# accept(2) — Linux manual page

Search online pages

ACCEPT(2)                    Linux Programmer's Manual                    ACCEPT(2)

## NAME      top

    accept, accept4 - accept a connection on a socket

## SYNOPSIS      top

    #include <sys/types.h>          /* See NOTES */
    #include <sys/socket.h>

    int accept(int *sockfd*, struct sockaddr *\*addr*, socklen_t *\*addrlen*);

    #define _GNU_SOURCE             /* See feature_test_macros(7) */
    #include <sys/socket.h>

    int accept4(int *sockfd*, struct sockaddr *\*addr*,
                socklen_t *\*addrlen*, int *flags*);

## DESCRIPTION      top

    The **accept**() system call is used with connection-based socket types
    (**SOCK_STREAM**, **SOCK_SEQPACKET**).  It extracts the first connection
    request on the queue of pending connections for the listening socket,
    *sockfd*, creates a new connected socket, and returns a new file
    descriptor referring to that socket.  The newly created socket is not
    in the listening state.  The original socket *sockfd* is unaffected by
    this call.

    The argument *sockfd* is a socket that has been created with socket(2),
    bound to a local address with bind(2), and is listening for
    connections after a listen(2).

    The argument *addr* is a pointer to a *sockaddr* structure.  This
    structure is filled in with the address of the peer socket, as known
    to the communications layer.  The exact format of the address
    returned *addr* is determined by the socket's address family (see
    socket(2) and the respective protocol man pages).  When *addr* is NULL,
    nothing is filled in; in this case, *addrlen* is not used, and should
    also be NULL.

    The *addrlen* argument is a value-result argument: the caller must
    initialize it to contain the size (in bytes) of the structure pointed
    to by *addr*; on return it will contain the actual size of the peer
    address.

The returned address is truncated if the buffer provided is too small; in this case, *addrlen* will return a value greater than was supplied to the call.

If no pending connections are present on the queue, and the socket is not marked as nonblocking, **accept**() blocks the caller until a connection is present.  If the socket is marked nonblocking and no pending connections are present on the queue, **accept**() fails with the error **EAGAIN** or **EWOULDBLOCK.**

In order to be notified of incoming connections on a socket, you can use select(2), poll(2), or epoll(7).  A readable event will be delivered when a new connection is attempted and you may then call **accept**() to get a socket for that connection.  Alternatively, you can set the socket to deliver **SIGIO** when activity occurs on a socket; see socket(7) for details.

If *flags* is 0, then **accept4**() is the same as **accept**().  The following values can be bitwise ORed in *flags* to obtain different behavior:

**SOCK_NONBLOCK**     Set the **O_NONBLOCK** file status flag on the open file description (see open(2)) referred to by the new file descriptor.  Using this flag saves extra calls to fcntl(2) to achieve the same result.

**SOCK_CLOEXEC**      Set the close-on-exec (**FD_CLOEXEC**) flag on the new file descriptor.  See the description of the **O_CLOEXEC** flag in open(2) for reasons why this may be useful.

## RETURN VALUE       top

On success, these system calls return a file descriptor for the accepted socket (a nonnegative integer).  On error, -1 is returned, *errno* is set appropriately, and *addrlen* is left unchanged.

### Error handling
Linux **accept**() (and **accept4**()) passes already-pending network errors on the new socket as an error code from **accept**().  This behavior differs from other BSD socket implementations.  For reliable operation the application should detect the network errors defined for the protocol after **accept**() and treat them like **EAGAIN** by retrying.  In the case of TCP/IP, these are **ENETDOWN**, **EPROTO**, **ENOPROTOOPT**, **EHOSTDOWN**, **ENONET**, **EHOSTUNREACH**, **EOPNOTSUPP**, and **ENETUNREACH.**

## ERRORS       top

**EAGAIN** or **EWOULDBLOCK**
        The socket is marked nonblocking and no connections are present to be accepted.  POSIX.1-2001 and POSIX.1-2008 allow either error to be returned for this case, and do not require these constants to have the same value, so a portable application should check for both possibilities.

**EBADF**    *sockfd* is not an open file descriptor.

**ECONNABORTED**
        A connection has been aborted.

**EFAULT** The *addr* argument is not in a writable part of the user
        address space.

**EINTR**  The system call was interrupted by a signal that was caught
        before a valid connection arrived; see signal(7).

**EINVAL** Socket is not listening for connections, or *addrlen* is invalid
        (e.g., is negative).

**EINVAL** (**accept4**()) invalid value in *flags*.

**EMFILE** The per-process limit on the number of open file descriptors
        has been reached.

**ENFILE** The system-wide limit on the total number of open files has
        been reached.

**ENOBUFS**, **ENOMEM**
        Not enough free memory.  This often means that the memory
        allocation is limited by the socket buffer limits, not by the
        system memory.

**ENOTSOCK**
        The file descriptor *sockfd* does not refer to a socket.

**EOPNOTSUPP**
        The referenced socket is not of type **SOCK_STREAM**.

**EPROTO** Protocol error.

In addition, Linux **accept**() may fail if:

**EPERM**  Firewall rules forbid connection.

In addition, network errors for the new socket and as defined for the
protocol may be returned.  Various Linux kernels can return other
errors such as **ENOSR, ESOCKTNOSUPPORT, EPROTONOSUPPORT, ETIMEDOUT**.
The value **ERESTARTSYS** may be seen during a trace.


## VERSIONS       top

        The **accept4**() system call is available starting with Linux 2.6.28;
        support in glibc is available starting with version 2.10.


## CONFORMING TO       top

        **accept**(): POSIX.1-2001, POSIX.1-2008, SVr4, 4.4BSD (**accept**() first
        appeared in 4.2BSD).

        **accept4**() is a nonstandard Linux extension.

        On Linux, the new socket returned by **accept**() does *not* inherit file
        status flags such as **O_NONBLOCK** and **O_ASYNC** from the listening

socket.  This behavior differs from the canonical BSD sockets
implementation.  Portable programs should not rely on inheritance or
noninheritance of file status flags and always explicitly set all
required flags on the socket returned from **accept**().

## NOTES          top

POSIX.1-2001 does not require the inclusion of *<sys/types.h>*, and
this header file is not required on Linux.  However, some historical
(BSD) implementations required this header file, and portable
applications are probably wise to include it.

There may not always be a connection waiting after a **SIGIO** is
delivered or select(2), poll(2), or epoll(7) return a readability
event because the connection might have been removed by an
asynchronous network error or another thread before **accept**() is
called.  If this happens, then the call will block waiting for the
next connection to arrive.  To ensure that **accept**() never blocks, the
passed socket *sockfd* needs to have the **O_NONBLOCK** flag set (see
socket(7)).

For certain protocols which require an explicit confirmation, such as
DECnet, **accept**() can be thought of as merely dequeuing the next
connection request and not implying confirmation.  Confirmation can
be implied by a normal read or write on the new file descriptor, and
rejection can be implied by closing the new socket.  Currently, only
DECnet has these semantics on Linux.

### The socklen_t type
In the original BSD sockets implementation (and on other older
systems) the third argument of **accept**() was declared as an *int \**.  A
POSIX.1g draft standard wanted to change it into a *size_t \**C; later
POSIX standards and glibc 2.x have *socklen_t \** .

## EXAMPLES          top

See bind(2).

## SEE ALSO          top

bind(2),  connect(2),  listen(2),  select(2),  socket(2),  socket(7)

## COLOPHON          top

This page is part of release 5.08 of the Linux *man-pages* project.  A
description of the project, information about reporting bugs, and the
latest version of this page, can be found at
https://www.kernel.org/doc/man-pages/.

**Linux**                              **2020-04-11**                              **ACCEPT(2)**

Pages that refer to this page: bind(2),  connect(2),  getpeername(2),  getsockname(2),
getsockopt(2),  listen(2),  _newselect(2),  pselect(2),  pselect6(2),  recv(2),  recvfrom(2),  recvmsg(2),
select(2),  select_tut(2),  setsockopt(2),  socket(2),  socketcall(2),  syscalls(2),  endhostent(3),

fd_clr(3), FD_CLR(3), fd_isset(3), FD_ISSET(3), fd_set(3), FD_SET(3), fd_zero(3), FD_ZERO(3), freeaddrinfo(3), gai_strerror(3), getaddrinfo(3), gethostbyaddr(3), gethostbyaddr_r(3), gethostbyname2(3), gethostbyname2_r(3), gethostbyname(3), gethostbyname_r(3), gethostent(3), gethostent_r(3), getnameinfo(3), h_errno(3), herror(3), hstrerror(3), sethostent(3), capabilities(7), ddp(7), ip(7), sctp(7), signal(7), signal-safety(7), sock_diag(7), socket(7), tcp(7), unix(7)

## Copyright and license for this manual page

HTML rendering created 2020-09-19 by Michael Kerrisk, author of *The Linux Programming Interface*, maintainer of the Linux *man-pages* project.

For details of in-depth **Linux/UNIX system programming training courses** that I teach, look here.

Hosting by jambit GmbH.