NAME

connect − initiate a connection on a socket

SYNOPSIS

#include <sys/types.h>
#include <sys/socket.h>

int connect(int *sockfd*, **const struct sockaddr ***serv_addr*, **socklen_t** *addrlen*);**

DESCRIPTION

The file descriptor *sockfd* must refer to a socket. If the socket is of type **SOCK_DGRAM** then the *serv_addr* address is the address to which datagrams are sent by default, and the only address from which datagrams are received. If the socket is of type **SOCK_STREAM** or **SOCK_SEQPACKET**, this call attempts to make a connection to another socket. The other socket is specified by *serv_addr*, which is an address (of length *addrlen*) in the communications space of the socket. Each communications space interprets the *serv_addr* parameter in its own way.

Generally, connection-based protocol sockets may successfully **connect** only once; connectionless protocol sockets may use **connect** multiple times to change their association. Connectionless sockets may dissolve the association by connecting to an address with the *sa_family* member of **sockaddr** set to **AF_UNSPEC**.

RETURN VALUE

If the connection or binding succeeds, zero is returned. On error, −1 is returned, and *errno* is set appropriately.

ERRORS

The following are general socket errors only. There may be other domain-specific error codes.

EBADF
The file descriptor is not a valid index in the descriptor table.

EFAULT
The socket structure address is outside the user's address space.

ENOTSOCK
The file descriptor is not associated with a socket.

EISCONN
The socket is already connected.

ECONNREFUSED
No one listening on the remote address.

ENETUNREACH
Network is unreachable.

EADDRINUSE
Local address is already in use.

EAFNOSUPPORT
The passed address didn't have the correct address family in its *sa_family* field.

EACCES, EPERM
The user tried to connect to a broadcast address without having the socket broadcast flag enabled or the connection request failed because of a local firewall rule.

SEE ALSO

accept(2), **bind**(2), **listen**(2), **socket**(2), **getsockname**(2)

NAME

opendir − open a directory / readdir − read a directory

SYNOPSIS

#include <sys/types.h>

#include <dirent.h>

DIR *opendir(const char ***name*);

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

DESCRIPTION opendir

The **opendir()** function opens a directory stream corresponding to the directory *name*, and returns a pointer to the directory stream. The stream is positioned at the first entry in the directory.

RETURN VALUE

The **opendir()** function returns a pointer to the directory stream or NULL if an error occurred.

DESCRIPTION readdir

The **readdir()** function returns a pointer to a dirent structure representing the next directory entry in the directory stream pointed to by *dir*. It returns NULL on reaching the end-of-file or if an error occurred.

DESCRIPTION readdir_r

The **readdir_r()** function initializes the structure referenced by *entry* and stores a pointer to this structure in *result*. On successful return, the pointer returned at **result* will have the same value as the argument *entry*. Upon reaching the end of the directory stream, this pointer will have the value NULL.

The data returned by **readdir()** is overwritten by subsequent calls to **readdir()** for the **same** directory stream.

The *dirent* structure is defined as follows:

```
struct dirent {
    long          d_ino;              /* inode number */
    off_t         d_off;              /* offset to the next dirent */
    unsigned short  d_reclen;         /* length of this record */
    unsigned char   d_type;       /* type of file */
    char          d_name[256];    /* filename */
};
```

RETURN VALUE

The **readdir()** function returns a pointer to a dirent structure, or NULL if an error occurs or end-of-file is reached.

readdir_r() returns 0 if successful or an error number to indicate failure.

ERRORS

EACCES
Permission denied.

ENOENT
Directory does not exist, or *name* is an empty string.

ENOTDIR
name is not a directory.

NAME

clearerr, feof, ferror, fileno − check and reset stream status

SYNOPSIS

#include <stdio.h>

void clearerr(FILE ****stream*);**
int feof(FILE ****stream*);**
int ferror(FILE ****stream*);**
int fileno(FILE ****stream*);**

DESCRIPTION

The function **clearerr**() clears the end-of-file and error indicators for the stream pointed to by *stream*.

The function **feof**() tests the end-of-file indicator for the stream pointed to by *stream*, returning non-zero if it is set. The end-of-file indicator can only be cleared by the function **clearerr**().

The function **ferror**() tests the error indicator for the stream pointed to by *stream*, returning non-zero if it is set. The error indicator can only be reset by the **clearerr**() function.

The function **fileno**() examines the argument *stream* and returns its integer descriptor.

For non-locking counterparts, see **unlocked_stdio**(3).

ERRORS

These functions should not fail and do not set the external variable *errno*. (However, in case **fileno**() detects that its argument is not a valid stream, it must return −1 and set *errno* to **EBADF**.)

CONFORMING TO

The functions **clearerr**(), **feof**(), and **ferror**() conform to C89 and C99.

SEE ALSO

open(2), **fdopen**(3), **stdio**(3), **unlocked_stdio**(3)

NAME

fopen, fdopen − stream open functions

SYNOPSIS

#include <stdio.h>

FILE *fopen(const char * *path***, const char *** *mode***);**
FILE *fdopen(int *fildes***, const char *** *mode***);**

DESCRIPTION

The **fopen** function opens the file whose name is the string pointed to by *path* and associates a stream with it.

The argument *mode* points to a string beginning with one of the following sequences (Additional characters may follow these sequences.):

r Open text file for reading. The stream is positioned at the beginning of the file.

r+ Open for reading and writing. The stream is positioned at the beginning of the file.

w Truncate file to zero length or create text file for writing. The stream is positioned at the beginning of the file.

w+ Open for reading and writing. The file is created if it does not exist, otherwise it is truncated. The stream is positioned at the beginning of the file.

a Open for appending (writing at end of file). The file is created if it does not exist. The stream is positioned at the end of the file.

a+ Open for reading and appending (writing at end of file). The file is created if it does not exist. The stream is positioned at the end of the file.

The **fdopen** function associates a stream with the existing file descriptor, *fildes*. The *mode* of the stream (one of the values "r", "r+", "w", "w+", "a", "a+") must be compatible with the mode of the file descriptor. The file position indicator of the new stream is set to that belonging to *fildes*, and the error and end-of-file indicators are cleared. Modes "w" or "w+" do not cause truncation of the file. The file descriptor is not dup'ed, and will be closed when the stream created by **fdopen** is closed. The result of applying **fdopen** to a shared memory object is undefined.

RETURN VALUE

Upon successful completion **fopen**, **fdopen** and **freopen** return a **FILE** pointer. Otherwise, **NULL** is returned and the global variable *errno* is set to indicate the error.

ERRORS

EINVAL

The *mode* provided to **fopen**, **fdopen**, or **freopen** was invalid.

The **fopen**, **fdopen** and **freopen** functions may also fail and set *errno* for any of the errors specified for the routine **malloc**(3).

The **fopen** function may also fail and set *errno* for any of the errors specified for the routine **open**(2).

The **fdopen** function may also fail and set *errno* for any of the errors specified for the routine **fcntl**(2).

SEE ALSO

open(2), **fclose**(3), **fileno**(3)

NAME

getaddrinfo, freeaddrinfo, gai_strerror − network address and service translation

SYNOPSIS

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>

int getaddrinfo(const char *node, const char *service,
        const struct addrinfo *hints,
        struct addrinfo **res);

void freeaddrinfo(struct addrinfo *res);

const char *gai_strerror(int errcode);
```

DESCRIPTION

Given *node* and *service*, which identify an Internet host and a service, **getaddrinfo**() returns one or more *addrinfo* structures, each of which contains an Internet address that can be specified in a call to **bind**(2) or **connect**(2).

The *addrinfo* structure used by **getaddrinfo**() contains the following fields:

```
struct addrinfo {
    int             ai_flags;
    int             ai_family;
    int             ai_socktype;
    int             ai_protocol;
    size_t          ai_addrlen;
    struct sockaddr *ai_addr;
    char            *ai_canonname;
    struct addrinfo *ai_next;
};
```

The *hints* argument points to an *addrinfo* structure that specifies criteria for selecting the socket address structures returned in the list pointed to by *res*. If *hints* is not NULL it points to an *addrinfo* structure whose *ai_family*, *ai_socktype*, and *ai_protocol* specify criteria that limit the set of socket addresses returned by **getaddrinfo**(), as follows:

ai_family This field specifies the desired address family for the returned addresses. Valid values for this field include **AF_INET** and **AF_INET6**. The value **AF_UNSPEC** indicates that **getaddrinfo**() should return socket addresses for any address family (either IPv4 or IPv6, for example) that can be used with *node* and *service*.

ai_socktype This field specifies the preferred socket type, for example **SOCK_STREAM** or **SOCK_DGRAM**. Specifying 0 in this field indicates that socket addresses of any type can be returned by **getaddrinfo**().

ai_protocol This field specifies the protocol for the returned socket addresses. Specifying 0 in this field indicates that socket addresses with any protocol can be returned by **getaddrinfo**().

ai_flags This field specifies additional options, described below. Multiple flags are specified by logically OR-ing them together.

All the other fields in the structure pointed to by *hints* must contain either 0 or a null pointer, as appropriate. Specifying *hints* as NULL is equivalent to setting *ai_socktype* and *ai_protocol* to 0; *ai_family* to **AF_UNSPEC**; and *ai_flags* to **(AI_V4MAPPED | AI_ADDRCONFIG)**.

node specifies either a numerical network address (for IPv4, numbers-and-dots notation as supported by **inet_aton**(3); for IPv6, hexadecimal string format as supported by **inet_pton**(3)), or a network hostname,

whose network addresses are looked up and resolved. If *hints.ai_flags* contains the **AI_NUMERICHOST** flag then *node* must be a numerical network address. The **AI_NUMERICHOST** flag suppresses any potentially lengthy network host address lookups.

If the **AI_PASSIVE** flag is specified in *hints.ai_flags*, and *node* is NULL, then the returned socket addresses will be suitable for **bind**(2)ing a socket that will **accept**(2) connections. The returned socket address will contain the "wildcard address" (**INADDR_ANY** for IPv4 addresses, **IN6ADDR_ANY_INIT** for IPv6 address). The wildcard address is used by applications (typically servers) that intend to accept connections on any of the hosts's network addresses.

The **getaddrinfo**() function allocates and initializes a linked list of *addrinfo* structures, one for each network address that matches *node* and *service*, subject to any restrictions imposed by *hints*, and returns a pointer to the start of the list in *res*. The items in the linked list are linked by the *ai_next* field. There are several reasons why the linked list may have more than one *addrinfo* structure, including: the network host is multi-homed; or the same service is available from multiple socket protocols (one **SOCK_STREAM** address and another **SOCK_DGRAM** address, for example).

If *hints.ai_flags* includes the **AI_CANONNAME** flag, then the *ai_canonname* field of the first of the *addrinfo* structures in the returned list is set to point to the official name of the host.

The remaining fields of each returned *addrinfo* structure are initialized as follows:

* The *ai_family*, *ai_socktype*, and *ai_protocol* fields return the socket creation parameters (i.e., these fields have the same meaning as the corresponding arguments of **socket**(2)). For example, *ai_family* might return **AF_INET** or **AF_INET6**; *ai_socktype* might return **SOCK_DGRAM** or **SOCK_STREAM**; and *ai_protocol* returns the protocol for the socket.
* A pointer to the socket address is placed in the *ai_addr* field, and the length of the socket address, in bytes, is placed in the *ai_addrlen* field.

If *hints.ai_flags* includes the **AI_ADDRCONFIG** flag, then IPv4 addresses are returned in the list pointed to by *result* only if the local system has at least one IPv4 address configured, and IPv6 addresses are only returned if the local system has at least one IPv6 address configured.

If *hint.ai_flags* specifies the **AI_V4MAPPED** flag, and *hints.ai_family* was specified as **AF_INET6**, and no matching IPv6 addresses could be found, then return IPv4-mapped IPv6 addresses in the list pointed to by *result*. If both **AI_V4MAPPED** and **AI_ALL** are specified in *hints.ai_family*, then return both IPv6 and IPv4-mapped IPv6 addresses in the list pointed to by *result*. **AI_ALL** is ignored if **AI_V4MAPPED** is not also specified.

The **freeaddrinfo**() function frees the memory that was allocated for the dynamically allocated linked list *res*.

Extensions to getaddrinfo() for Internationalized Domain Names

SEE ALSO

gethostbyname(3), **getnameinfo**(3), **inet**(3), **hostname**(7), **ip**(7)

NAME

gets, fgets − get a string from a stream
fputs, puts − output of strings

SYNOPSIS

#include <stdio.h>

char *gets(char * *s***);**

char *fgets(char * *s***, int** *n***, FILE *** *stream***);**

int fputs(const char * *s***, FILE *** *stream***);**

int puts(const char * *s***);**

DESCRIPTION gets/fgets

The **gets( )** function reads characters from the standard input stream (see **intro**(3)), **stdin**, into the array pointed to by *s*, until a newline character is read or an end-of-file condition is encountered. The newline character is discarded and the string is terminated with a null character.

The **fgets( )** function reads characters from the *stream* into the array pointed to by *s*, until *n*−1 characters are read, or a newline character is read and transferred to *s*, or an end-of-file condition is encountered. The string is then terminated with a null character.

When using **gets( )**, if the length of an input line exceeds the size of *s*, indeterminate behavior may result. For this reason, it is strongly recommended that **gets( )** be avoided in favor of **fgets( )**.

RETURN VALUES

If end-of-file is encountered and no characters have been read, no characters are transferred to *s* and a null pointer is returned. If a read error occurs, such as trying to use these functions on a file that has not been opened for reading, a null pointer is returned and the error indicator for the stream is set. If end-of-file is encountered, the **EOF** indicator for the stream is set. Otherwise *s* is returned.

ERRORS

The **gets( )** and **fgets( )** functions will fail if data needs to be read and:

EOVERFLOW The file is a regular file and an attempt was made to read at or beyond the offset maximum associated with the corresponding *stream*.

DESCRIPTION puts/fputs

fputs() writes the string *s* to *stream*, without its trailing **'\0'**.

puts() writes the string *s* and a trailing newline to *stdout*.

Calls to the functions described here can be mixed with each other and with calls to other output functions from the **stdio** library for the same output stream.

RETURN VALUE

puts() and **fputs()** return a non - negative number on success, or **EOF** on error.

NAME

ipv6, PF_INET6 − Linux IPv6 protocol implementation

SYNOPSIS

#include <sys/socket.h>
#include <netinet/in.h>

tcp6_socket = **socket(PF_INET6, SOCK_STREAM, 0);**
raw6_socket = **socket(PF_INET6, SOCK_RAW,** *protocol***);**
udp6_socket = **socket(PF_INET6, SOCK_DGRAM,** *protocol***);**

DESCRIPTION

Linux 2.2 optionally implements the Internet Protocol, version 6. This man page contains a description of the IPv6 basic API as implemented by the Linux kernel and glibc 2.1. The interface is based on the BSD sockets interface; see **socket**(7).

The IPv6 API aims to be mostly compatible with the **ip**(7) v4 API. Only differences are described in this man page.

To bind an **AF_INET6** socket to any process the local address should be copied from the *in6addr_any* variable which has *in6_addr* type. In static initializations **IN6ADDR_ANY_INIT** may also be used, which expands to a constant expression. Both of them are in network order.

The IPv6 loopback address (::1) is available in the global *in6addr_loopback* variable. For initializations **IN6ADDR_LOOPBACK_INIT** should be used.

IPv4 connections can be handled with the v6 API by using the v4-mapped-on-v6 address type; thus a program only needs only to support this API type to support both protocols. This is handled transparently by the address handling functions in libc.

IPv4 and IPv6 share the local port space. When you get an IPv4 connection or packet to a IPv6 socket its source address will be mapped to v6 and it will be mapped to v6.

Address Format

```
struct sockaddr_in6 {
   uint16_t       sin6_family;   /* AF_INET6 */
   uint16_t       sin6_port;     /* port number */
   uint32_t       sin6_flowinfo; /* IPv6 flow information */
   struct in6_addr sin6_addr;    /* IPv6 address */
   uint32_t       sin6_scope_id; /* Scope ID (new in 2.4) */
};

struct in6_addr {
   unsigned char   s6_addr[16];   /* IPv6 address */
};
```

sin6_family is always set to **AF_INET6**; *sin6_port* is the protocol port (see *sin_port* in **ip**(7)); *sin6_flowinfo* is the IPv6 flow identifier; *sin6_addr* is the 128-bit IPv6 address. *sin6_scope_id* is an ID of depending of on the scope of the address. It is new in Linux 2.4. Linux only supports it for link scope addresses, in that case *sin6_scope_id* contains the interface index (see **netdevice**(7))

NOTES

The *sockaddr_in6* structure is bigger than the generic *sockaddr*. Programs that assume that all address types can be stored safely in a *struct sockaddr* need to be changed to use *struct sockaddr_storage* for that instead.

SEE ALSO

cmsg(3), **ip**(7)

NAME

mkdir − create a directory

SYNOPSIS

#include <sys/stat.h>
#include <sys/types.h>

int mkdir(const char * *pathname***, mode_t** *mode***);**

DESCRIPTION

mkdir() attempts to create a directory named *pathname*.

The argument *mode* specifies the permissions to use. It is modified by the process's *umask* in the usual way: the permissions of the created directory are (*mode* & ˜*umask* & 0777). Other mode bits of the created directory depend on the operating system. For Linux, see below.

The newly created directory will be owned by the effective user ID of the process. If the directory containing the file has the set-group-ID bit set, or if the file system is mounted with BSD group semantics (*mount -o bsdgroups* or, synonymously *mount -o grpid*), the new directory will inherit the group ownership from its parent; otherwise it will be owned by the effective group ID of the process.

If the parent directory has the set-group-ID bit set then so will the newly created directory.

RETURN VALUE

mkdir() returns zero on success, or −1 if an error occurred (in which case, *errno* is set appropriately).

ERRORS

EACCES
The parent directory does not allow write permission to the process, or one of the directories in *pathname* did not allow search permission. (See also **path_resolution**(7).)

EEXIST
pathname already exists (not necessarily as a directory). This includes the case where *pathname* is a symbolic link, dangling or not.

ENOTDIR
A component used as a directory in *pathname* is not, in fact, a directory.

EPERM
The file system containing *pathname* does not support the creation of directories.

SEE ALSO

mkdir(1), **chmod**(2), **chown**(2), **mkdirat**(2), **mknod**(2), **mount**(2), **rmdir**(2), **stat**(2), **umask**(2), **unlink**(2), **path_resolution**(7)

NAME

pthread_create − create a new thread / pthread_exit − terminate the calling thread

SYNOPSIS

#include <pthread.h>

int pthread_create(pthread_t * *thread***, pthread_attr_t *** *attr***, void * (***start_routine***)(void *), void *** *arg***);**

void pthread_exit(void * *retval***);**

DESCRIPTION

pthread_create creates a new thread of control that executes concurrently with the calling thread. The new thread applies the function *start_routine* passing it *arg* as first argument. The new thread terminates either explicitly, by calling **pthread_exit**(3), or implicitly, by returning from the *start_routine* function. The latter case is equivalent to calling **pthread_exit**(3) with the result returned by *start_routine* as exit code.

The *attr* argument specifies thread attributes to be applied to the new thread. See **pthread_attr_init**(3) for a complete list of thread attributes. The *attr* argument can also be **NULL**, in which case default attributes are used: the created thread is joinable (not detached) and has default (non real-time) scheduling policy.

pthread_exit terminates the execution of the calling thread. All cleanup handlers that have been set for the calling thread with **pthread_cleanup_push**(3) are executed in reverse order (the most recently pushed handler is executed first). Finalization functions for thread-specific data are then called for all keys that have non- **NULL** values associated with them in the calling thread (see **pthread_key_create**(3)). Finally, execution of the calling thread is stopped.

The *retval* argument is the return value of the thread. It can be consulted from another thread using **pthread_join**(3).

RETURN VALUE

On success, the identifier of the newly created thread is stored in the location pointed by the *thread* argument, and a 0 is returned. On error, a non-zero error code is returned.

The **pthread_exit** function never returns.

ERRORS

EAGAIN
not enough system resources to create a process for the new thread.

EAGAIN
more than **PTHREAD_THREADS_MAX** threads are already active.

AUTHOR

Xavier Leroy <Xavier.Leroy@inria.fr>

SEE ALSO

pthread_join(3), **pthread_detach**(3), **pthread_attr_init**(3).

NAME

socket − create an endpoint for communication

SYNOPSIS

#include <sys/types.h>
#include <sys/socket.h>

int socket(int *domain***, int** *type***, int** *protocol***);**

DESCRIPTION

socket( ) creates an endpoint for communication and returns a descriptor.

The *domain* parameter specifies a communications domain within which communication will take place; this selects the protocol family which should be used. The protocol family generally is the same as the address family for the addresses supplied in later operations on the socket. The currently understood formats are:

PF_INET ARPA Internet protocols

The socket has the indicated *type*, which specifies the communication semantics. Currently defined types are:

SOCK_STREAM
SOCK_DGRAM

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).

protocol specifies a particular protocol to be used with the socket. Normally only a single protocol exists to support a particular socket type within a given protocol family. However, multiple protocols may exist, in which case a particular protocol must be specified in this manner. The protocol number to use is particular to the "communication domain" in which communication is to take place. If a protocol is specified by the caller, then it will be packaged into a socket level option request and sent to the underlying protocol layers.

Sockets of type **SOCK_STREAM** are full-duplex byte streams, similar to pipes. A stream socket must be in a *connected* state before any data may be sent or received on it. A connection to another socket is created with a **connect**(3N) call. Once connected, data may be transferred using **read**(2) and **write**(2) calls or some variant of the **send**(3N) and **recv**(3N) calls. When a session has been completed, a **close**(2) may be performed. Out-of-band data may also be transmitted as described on the **send**(3N) manual page and received as described on the **recv**(3N) manual page.

The communications protocols used to implement a **SOCK_STREAM** insure that data is not lost or duplicated. If a piece of data for which the peer protocol has buffer space cannot be successfully transmitted within a reasonable length of time, then the connection is considered broken and calls will indicate an error with −1 returns and with **ETIMEDOUT** as the specific code in the global variable **errno**. 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.

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.
ENOMEM	Insufficient user memory is available.

SEE ALSO

close(2), **read**(2), **write**(2), **accept**(3N), **bind**(3N), **connect**(3N), **listen**(3N),

NAME

unlink − remove directory entry

SYNOPSIS

#include <unistd.h>

int unlink(const char * *path***);**

DESCRIPTION

The **unlink( )** function removes a link to a file. It removes the link named by the pathname pointed to by *path* and decrements the link count of the file referenced by the link.

When the file's link count becomes 0 and no process has the file open, the space occupied by the file will be freed and the file will no longer be accessible. If one or more processes have the file open when the last link is removed, the link will be removed before **unlink( )** returns, but the removal of the file contents will be postponed until all references to the file are closed.

RETURN VALUES

Upon successful completion, **0** is returned. Otherwise, **−1** is returned and **errno** is set to indicate the error.

ERRORS

The **unlink( )** function will fail and not unlink the file if:

EACCES	Search permission is denied for a component of the *path* prefix.
EACCES	Write permission is denied on the directory containing the link to be removed.
ENOENT	The named file does not exist or is a null pathname.
ENOTDIR	A component of the *path* prefix is not a directory.
EPERM	The named file is a directory and the effective user of the calling process is not superuser.

SEE ALSO

rm(1), **close**(2), **link**(2), **open**(2), **rmdir**(2),