NAME
accept − accept a connection on a socket

SYNOPSIS
#include <sys/types.h>
#include <sys/socket.h>
int accept(int s, struct sockaddr *addr, int *addrlen);

DESCRIPTION
The argument s is a socket that has been created with socket(2) and bound to an address with bind(2), and that is listening for connections after a call to listen(2). The accept() function extracts the first connection on the queue of pending connections, creates a new socket with the properties of s, and allocates a new file descriptor, ns, for the socket. If no pending connections are present on the queue and the socket is not marked as non-blocking, accept() blocks the caller until a connection is present. If the socket is marked as non-blocking and no pending connections are present on the queue, accept() returns an error as described below. The accept() function uses the netconfig(4) file to determine the STREAMS device file name associated with s. This is the device on which the connect indication will be accepted. The accepted socket, ns, is used to read and write data to and from the socket that connected to ns; it is not used to accept more connections. The original socket(s) remains open for accepting further connections.

The argument addr is a result parameter that is filled in with the address of the connecting entity as it is known to the communications layer. The exact format of the addr parameter is determined by the domain in which the communication occurs.

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

The accept() function is used with connection-based socket types, currently with SOCK_STREAM.

It is possible to use select(2) 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 VALUE
On success, these system calls return a nonnegative integer that is a file descriptor for the accepted socket. On error, −1 is returned, and errno is set appropriately.

ERRORS
accept() will fail if:
EBADF   The descriptor is invalid.
EINTR   The accept attempt was interrupted by the delivery of a signal.
EMFILE  The per-process descriptor table is full.
ENOENT  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.
EPROT    A protocol error has occurred; for example, the STREAMS protocol stack has not been initialized or the connection has already been released.
EWOULDBLOCK  The socket is marked as non-blocking and no connections are present to be accepted.

SEE ALSO
poll(2), bind(2), connect(2), listen(2), select(2), socket(2), netconfig(4), attributes(5), socket(5)

NAME
bind − bind a name to a socket

SYNOPSIS
#include <sys/types.h>
#include <sys/socket.h>
int bind(int s, const struct sockaddr *name, int namelen);

DESCRIPTION
bind() assigns a name to an unnamed socket s. When a socket is created with socket(2), 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 VALUE
On success, zero is returned. On error, −1 is returned, and errno is set appropriately.

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.
ENOMEM   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.
ENOTDIR   A null pathname was specified.
ENOTDIR   Too many symbolic links were encountered in translating the pathname in name.
ENOTDIR   A component of the path prefix of the pathname in name does not exist.
ENOTDIR   A component of the path prefix of the pathname in name is not a directory.
EROFS   The inode would reside on a read-only file system.

SEE ALSO
unlink(2), socket(2), attributes(5), socket(5)

NOTES
Binding a name in the UNIX domain creates a socket in the file system that must be deleted by the caller when it is no longer needed (using unlink(2)). The rules used in name binding vary between communication domains.
NAME
fopen, fdopen, fileno – stream open functions

SYNOPSIS
#include <stdio.h>
FILE *

fopen(const char *

mode);

NAME
ipv6/socket(7)

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

 FILE *

socket(AF_INET6, SOCK_STREAM, 0);
 FILE *

tcp6_socket = socket(AF_INET6, SOCK_STREAM, 0);

FILE *

fdopen(int

fildes

mode);

NAME
ipv6/sock et(7)

SYNOPSIS
#include <stdio.h>
FILE *

fopen(const char *

mode);

SYNOPSIS
#include <stdio.h>
FILE *

fdopen(int

fildes

mode);

FILE *

fileno(FILE *

stream);

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.

The function fileno() examines the argument stream and returns its integer descriptor. The fclose() function flushes the stream pointed to by stream (writing any buffered output data using flush(3)) and closes the underlying file descriptor.

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. Upon successful completion of fclose, 0 is returned. Otherwise, EOF is returned and errno is set to indicate the error.

ERRORS
EINVAL
The mode provided to fopen, fdopen, or freopen was invalid.
EBADF
The file descriptor underlying stream passed to fclose is not valid.

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 fdopen(2).

DESCRIPTION
Linux 2.2 optionally implements the Internet Protocol, version 6. This man page contains a description of socket(7).

TCP6_SOCKET = socket(AF_INET6, SOCK_STREAM, 0);
TCP6_SOCKET = socket(AF_INET6, SOCK_DGRAM, protocol);
TCP6_SOCKET = socket(AF_INET6, SOCK_RAW, protocol);

ADDRESS FORMAT
struct sockaddr_in6 {
  struct in6_addr

  sin6_family; /* AF_INET6 */
  uint6_t

  sin6_port; /* Port number */
 struct in6_addr

  sin6_flowinfo; /* IPv6 flow information */
  uint32_t

  sin6_addr; /* IPv6 address */
  uint32_t

  sin6_scope_id; /* Scope ID (new in 2.4.1) */
};

struct in6_addr {
  unsigned char

  sin6_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 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)).

RETURN VALUES
-1 is returned if an error occurs. Otherwise the return value is a descriptor referencing the socket.

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
msg(3), ip(7)
NAME
listen -- listen for connections on a socket

SYNOPSIS
#include <sys/types.h>
/* See NOTES */
#include <sys/socket.h>
int listen(int sockfd, int backlog);

DESCRIPTION
listen() marks the socket referred to by sockfd as a passive socket, that is, as a socket that will be used to
accept incoming connection requests using accept(2).

The sockfd argument is a file descriptor that refers to a socket of type SOCK_STREAM or SOCK_SEQ-
PACKET.

The backlog argument defines the maximum length to which the queue of pending connections for sockfd
may grow. If a connection request arrives when the queue is full, the client may receive an error with an
indication of ECONNREFUSED or, if the underlying protocol supports retransmission, the request may be
ignored so that a later reattempt at connection succeeds.

RETURN VALUE
On success, zero is returned. On error, -1 is returned, and errno is set appropriately.

ERRORS
EADDRINUSE
Another socket is already listening on the same port.
EBADF
The argument sockfd is not a valid descriptor.
ENOTSOCK
The argument sockfd is not a socket.

NOTES
To accept connections, the following steps are performed:
1. A socket is created with socket(2).
2. The socket is bound to a local address using bind(2), so that other sockets may be connect(2)ed to it.
3. A willingness to accept incoming connections and a queue limit for incoming connections are
specified with listen().
4. Connections are accepted with accept(2).

If the backlog argument is greater than the value in /proc/sys/net/core/somaxconn, then it is silently truncated
to that value; the default value in this file is 128.

EXAMPLE
See bind(2).

SEE ALSO
accept(2), bind(2), connect(2), socket(2), socket(7)

pthread_create/pthread_exit(3)
NAME
pthread_detach − put a running thread in the detached state

SYNOPSIS
#include <pthread.h>
int pthread_detach(pthread_t th);

DESCRIPTION
pthread_detach put the thread th in the detached state. This guarantees that the memory resources consumed by th will be freed immediately when th terminates. However, this prevents other threads from synchronizing on the termination of th using pthread_join.

A thread can be created initially in the detached state, using the detachstate attribute to pthread_create(3). In contrast, pthread_detach applies to threads created in the joinable state, and which need to be put in the detached state later.

After pthread_detach completes, subsequent attempts to perform pthread_join on th will fail. If another thread is already joining the thread th at the time pthread_detach is called, pthread_detach does nothing and leaves th in the joinable state.

RETURN VALUE
On success, 0 is returned. On error, a non-zero error code is returned.

ERRORS
ESRCH
No thread could be found corresponding to that specified by th
EINVAL
the thread th is already in the detached state

AUTHOR
Xavier Leroy <Xavier.Leroy@inria.fr>

SEE ALSO
pthread_create(3), pthread_join(3), pthread_attr_setdetachstate(3), sigaction(2)
NAME
strerror — get error message string

SYNOPSIS
#include <string.h>
char *strerror(int errnum);

DESCRIPTION
For strerror(): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard. The strerror() function shall map the error number in errnum to a locale-dependent error message string and shall return a pointer to it. Typically, the values for errnum come from errno, but strerror() shall map any value of type int to a message.

The application shall not modify the string returned. The returned string pointer might be invalidated or the string content might be overwritten by a subsequent call to strerror() in the same thread. The returned pointer and the string content might also be invalidated if the calling thread is terminated.

The contents of the error message strings returned by strerror() should be determined by the setting of the LC_MESSAGES category in the current locale.

The implementation shall behave as if no function defined in this volume of POSIX.1-2008 calls strerror(). The strerror() function shall not change the setting of errno if successful.

If the value of errnum is a valid error number, the message string shall indicate what error occurred; if the value of errnum is zero, the message string shall either be an empty string or indicate that no error occurred; otherwise, if these functions complete successfully, the message string shall indicate that an unknown error occurred.

RETURN VALUE
Upon completion, whether successful or not, strerror() shall return a pointer to the generated message string. On error errno may be set, but no return value is reserved to indicate an error.

ERRORS
These functions may fail if:

EINVAL
The value of errnum is neither a valid error number nor zero.

SEE ALSO
perror(3)

string(3)

NAME
strcat, strchr, strcmp, strcpy, strdup, strlen, strncat, strncmp, strncpy, strrchr, strstr, strtok — string operations

SYNOPSIS
#include <string.h>

d char *strcat(char * dest, const char * src);

d char *strchr(const char *, int c);

d int strcmp(const char *, const char *);

d int strncmp(const char *, const char *, size_t n);

d char *strncpy(char *, const char *, size_t n);

d char *strrchr(const char *, int c);

d char *strstr(const char *, const char *);

d char *strtok(char *, const char *);

DESCRIPTION
The string functions perform operations on null-terminated strings.