default_domain, 3N-1245
  — yp_master, 3N-1245
  — yp_match, 3N-1245
  — yp_next, 3N-1245
  — yp_order, 3N-1245
  — yp_unbind, 3N-1245
  — ypclnt, 3N-1245
  — yperr_string, 3N-1245
  — yperrprot_err, 3N-1245

NIS+ database functions
  — db_add_entry, 3N-754
  — db_checkpoint, 3N-754
  — db_create_table, 3N-754
  — db_destroy_table, 3N-754
  — db_first_entry, 3N-754
  — db_free_result, 3N-754
  — db_initialize, 3N-754
  — db_list_entries, 3N-754
  — db_next_entry, 3N-754
  — db_remove_entry, 3N-754
  — db_reset_next_entry, 3N-754
  — db_standby, 3N-754
  — db_table_exists, 3N-754
  — db_unload_table, 3N-754
  — nis_db, 3N-754

NIS+ error messages
  — nis_error, 3N-758
  — nis_lerror, 3N-758
  — nis_serror, 3N-758
  — nis_sperror, 3N-758
  — nis_serror_r, 3N-758

NIS+ group manipulation functions
  — nis_addmember, 3N-759
  — nis_creategroup, 3N-759
  — nis_destroygroup, 3N-759
  — nis_groups, 3N-759
  — nis_ismember, 3N-759
  — nis_print_group_entry, 3N-759
  — nis_removemember, 3N-759
  — nis_verifygroup, 3N-759

NIS+ local names
  — nis_freenames, 3N-780
  — nis_getnames, 3N-780

NIS+ local names, continued
  — nis_local_directory, 3N-762
  — nis_local_group, 3N-762
  — nis_local_host, 3N-762
  — nis_local_names, 3N-762
  — nis_local_principal, 3N-762

NIS+ log administration functions
  — nis_checkpoint, 3N-777
  — nis_ping, 3N-777

NIS+ miscellaneous functions
  — nis_freeservelist, 3N-778
  — nis_freetags, 3N-778
  — nis_getservlist, 3N-778
  — nis_mkdir, 3N-778
  — nis_rrmdir, 3N-778
  — nis_server, 3N-778
  — nis_servstate, 3N-778
  — nis_stats, 3N-778

NIS+ namespace functions
  — nis_add, 3N-763
  — nis_freesresult, 3N-763
  — nis_lookup, 3N-763
  — nis_modify, 3N-763
  — nis_name, 3N-763
  — nis_remove, 3N-763

NIS+ object formats
  — nis_objects, 3N-768

NIS+ subroutines
  — nis_clone_object, 3N-780
  — nis_destroy_object, 3N-780
  — nis_dir_cmp, 3N-780
  — nis_domain_of, 3N-780
  — nis_leaf_of, 3N-780
  — nis_name_of, 3N-780
  — nis_print_object, 3N-780
  — nis_subr, 3N-780

NIS+ table functions
  — nis_add_entry, 3N-782
  — nis_first_entry, 3N-782
  — nis_list, 3N-782
  — nis_modify_entry, 3N-782
  — nis_next_entry, 3N-782
  — nis_remove_entry, 3N-782
  — nis_tables, 3N-782

NIS, change information
NIS, change information, continued
   — yp_update, 3N-1244
nis_tables — NIS+ table functions, 3N-782
nis_tables — NIS+ table functions, 3N-782
nis_tables — NIS+ table functions, 3N-782
nlist — get entries from symbol table, 3B-791
non-local goto — setjmp, 3B-1008, 3C-1011
   _longjmp, 3B-1008
   _setjmp, 3B-1008
   longjmp, 3B-1008, 3C-1011
   siglongjmp, 3C-1011
   sigsetjmp, 3C-1011
NOTE — annotate source code with info for tools, 3X-116
   NOTE vs _NOTE, 3X-117
NoteInfo Argument, 3X-117
numbers, convert to strings — econvert, 3-324

O
offsetof — offset of structure member, 3C-797
openlog — initialize system log file, 3-1086
openpl — graphics interface, 3-817
openvt — graphics interface, 3-817
output conversion
   wsprintf — convert to wchar_t string, 3I-1215
output conversion, formatted
   — fprintf, 3B-821
   — printf, 3B-821
   — sprintf, 3B-821
   — vsprintf, 3B-821
   — vprintf, 3B-821
   — vsprintf, 3B-821

P
p2close — close pipes to and from a command, 3G-798
p2open — open pipes to and from a command, 3G-798
page size, system
   get — getpagesize, 3C-515
panel library, See also curses library
panels — character based panels package, 3X-808
panels deck manipulation routines, continued
   — bottom_panel, 3X-804
   — hide_panel, 3X-803
   — panel_hidden, 3X-803
   — panel_top, 3X-804
   — panel_window, 3X-807
   — replace_panel, 3X-807
panels deck traversal primitives
   — panel_above, 3X-800
   — panel_bottom, 3X-800
panels panel, associate application data
   — panel_userptr, 3X-806
   — set_panel_userptr, 3X-806
panels panel, get or set current window
   — panel_window, 3X-807
   — replace_panel, 3X-807
panels virtual screen refresh routine
   — panel_update, 3X-805
   — update_panel, 3X-805
panels window on virtual screen, move
   — move_panel, 3X-801
   — panel_move, 3X-801
panels, create and destroy
   — del_panel, 3X-802
   — new_panel, 3X-802
   — panel_new, 3X-802
password
   — getpass, 3C-516
password databases
   lock the lock file — lckpwdf, 3C-630
   unlock the lock file — ulckpwdf, 3C-630
password encryption
   — crypt, 3C-201
   — encrypt, 3C-201
   — setkey, 3C-201
passwords
   get passwd entry from UID — getpw, 3C-523
   get password entry from a file — fgetpwent, 3C-524
   write password file entry — putpwent, 3C-880
passwords, shadow
   get shadow password database entry —
endspent, passwords, shadow, continued
3C-543
get shadow password database entry (reentrant) — fgetsspent_r, 3C-543
write shadow password file entry —
putspent, 3C-882
path name
return last element — path name, 3G-164
pathfind — search for named file in named directories, 3G-810
pclose — initiate pipe to/from a process, 3S-819
pechochar — create and display curses pads, 3X-265
pechowchar — create and display curses pads, 3X-265
perform word expansions
— wordexp, 3C-1212
— wordfree, 3C-1212
perror — print system error messages, 3C-812
pfmt — display error message in standard format, 3C-813
pipes
initiate to/from a process — pclose, 3S-819
open, close to and from a command —
p2open, p2close, 3G-798
plock — lock or unlock into memory process, text, or data, 3C-816
plot — graphics interface, 3-817
Link Editor, 3-818
pnoutrefresh — create and display curses pads, 3X-265
point — graphics interface, 3-817
popen — initiate pipe to/from a process, 3S-819
pow — raise to power, 3M-375
prefresh — create and display curses pads, 3X-265
print formatted output
— fprintf, 3S-825
— printf, 3S-825
— sprintf, 3S-825
print formatted output of a variable argument list
— vfprintf, 3S-1187
— vsprintf, 3S-1187
printf — formatted output conversion, 3B-821, 3S-825
probe insertion interface — TNF_PROBE, 3X-121
proc_service — process service interface, 3T-831
Process Code character classification macros and functions
— isenglish, 3I-595
— isideogram, 3I-595
— isnumber, 3I-595
— isphonogram, 3I-595
— isspecial, 3I-595
— iswalnum, 3I-595
— iswalpha, 3I-595
— iswascii, 3I-595
— iswcntrl, 3I-595
— iswdigit, 3I-595
— iswgraph, 3I-595
— iswlower, 3I-595
— iswprint, 3I-595
— iswpunct, 3I-595
— iswspace, 3I-595
— iswupper, 3I-595
— iswxdigit, 3I-595
Process Code character conversion macros
— tolower, 3I-1197
— toupper, 3I-1197
— wconv, 3I-1197
Process Code string operations — wstring, 3I-1217
— wcasecmp, 3I-1217
— wcscol, 3I-1217
— wscdup, 3I-1217
— wcsncasecmp, 3I-1217
process service interface
— proc_service, 3T-831
— ps_lcontinue, 3T-831
— ps_lgetfregs, 3T-831
— ps_lgetregs, 3T-831
— ps_lgetxregs, 3T-831
— ps_lgetxregsize, 3T-831
— ps_lsetfregs, 3T-831
process service interface, continued
   — ps_lsetfpregs, 3T-831
   — ps_lsetxregs, 3T-831
   — ps_lstop, 3T-831
   — ps_pcontinue, 3T-831
   — ps_pdread, 3T-831
   — ps_pdwrite, 3T-831
   — ps_pglobal_lookup, 3T-831
   — ps_plog, 3T-831
   — ps_pstop, 3T-831
   — ps_ptread, 3T-831
   — ps_ptwrite, 3T-831
process statistics
   prepare execution profile — monitor, 3C-718
processes
   change priority — nice, 3B-753
   duplicate an open file descriptor — dup2, 3C-323
   generate path name for controlling terminal —
   ctermid, ctermid_r, 3S-203
   get character-string representation —
   cuserid, 3S-297
   initiate pipe to/from a process — popen, pclose, 3S-819
   manipulate user contexts — makecontext, swapcontext, 3C-661
   memory lock or unlock — plock, 3C-816
   prepare execution profile — monitor, 3C-718
   report CPU time used — clock, 3C-186
   send signal to a process group — killpg, 3C-605
   send signal to program — raise, 3C-887
   set terminal foreground process group id —
   tcsetpgid, 3C-1136
   suspend execution for interval — sleep,
   3B-1040, 3C-1041
   terminate process — exit, 3C-374
   terminate the process abnormally — abort,
   3C-126
   wait for process to terminate or stop —
   WIFEXITED, 3B-1190
profiling utilities
   prepare process execution profile — monitor,
   3C-718
program assertion, continued
   verify — assert, 3C-155
program messages
   open/close a message catalog — catopen,
   catclose, 3C-184
   read — catgets, 3C-183
programs
   last locations — end, etext, edata, 3C-369
   ps_lcontinue — process service interface, 3T-831
   ps_lgetfpregs — process service interface,
   3T-831
   ps_lgetregs — process service interface, 3T-831
   ps_lgetxregs — process service interface, 3T-831
   ps_lgetxregsize — process service interface,
   3T-831
   ps_lsetfpregs — process service interface,
   3T-831
   ps_lsetregs — process service interface, 3T-831
   ps_lsetxregs — process service interface, 3T-831
   ps_lstop — process service interface, 3T-831
   ps_pcontinue — process service interface, 3T-831
   ps_pdread — process service interface, 3T-831
   ps_pdwrite — process service interface, 3T-831
   ps_pglobal_lookup — process service interface,
   3T-831
   ps_plog — process service interface, 3T-831
   ps_pstop — process service interface, 3T-831
   ps_ptread — process service interface, 3T-831
   ps_ptwrite — process service interface, 3T-831
pseudo-terminal device
   get name of the slave pseudo-terminal device —
   ptsname, 3C-876
   grant access to the slave pseudo-terminal dev-
   ice — grantpt, 3C-574
psiginfo — system signal messages, 3C-834
psignal — system signal messages, 3B-833, 3C-834
pthread_atfork — register fork handlers, 3T-835
pthread_getspecific — thread-specific-data
   functions, 3T-858
pthread_key_create — thread-specific-data
   functions, 3T-858
Create Key, 3T-858
POSIX, 3T-859

Index—35
pthreads_key_create — thread-specific-data functions, continued
POSIX Delete Key, 3T-859
POSIX Get Value, 3T-859
Set Value, 3T-858
Solaris, 3T-858
Solaris Get Value, 3T-859
pthreads_key_delete — thread-specific-data functions, 3T-858
pthreads_setspecific — thread-specific-data functions, 3T-858
ptsname — get name of the slave pseudo-terminal device, 3C-876
publickey — retrieve public or secret key, 3N-522
putenv — change or add value to environment, 3C-879
putmntent — get mnttab file information, 3C-501
putpwent — write password file entry, 3C-880
putspent — write shadow password file entry, 3C-882
pututline — access utmp file entry, 3C-558
pututxline — access utmpx file entry, 3C-560
putwc — put wide character on stream, 3I-883
putwchar — put wide character on stdout, 3I-883
putws — convert a string of Process Code characters to EUC characters and put it on a stream, 3I-884
q gconvert — convert number to ASCII, 3-324
qfconvert — convert number to ASCII, 3-324
qgconvert — convert number to ASCII, 3-324
qsort — quick sort, 3C-885
quadruple_to_decimal — decimal record from quadruple-precision floating, 3-383
queues
insert/remove element from a queue —
insque, remque, 3C-591
random number generator
changing generators — initstate, setstate, 3C-890
— drand48, 3C-321
— erand48, 3C-321
— jrand48, 3C-321
— lcong48, 3C-321
— irand48, 3C-321
— mrand48, 3C-321
— nrand48, 3C-321
— rand, 3B-888
— random, 3C-890
— seed48, 3C-321
— srand48, 3C-321
random number generator, simple
— rand, 3C-889
— srand, 3C-889
rcmd — execute command remotely, 3N-892
read a directory entry — readdir, 3B-895
read and write a disk's VTOC — read_vtoc, 3X-894
write_vtoc, 3X-894
read or write kstat data
— kstat_read, 3K-621
— kstat_write, 3K-621
read system process structures
— kvm_getproc, 3K-624
— kvm_nextproc, 3K-624
— kvm_setproc, 3K-624
read_vtoc — read and write a disk's VTOC, 3X-894
readdir — read a directory entry, 3B-895
realloc — memory allocator, 3X-169
realpath — returns the real file name, 3C-897
reboot — reboot system or halt processor, 3C-898
receive a message from a socket — recv, 3N-899
— recvfrom, 3N-899
— recvmsg, 3N-899
recv — receive a message from a socket, 3N-899
Index—36
recvfrom — receive a message from a socket, 3N-899
recvmsg — receive a message from a socket, 3N-899
regcmp — compile regular expression, 3G-901
regcomp — regular expression matching, 3C-903
regerror — regular expression matching, 3C-903
regex — execute regular expression, 3G-901
regexec — regular expression matching, 3C-903
regexpr — regular expression compile and match routines, 3G-909
regfree — regular expression matching, 3C-903
register fork handlers — pthread_atfork, 3T-835
regular expression compile and match routines
— advance, 3G-909
— compile, 3G-909
— regex, 3G-909
— step, 3G-909
regular expression handler
— re_comp, 3C-908
— re_exec, 3C-908
— regex, 3C-908
regular expression matching
— regcomp, 3C-903
— regerror, 3C-903
— regexec, 3C-903
— regfree, 3C-903
regular expressions
compile and execute — regcmp, regex, 3G-901
remainder() function, 3M-584
remote command, return stream to
— rcmd, 3N-892
remote procedure calls, library routines for — rpc, 3N-917
remote system
return information about users — rusers, rnusers, 3N-966
write to — rstat, 3N-965, 3N-967
remove — remove file, 3C-912
remque — remove element from a queue, 3C-591
res_init — resolver routines, 3N-913
res_mkquery — resolver routines, 3N-913
res_search — resolver routines, 3N-913
res_send — resolver routines, 3N-913
resolver — resolver routines, 3N-913
resolver routines — resolver, 3N-913
dn_comp, 3N-913
dn_expand, 3N-913
res_init, 3N-913
res_mkquery, 3N-913
res_search, 3N-913
res_send, 3N-913
resource utilization
get information — getrusage, 3C-531
retrieve archive symbol table — elf_getarsym, 3E-351
retrieve public or secret key — getpublickey, 3N-522
getsecretkey, 3N-522
publickey, 3N-522
return physical memory information — sysmem, 3-1089
asysmem, 3-1089
return stream to a remote command — reexec, 3N-916
return the Volume Management root directory — volmgt_root, 3X-1181
return whether or not Volume Management is running — volmgt_running, 3X-1182
reexec — return stream to a remote command, 3N-916
rindex — string operations, 3C-587
rint — round to integral value, 3M-389
rmdirp — remove directories in a path, 3G-710
rnusers — return information about users on remote machines, 3N-966
rpc — library routines for remote procedure calls, 3N-917
RPC
data transmission using XDR routines — xdr, 3N-1218
RPC bind service library routines
— rpc_getmaps, 3N-963
— rpcb_getaddr, 3N-963
— rpcb_gettime, 3N-963

Index—37
RPC bind service library routines, continued
  — rpcb_rmtcall, 3N-963
  — rpcb_set, 3N-963
  — rpcb_unset, 3N-963
  — rpcbind, 3N-963

RPC entry
  — endrpcent, 3N-528
  — getrpcbyname, 3N-528
  — getrpcbyname_r, 3N-528
  — getrpcbynumber, 3N-528
  — getrpcbynumber_r, 3N-528
  — getrpcent, 3N-528
  — getrpcent_r, 3N-528
  — setrpcent, 3N-528

RPC library routine for manipulating global RPC attributes for client and server applications
  — rpc_control, 3N-935

RPC library routines for creation and manipulation of server handles
  — rpc_svc_create, 3N-954
  — svc_create, 3N-954
  — svc_destroy, 3N-954
  — svc_dg_create, 3N-954
  — svc_fd_create, 3N-954
  — svc_raw_create, 3N-954
  — svc_tli_create, 3N-954
  — svc_tp_create, 3N-954
  — svc_vc_create, 3N-954

RPC library routines for registering servers
  — rpc_reg, 3N-959
  — rpc_svc_reg, 3N-959
  — svc_auth_reg, 3N-959
  — svc_reg, 3N-959
  — svc_unreg, 3N-959
  — xprt_register, 3N-959
  — xprt_unregister, 3N-959

RPC library routines for server side errors
  — rpc_svc_err, 3N-957
  — svcerr_auth, 3N-957
  — svcerr_decode, 3N-957
  — svcerr_noproc, 3N-957
  — svcerr_noprogs, 3N-957
  — svcerr_progvers, 3N-957
  — svcerr_systemerr, 3N-957
  — svcerr_weakauth, 3N-957

RPC obsolete library routines
  — authdes_create, 3N-940
  — authunix_create_default, 3N-940
  — callrpc, 3N-940
  — clnt_broadcast, 3N-940
  — clnt_raw_create, 3N-940
  — clnttcp_create, 3N-940
  — clntudp_bufcreate, 3N-940
  — clntudp_create, 3N-940
  — get_myaddress, 3N-940
  — getrpcport, 3N-940
  — pmap_getmaps, 3N-940
  — pmap_getport, 3N-940
  — pmap_rmtcall, 3N-940
  — pmap_set, 3N-940
  — pmap_unset, 3N-940
  — registerrpc, 3N-940
  — rpc_soc, 3N-940
  — svc_fds, 3N-940
  — svc_getcaller, 3N-940
  — svc_getreq, 3N-940
  — svc_register, 3N-940
  — svc_unregister, 3N-940
  — svcfd_create, 3N-940
  — svcraw_create, 3N-940
  — svc_rcvcv_create, 3N-940
  — svcudp_bufcreate, 3N-940
  — svcudp_create, 3N-940
  — xdr_authunix_parms, 3N-940

rpc routines
  rac_drop() — remote asynchronous calls, 3N-937
  rac_poll() — remote asynchronous calls, 3N-937
  rac_recv() — remote asynchronous calls, 3N-937
  rac_send() — remote asynchronous calls, 3N-937

RPC servers library routines
  — rpc_svc_calls, 3N-950
  — svc_dg_enablecache, 3N-950
  — svc_done, 3N-950
  — svc_exit, 3N-950
  — svc_fdset, 3N-950
  — svc_freeargs, 3N-950

Index—38
RPC servers library routines, continued
— svc_getargs, 3N-950
— svc_getreq_common, 3N-950
— svc_getreq_poll, 3N-950
— svc_getreqset, 3N-950
— svc_getrpccaller, 3N-950
— svc_pollset, 3N-950
— svc_run, 3N-950
— svc_sendreply, 3N-950

RPC using Kerberos authentication routines
— authkerb_getucred, 3N-602
— authkerb_seccreate, 3N-602
— kerberos_rpc, 3N-602
— svc_kerb_reg, 3N-602

RPC, secure library routines
— authdes_getucred, 3N-983
— authdes_seccreate, 3N-983
— getnetname, 3N-983
— host2netname, 3N-983
— key_decryptsession, 3N-983
— key_encryptsession, 3N-983
— key_gendes, 3N-983
— key_secretkey_is_set, 3N-983
— key_setsecret, 3N-983
— netname2host, 3N-983
— netname2user, 3N-983
— secure_rpc, 3N-983
— user2netname, 3N-983

RPC, XDR library routines
— rpc_xdr, 3N-961
— xdr_accepted_reply, 3N-961
— xdr_authsys_parms, 3N-961
— xdr_callhdr, 3N-961
— xdr_callmsg, 3N-961
— xdrOpaque_auth, 3N-961
— xdr_rejected_reply, 3N-961
— xdr_replmsg, 3N-961

rpc_broadcast — library routines for client side calls, 3N-927
rpc_broadcast_exp — library routines for client side calls, 3N-927
rpc_call — library routines for client side calls, 3N-927
rpc_clnt_auth — library routines for client side remote procedure call authentication, 3N-925
rpc_clnt_calls — library routines for client side calls, 3N-927
Routines, 3N-927
rpc_clnt_create — library routines for dealing with creation and manipulation of CLIENT handles, 3N-930
Routines, 3N-930
rpc_createerr — library routines for dealing with creation and manipulation of CLIENT handles, 3N-930
rresvport — get privileged socket, 3N-892
rstat — get performance data from remote kernel, 3N-965
ruserok — authenticate user, 3N-892
ruusers — return information about users on remote machines, 3N-966
xdr_utmpidlearr, 3N-966
rw_rdlock() — acquire a read lock, 3T-968
rw_tryrdlock() — acquire a read lock, 3T-968
rw_trywrlock() — acquire a write lock, 3T-968
rw_unlock() — unlock a readers/writer lock, 3T-968
rw_wrlock() — acquire a write lock, 3T-968
rwlock() — write to specified remote machines, 3N-967
rwlock_destroy() — destroy a readers/writer lock, 3T-968
rwlock_init() — initialize a readers/writer lock, 3T-968

S
scalb() function, 3M-586
scalbn() function, 3M-584
scandir — scan a directory
— alphasort, 3B-971
— scandir, 3B-971
scandir — scan a directory, 3B-971
scanf — convert formatted input, 3S-972
sched_get_priority_max — get scheduling parameter limits, 3R-977
sched_get_priority_min — get scheduling parameter limits, 3R-977

Index—39
sched_getparam — get scheduling parameters, 3R-978
sched_getscheduler — get scheduling policy and scheduling parameters, 3R-980
sched_rr_get_interval — get scheduling parameter limits, 3R-977
sched_setparam — set scheduling parameters, 3R-978
sched_setscheduler — set scheduling policy and scheduling parameters, 3R-980
sched_yield — yield processor, 3R-982
scheduling parameters
set/get — sched_setparam, sched_getparam, 3R-978
scheduling parameters list
— sched_get_priority_max, 3R-977
— sched_get_priority_min, 3R-977
— sched_rr_get_interval, 3R-977
scheduling policy and parameters
set/get — sched_setscheduler, sched_getscheduler, 3R-980
scheduling priority
change priority of a process — nice, 3B-753
get/set for process, process group or user
getpriority, setpriority, 3C-518
search functions
linear search and update routine — lsearch, lfind, 3C-655
manage hash search tables — hsearch, 3C-575
search routines
binary search a sorted table — bsearch, 3C-171
seconvert — convert number to ASCII, 3-324
secure, RPC
See RPC, secure, 3N-983
select — synchronous I/O multiplexing, 3C-987
sem_close — close a named semaphore, 3R-989
sem_destroy — destroy an unnamed semaphore, 3R-990
sem_getvalue — get the value of a semaphore, 3R-991
sem_init — initialize an unnamed semaphore, 3R-992
sem_open — initialize/open a named semaphore, 3R-993
sem_post — increment the count of a semaphore, 3R-995
sem_trywait — acquire or wait for a semaphore, 3R-997
sem_unlink — remove a named semaphore, 3R-996
sem_wait — acquire or wait for a semaphore, 3R-997
sema_destroy() — destroy a semaphore, 3T-1000
sema_init() — initialize a semaphore, 3T-1000
sema_post() — increment a semaphore, 3T-1000
sema_trywait() — decrement a semaphore, 3T-1000
sema_wait() — decrement a semaphore, 3T-1000
semaphore
acquire or wait for — sem_wait, sem_trywait, 3R-997
close a named one — sem_close, 3R-989
destroy an unnamed one — sem_destroy, 3R-990
get the value — sem_getvalue, 3R-991
increment the count — sem_post, 3R-995
initialize an unnamed one — sem_init, 3R-992
initialize/open a named one — sem_open, 3R-993
remove a named one — sem_unlink, 3R-996
send — send message from a socket, 3N-1003
sendmsg — send message from a socket, 3N-1003
sendto — send message from a socket, 3N-1003
Service Access Facility library function
— doconfig, 3N-319
setac — get audit control file information, 3-469
setauclass — rewind audit_class database file, 3-471
setauuser — rewind audit_event database file, 3-474
setauuser — get audit_user database entry, 3-476
setcat — define default catalog, 3C-1007
setgrent — get group entry from database, 3C-489
sethostname — set name of current host, 3C-498
setjmp — non-local goto, 3B-1008, 3C-1011
setLabel — define the label for pfmt() and fmt(), 3C-1014
setlocale — modify and query a program's locale, 3C-1015
setlogmask — set log priority mask, 3-1086
setpriority — set scheduling priority for process, process group or user, 3C-518
setpwnam — get password entry from user database, 3C-524
setservent — get service entry, 3N-535
setspent — get shadow password database entry, 3C-543
setState — routines for changing random number generators, 3C-890
settimeofday — set date and time, 3C-554, 3B-553
setusershell() — function, 3C-557
setutent — access utmp file entry, 3C-558
setutxent — access utmpx file entry, 3C-560
severity levels, applications
build a list for use with fmtmsg — addseverity, 3C-138
sfconvert — convert number to ASCII, 3-324
sgconvert — convert number to ASCII, 3-324
shared memory object
open — shm_open, 3R-1017
remove — shm_unlink, 3R-1019
shared object
close — dlclose, 3X-313
get address of symbol — dladdr, 3X-317
get diagnostic information — dlerror, 3X-314
open — dlopen, 3X-315
shell command
issue one — system, 3S-1090
shell global pattern matching — gmatch, 3G-573
shm_open — open a shared memory object, 3R-1017
shm_unlink — remove a shared memory object, 3R-1019
shutdown — shut down part of a full-duplex connection, 3N-1020
sig2str — translation between signal name and signal number, 3C-1055
sigaddset — manipulate sets of signals, 3C-1031
sigdelset — manipulate sets of signals, 3C-1031
sigemptyset — manipulate sets of signals, 3C-1031
sigfillset — manipulate sets of signals, 3C-1031
sigfpe() function, 3-1022
sighold — adds sig to the calling process's signal mask, 3C-1028
sigignore — sets the disposition of sig to SIG_IGN, 3C-1028
siginterrupt — allow signals to interrupt functions, 3B-1025
sigismember — manipulate sets of signals, 3C-1031
siglongjmp — non-local goto, 3C-1011
signal — modify signal disposition, 3C-1028, 3B-1026
queue one to a process — sigqueue, 3R-1030
schedule after interval in microseconds — ualarm, 3C-1172
suspend execution for interval in microseconds — usleep, 3C-1176
wait for queued signals — sigwaitinfo, sigtimedwait, 3R-1038
signal management
simplified, for application processes — signal, 3C-1028
signal messages
get error message string — strsignal, 3C-1076
signal messages, system
— psignal, 3B-833, 3C-834
signals, block
— sigblock, 3B-1021
— sigmask, 3B-1021
— sigpause, 3B-1021
— sigsetmask, 3B-1021
signals, software

Index—41
signals, software, continued
— gsignal, 3C-1049
— ssignal, 3C-1049
significand() function, 3M-586
sigpause — removes sig from the calling
process’s signal mask and suspends the calling
process until a signal is received, 3C-1028
sigqueue — queue a signal to a process, 3R-1030
sigrelse — removes sig from the calling
process’s signal mask, 3C-1028
sigset — modify signal disposition, 3C-1028
sigsetjmp — non-local goto, 3C-1011
sigsetops — manipulate sets of signals, 3C-1031
sigstack — set and/or get signal stack context,
3B-1032
sigtimedwait — wait for queued signals, 3R-1038
sigvec — software signal facilities, 3B-1034
sigwaitinfo — wait for queued signals, 3R-1038
sin — trigonometric sine, 3M-1164
single_to_decimal — decimal record from
single-precision floating, 3-383
sinh — hyperbolic sine, 3M-578
sleep — suspend execution for interval, 3B-1040,
3C-1041
high resolution — nanosleep, 3R-745
suspend execution for interval in microseconds
— usleep, 3C-1176
socket — create an endpoint for communication,
3N-1042
accept a connection — accept, 3N-128
bind a name — bind, 3N-167
get name — getsockname, 3N-539
get name of connected peer — getpeername,
3N-517
get options — getsockopt, 3N-540
initiate a connection — connect, 3N-197
listen for connections — listen, 3N-648
send message from — send, sendto,
sendmsg, 3N-1003
set options — setsockopt, 3N-540
shut down part of a full-duplex connection —
shutdown, 3N-1020
socketpair — create a pair of connected sockets,
3N-1045
software signals
— gsignal, 3C-1049
— ssignal, 3C-1049
sort
quick — qsort, 3C-885
space — graphics interface, 3-817
sprintf — formatted output conversion, 3B-821
spray — scatter data in order to test the network,
3N-1046
sqrt — print formatted output, 3S-825
square root — sqrt, 3M-1048
stdio — standard buffered input/output package,
3S-1050
stdin
convert EUC character from the stream to Process Code —
getwchar, 3I-565
push a Process Code character back into input
stream — ungetwc, 3I-1174
stdout — curses character and window attribute control routines, 3X-220
step — regular expression compile and match routines, 3G-909
string collation
— strcoll, 3C-1057
store — data base subroutines, 3B-298
strfind — string manipulations, 3G-1059
str2sig — translation between signal name and
signal number, 3C-1055
strcadd — copy strings, compressing or expanding
C language escape codes, 3G-1056
strccpy — copy strings, compressing or expand-
stream, put character or word, continued
— fputs, 3S-877
— putc, 3S-877
— putc_unlocked, 3S-877
— putchar, 3S-877
— putchar_unlocked, 3S-877
— putw, 3S-877

STREAMS
accept a connection on a socket — accept, 3N-128
attach a STREAMS-based file descriptor to an object in the file system name space — fattach, 3C-376
bind a name to a socket — bind, 3N-167
buffered binary input/output — fread, 3S-459
close a stream — fclose, 3S-378
create a pair of connected sockets — socket-pair, 3N-1045
create an endpoint for communication — socket, 3N-1042
detach a name from a STREAMS-based file descriptor — fdetach, 3C-380
determine whether a buffer of characters is encrypted — isencrypt, 3G-593
flush a stream — fflush, 3S-378
get and set socket options — getsockopt, setsockopt, 3N-540
get name of peer connected to socket — getpeername, 3N-517
get socket name — gethostname, 3N-539
initiate a connection on a socket — connect, 3N-197
listen for connections on a socket — listen, 3N-648
open a stream with a file descriptor — fdo-
open, 3S-426
open and associate stream with file — fopen, 3S-426
open stream with name of file — freopen, 3S-426
read stream up to next delimiter — bgets, 3G-166
reposition a file pointer — fseek, 3S-462
return to remote command — rcmd, 3N-892

Index–43
STREAMS, continued
send a message from a socket — send, sendto, sendmsg, 3N-1003
shut down part of a full-duplex connection — shutdown, 3N-1020
split buffer into fields — bufsplit, 3G-174
test file descriptor for a STREAMS file — isastream, 3C-592
strcpy — copy strings, compressing or expanding C language escape codes, 3G-1056
strfind — string manipulations, 3G-1059
strfmon — convert monetary value to string, 3C-1060
strftime — convert date and time to string, 3C-1064
string manipulations — strfind, 3G-1059
— strfind, 3G-1059
— strspn, 3G-1059
— strtrns, 3G-1059
string operation
get error message string — strerror, 3C-1058
get TLI error message string — t_strerror, 3N-1132
string operations
— strftime, 3C-1067
— strtok, 3C-1067
string transformation
— strxfrm, 3C-1080
string, convert to double-precision floating-point number
— atof, 3C-1077
— strtod, 3C-1077
string, convert to integer
— atoi, 3C-1078
— atol, 3C-1078
— atol, 3C-1078
— strtol, 3C-1078
— strtoul, 3C-1078
— strtoul, 3C-1078
string_to_decimal — decimal record from character string, 3-1070
strings
— copy, compressing or expanding C language escape codes, 3G-1056
strings, convert from numbers — econvert, 3-324
strptime — date and time conversion, 3C-1073
strfind — string manipulations, 3G-1059
strfind — string manipulations, 3G-1059
subpad — create and display curses pads, 3X-265
swab — swap bytes, 3C-1081
swapcontext — manipulate user contexts, 3C-661
symbol table
— get entries — nlist, 3B-791
synchronize a file’s data
— fdatasync, 3R-379
synchronous I/O multiplexing
— select, 3C-987
sys_siglist — system signal messages list, 3B-833
syscall — indirect system call, 3B-1082
sysconf — get configurable system variables, 3C-1083
syslog — write message to system log, 3-1086
sysmem — return physical memory information, 3-1089

Index−44
system — issue shell command, 3S-1090
system error messages
  print — perror, 3C-812
system log
  log message with variable argument list —
    vsyslog, 3-1189
system log, control — syslog, 3-1086
system signal messages
  — psignal, 3C-834
system variables
  get configurable ones — sysconf, 3C-1083

T

_t_accept — accept a connect request, 3N-1091
_t_alloc — allocate memory for argument structures, 3N-1095
_t_bind — bind an address to a transport endpoint, 3N-1095
_t_close — close a transport endpoint, 3N-1098
_t_connect — establish a connection with another transport user, 3N-1099
_t_error — produce error message, 3N-1102
_t_free — free allocated memory, 3N-1103
_t_getinfo — get protocol-specific service information, 3N-1104
_t_getstate — get the current state, 3N-1107
_t_listen — listen for a connect request, 3N-1108
_t_look — look at the current event on a transport endpoint, 3N-1110
_t_open — establish a transport endpoint, 3N-1111
_t_optmgmt — manage options for a transport endpoint, 3N-1112
_t_rcv — manage options for a transport endpoint, 3N-1114
_t_rcvconnect — receive the confirmation from a connect request, 3N-1116
_t_rcvdis — retrieve information from disconnect, 3N-1118
_t_rcvrel — acknowledge receipt of an orderly release indication, 3N-1120
_t_rcvudata — receive a data unit, 3N-1121
_t_rcvuderr — receive a unit data error indication, 3N-1123
t_snd — send data or expedited data over a connection, 3N-1125
t_snddis — send user-initiated disconnect request, 3N-1127
t_sndrel — initiate an orderly release of connection, 3N-1129
t_sndudata — send a data unit, 3N-1130
t_strerror — get error message string, 3N-1132
t_sync — synchronize transport library, 3N-1133
t_unbind — disable a transport endpoint, 3N-1135
	addr2uaddr — generic transport name-to-address translation, 3N-749
tan — trigonometric tangent, 3M-1164
tanh — hyperbolic tangent, 3M-578
tcdrain — general terminal interface, 3-1137, 3-1138
tcflow — general terminal interface, 3-1137, 3-1138
tcflush — general terminal interface, 3-1137, 3-1138
tcgetattr — general terminal interface, 3-1137, 3-1138
tcgetgpr — general terminal interface, 3-1137, 3-1140
tcgetpgrp — general terminal interface, 3-1137, 3-1140
tcgetsid — general terminal interface, 3-1137, 3-1140
tcsendbreak — general terminal interface, 3-1137, 3-1138
tcsetattr — general terminal interface, 3-1137, 3-1138
tcsetpgrp — general terminal interface, 3-1137, 3-1140
td_init — interface to libthread threads information, 3T-637
td_log — interface to libthread threads information, 3T-637
td_ta_delete — interface to libthread threads information, 3T-637
td_ta_get_nthreads — interface to libthread threads information, 3T-637
td_ta_get_ph — interface to libthread threads information, 3T-637
td_ta_map_id2thr — interface to libthread threads information, 3T-637

td_ta_map_lwp2thr — interface to libthread threads information, 3T-637

td_ta_new — interface to libthread threads information, 3T-637

td_ta_thr_iter — interface to libthread threads information, 3T-637

td_ta_tsd_iter — interface to libthread threads information, 3T-637

td_thr_get_info — interface to libthread threads information, 3T-637

td_thr_getfregs — interface to libthread threads information, 3T-637

td_thr_getgregs — interface to libthread threads information, 3T-637

td_thr_getxregs — interface to libthread threads information, 3T-637

td_thr_getxregsize — interface to libthread threads information, 3T-637

td_thr_setfregs — interface to libthread threads information, 3T-637

td_thr_setgregs — interface to libthread threads information, 3T-637

td_thr_setprio — interface to libthread threads information, 3T-637

td_thr_setsigpending — interface to libthread threads information, 3T-637

td_thr_setxregs — interface to libthread threads information, 3T-637

td_thr_sigsetmask — interface to libthread threads information, 3T-637

td_thr_tsd — interface to libthread threads information, 3T-637

td_thr_validate — interface to libthread threads information, 3T-637

tempnam — create a name for a temporary file, 3S-1161

terminal

find name — ttyslot, 3C-1170

find the slot in the utmp file of the current user — ttyslot, 3C-1171

terminal attributes

get — tcgetattr, 3-1138

set — tcsetattr, 3-1138

terminal device, slave pseudo

generate path name for controlling terminal —

ctermid, ctermid_r, 3S-203

terminal interface

baud rate — cfgetispeed, 3-1139

get terminal foreground process group —
tcgetpgrp, 3-1140

generate path name for controlling terminal —
tcgetattr, 3-1138

generate path name for controlling terminal —
tcgetpgrp, 3-1140

line control — tcdrain, 3-1138

set terminal foreground process group —
tcsetpgrp, 3-1140

terminal ID

establish an outgoing connection — dial,

3N-302

terminals

set terminal foreground process group id —
tcsetpgrp, 3C-1136

termios — general terminal interface, 3-1137

Get and Set Baud Rate, 3-1139

Get and Set Terminal Attributes, 3-1138

Get and Set Terminal Foreground Process Group ID, 3-1140

Get Terminal Session ID, 3-1140

Line Control, 3-1138

test character for specified class — iswctype,

3I-597

text processing utilities

compile and execute regular expressions —

regcmp, regex, 3G-901

quick sort — qsort, 3C-885

regular expression handler — regex, 3C-908

text string

— gettxt, 3C-555

textdomain — select domain of messages, 3I-550

thr_continue — continue thread execution,

3T-1147

thr_getconcurrency — get thread concurrency level, 3T-1145

Index—46
thr_getspecific — thread-specific-data functions, 3T-858
thr_keycreate — thread-specific-data functions, 3T-858
thr_main — identifies the calling thread as the
    main thread or not the main thread, 3T-1142
thr_setconcurrency — set thread concurrency
    level, 3T-1145
thr_setspecific — thread-specific-data functions, 3T-858
thr_stksegment — get thread stack bottom and size, 3T-1146
thr_suspend — suspend thread execution, 3T-1147

thread-specific-data functions
- pthread_getspecific, 3T-858
- pthread_key_create, 3T-858
- pthread_key_delete, 3T-858
- pthread_setspecific, 3T-858
- thr_getspecific, 3T-858
- thr_keycreate, 3T-858
- thr_setspecific, 3T-858

time
    computes the difference between two
    calendar times — difftime, 3C-304

    time accounting
    for current process — times, 3B-1159

time and date
    convert to string — asctime, 3C-204
    convert user format date and time —
    getdate, 3C-481
    get — ftime, 3C-466
    — strftime, 3C-554

time of day
    get and set — strftime, settimeofday, 3B-553

    time, calendar
    convert from a tm structure —
    mktime, 3C-713

timer_create — create a per-process, per-LWP, or per-thread timer,

    3R-1154
    timer_delete — delete a per-process, per-LWP, or per-thread timer,
    3R-1156
    timer_getoverrun — high-resolution
    timer operations, 3R-1157
    timer_gettime — high-resolution timer
    operations, 3R-1157
    timer_settime — high-resolution timer
    operations, 3R-1157
    times — get process times, 3B-1159

tmpfile — create a temporary file, 3S-1160
tmpnam — create a name for a temporary
    file, 3S-1161

    TNF_PROBE — probe insertion interface, 3X-121
    tnf_process_disable() — disables prob-
    ing for the process, 3X-1163
    tnf_process_enable() — enables probing
    for the process, 3X-1163
    tnf_thread_disable() — disables probing
    for the calling thread, 3X-1163
    tnf_thread_enable() — enables probing
    for the calling thread, 3X-1163

    towlower — Process Code character
    conversion macros, 3I-1197
    towupper — Process Code character
    conversion macros, 3I-1197

    translate address to symbolic informa-
    tion — dladdr, 3X-311

    translation between signal name and
    signal number — str2sig, 3C-1055
    sig2str, 3C-1055

transport functions
    accept a connect request —
    _t_accept, 3N-1091
    acknowledge receipt of an orderly
    release indication —
    _t_rcvrel, 3N-1120

    allocate memory for argument struc-
    tures — _t_alloc, 3N-1093
transport functions, continued
bind an address to a transport endpoint – t_bind, 3N-1095
close a transport endpoint – t_close, 3N-1098
disable a transport endpoint – t_unbind, 3N-1135
establish a connection with another transport user – t_connect, 3N-1099
establish a transport endpoint – t_open, 3N-1111
free allocated memory – t_free, 3N-1103
get protocol-specific service information – t_getinfo, 3N-1104
get the current state – t_getstate, 3N-1107
initiate an orderly release of connection – t_sndrel, 3N-1129
listen for a connect request – t_listen, 3N-1108
look at the current event on a transport endpoint – t_look, 3N-1110
manage options for a transport endpoint – t_optmgmt, 3N-1112
produce error message – t_error, 3N-1102
receive a data unit – t_rcvudata, 3N-1121
receive a unit data error indication – t_rcvuderr, 3N-1123
receive data or expedited data sent over a connection – t_rcv, 3N-1114
receive the confirmation from a connect request – t_rcvconnect, 3N-1116
retrieve information from disconnect – t_rcvdis, 3N-1118
send a data unit – t_sndudata, 3N-1130
send data or expedited data over a connection – t_snd, 3N-1125
send user-initiated disconnect request – t_snddis, 3N-1127
synchronize transport library – t_sync, 3N-1133
trigonometric functions
– acos, 3M-1164
– asin, 3M-1164
– atan, 3M-1164
– atan2, 3M-1164
– cos, 3M-1164
– sin, 3M-1164
– tan, 3M-1164
truncate – set a file to a specified length, 3C-1165
ttynam – find name of a terminal, 3C-1170
ttynam_r – find name of a terminal, 3C-1170
ttyslot – find the slot in the utmp file of the current user, 3C-1171
U
uaddr2taddr – generic transport name-to-address translation, 3N-749
ualarm – schedule signal after interval in microseconds, 3C-1172
ungetwc – push a Process Code character back into input stream, 3I-1174
ungetwch – get (or push back) wchar_t characters from curses terminal keyboard, 3X-237
unlock a pseudo-terminal master/slave pair – unlockpt, 3C-1175
unlock address space – munlockall, 3C-717
unlock memory pages – munlock, 3C-715
unlockpt – unlock a pseudo-terminal master/slave pair, 3C-1175
updwtmp – access utmpx file entry,
 updtmpx: access utmpx file entry, 3C-560
 user context
   − makecontext, 3C-661
   − swapcontext, 3C-661
 user IDs
   − get character-string representation
     − cuserid, 3S-297
 users
   − return information from remote
     machines − rusers, rnusers, 3N-966
 usleep: suspend execution for interval
         in microseconds, 3C-1176
 utmp: access entry − getutent, 3C-558
        find the slot of current user −
        ttyslot, 3C-1171
 utmpname: access utmp file entry,
           3C-558
 utmpx: access entry − getutxent, 3C-560
 utmpxname: access utmpx file entry,
           3C-560
 V
 vfprintf: formatted output conversion, 3B-821
 vfstab: access entry − getvfsent, 3C-563
 virtual memory
   − optimizing usage of user mapped
     memory − madvise, 3-657
 vlfmt: display error message in stand-
        ard format and pass to logging and
        monitoring services, 3C-1177
 volmgt_check: have Volume Management
               check for media, 3X-1179
 volmgt_inuse: check whether or not
               Volume Management is managing a
               pathname, 3X-1180
 volmgt_root: return the Volume Manage-
               ment root directory, 3X-1181
 volmgt_running: return whether or not
                 Volume Management is running,
                 3X-1182
 volmgt_symdev: convert between Volume
                Management symbolic names, and the
                devices that correspond to them,
                3X-1183
 volmgt_symname: convert between Volume
                 Management symbolic names, and the
                 devices that correspond to them,
                 3X-1183
 vpfmt: display error message in stan-
        dard format and pass to logging and
        monitoring services, 3C-1185
 vprintf: formatted output conversion, 3B-821
 vsprintf: formatted output conversion, 3B-821
 vsyslog(): log message with variable argument
           list, 3-1189
 VTOC, disk’s
   − read a disk’s VTOC — read_vtoc, 3X-894
   − write a disk’s VTOC — write_vtoc, 3X-894
 W
 waddnwstr: add a string of wchar_t characters to
            a curses window and advance cursor, 3X-218
 waddwch: add a wchar_t character (with attri-
          butes) to a curses window and advance cursor,
          3X-214
 waddwchnstr: add string of wchar_t characters
             (and attributes) to a curses window, 3X-216
 waddwchstr: add string of wchar_t characters
             (and attributes) to a curses window, 3X-216
 waddwstr: add string of wchar_t characters
            to a curses window and advance cursor, 3X-218
 wadjcurspos: moving the cursor by character,
              3X-219
 wait3: wait for process to terminate or stop,
        3B-1190, 3C-1194
 w atof: convert wide character string to double
precision number, 3I-1200

watoi — convert wide character string to long
integer, 3I-1202

watol — convert wide character string to long
integer, 3I-1202

watoll — convert wide character string to long
integer, 3I-1202

wattroff — curses character and window attribute
control routines, 3X-220

watron — curses character and window attribute
control routines, 3X-220

wartrset — curses character and window attribute
control routines, 3X-220

wchar_t string
number conversion — wscanf, 3I-1216

wconv — Process Code character conversion macros, 3I-1197

wcscat — wide character string operations, 3I-1207

wcschr — wide character string operations, 3I-1208

wcsncmp — wide character string operations, 3I-1208

wcscoll — wide character string comparison using
collating information, 3I-1198

wcsncpy — wide character string operations, 3I-1207

wcscat — wide character string comparison using
collating information, 3I-1198

wcsncmp, 3I-1198

wcscoll, 3I-1198

wctype — define character class, 3I-1211

wcwidth — wide character string operations, 3I-1208

wchowchar — add a wchar_t character (with
attributes) to a curses window and advance
cursor, 3X-214

wgetnwstr — get wchar_t character strings from
curses terminal keyboard, 3X-241

wgetwch — get (or push back) wchar_t characters
from curses terminal keyboard, 3X-237

wgetwstr — get wchar_t character strings from
curses terminal keyboard, 3X-241

wide character string comparison using collating
information

— wcscoll, 3I-1198

— wcsncmp, 3I-1198

wide character string operations

— wcscat, 3I-1207

— wcschr, 3I-1208

— wcsncmp, 3I-1207

— wcsncpy, 3I-1207

— wcscat, 3I-1207

— wcsncmp, 3I-1207

— wcsncpy, 3I-1207

— wcscat, 3I-1207

— wcsncmp, 3I-1207

— wcsncpy, 3I-1207

— wcscat, 3I-1207

— wcsncmp, 3I-1207

— wcsncpy, 3I-1207

— wcscat, 3I-1207

— wcsncmp, 3I-1207

— wcsncpy, 3I-1207
wide character string operations, continued
   — wcstok, 3I-1209
   — wcstring, 3I-1207
   — wcswcs, 3I-1209
   — wcswidth, 3I-1208
   — wcwidth, 3I-1208
   — windex, 3I-1208
   — wrindex, 3I-1208
wide character string to long integer, convert
   — watoi, 3I-1202
   — watol, 3I-1202
   — watoll, 3I-1202
   — wcstol, 3I-1202
   — wstol, 3I-1202
wide character string transformation
   —wcsxfrm, 3I-1210
   — wsxfrm, 3I-1210
windex — wide character string operations, 3I-1208
winnwstr — get a string of wchar_t characters from a curses window, 3X-254
winsnwstr — insert wchar_t string before character under the cursor in a curses window, 3X-254
winswch — insert a wchar_t character before the character under the cursor in a curses window, 3X-253
winswstr — insert wchar_t string before character under the cursor in a curses window, 3X-254
winwch — get a wchar_t character and its attributes from a curses window, 3X-256
winwchnstr — get a string of wchar_t characters (and attributes) from a curses window, 3X-257
winwchstr — get a string of wchar_t characters (and attributes) from a curses window, 3X-257
winwstr — get a string of wchar_t characters from a curses window, 3X-258
wmovenextch — moving the cursor by character, 3X-219
wmoveprevch — moving the cursor by character, 3X-219
wordexp — perform word expansions, 3C-1212
wordfree — perform word expansions, 3C-1212
working directory, continued
   get pathname — getwd, 3C-566
wrindex — wide character string operations, 3I-1208
write_vtoc — read and write a disk’s VTOC, 3X-894
wscasecmp — Process Code string operations, 3I-1217
wscoll — Process Code string operations, 3I-1217
wscoll — wide character string comparison using collating information, 3I-1198
wsdup — Process Code string operations, 3I-1217
wsncasecmp — Process Code string operations, 3I-1217
wsprintf — formatted output conversion, 3I-1215
wsscanf — formatted input conversion, 3I-1216
xdr — XDR library routines
   — xdr, 3N-1218
   — xdr_admin, 3N-1220
   — xdr_control, 3N-1220
   — xdr_getpos, 3N-1220
   — xdr-inline, 3N-1220
   — xdr_setpos, 3N-1220
   — xdr_sizeof, 3N-1220
   — xdrrec_endofrecord, 3N-1220
   — xdrrec_eof, 3N-1220
   — xdrrec_readbytes, 3N-1220
   — xdrrec_skiprecord, 3N-1220
XDR library routines for complex data structures
   — xdr-array, 3N-1223
   — xdr_bytes, 3N-1223
XDR library routines for complex data structures, continued
— xdr_complex, 3N-1223
— xdrOpaque, 3N-1223
— xdr_pointer, 3N-1223
— xdr_reference, 3N-1223
— xdr_string, 3N-1223
— xdr_union, 3N-1223
— xdr_vector, 3N-1223
— xdr_wrapstring, 3N-1223
XDR library routines for RPC
— rpc_xdr, 3N-961
— xdr_accepted_reply, 3N-961
— xdr_authsys_parms, 3N-961
— xdr_callhdr, 3N-961
— xdr_callmsg, 3N-961
— xdr_opaque_auth, 3N-961
— xdr_rejected_reply, 3N-961
— xdr_replymsg, 3N-961
XDR library routines for simple data structures
— xdr_bool, 3N-1228
— xdr_char, 3N-1228
— xdr_double, 3N-1228
— xdr_enum, 3N-1228
— xdr_float, 3N-1228
— xdr_free, 3N-1228
— xdr_hyper, 3N-1228
— xdr_int, 3N-1228
— xdr_long, 3N-1228
— xdr_longlong_t, 3N-1228
— xdr_quadruple, 3N-1228
— xdr_short, 3N-1228
— xdr_simple, 3N-1228
— xdr_u_char, 3N-1228
— xdr_u_hyper, 3N-1228
— xdr_u_int, 3N-1228
— xdr_u_long, 3N-1228
— xdr_u_longlong_t, 3N-1228
— xdr_u_short, 3N-1228
— xdr_void, 3N-1228
XDR stream creation library routines, continued
— xdrstdio_create, 3N-1226
xdr_statstime — get performance data from remote kernel, 3N-965
xdr_statsvar — get performance data from remote kernel, 3N-965

Y
y0 — Bessel function, 3M-165
y1 — Bessel function, 3M-165
yn — Bessel function, 3M-165