Full credit is given to the above companies including the Operating System (OS) that this PDF file was generated!

Rocky Enterprise Linux 9.2 Manual Pages on command 'socket.7'

$ man socket.7

SOCKET(7) Linux Programmer's Manual SOCKET(7)

NAME

socket - Linux socket interface

SYNOPSIS

#include <sys/socket.h>

sockfd = socket(int socket_family, int socket_type, int protocol);

DESCRIPTION

This manual page describes the Linux networking socket layer user interface. The BSD com?

patible sockets are the uniform interface between the user process and the network proto?

col stacks in the kernel. The protocol modules are grouped into protocol families such as

AF_INET, AF_IPX, and AF_PACKET, and socket types such as SOCK_STREAM or SOCK_DGRAM. See

socket(2) for more information on families and types.

Socket-layer functions

These functions are used by the user process to send or receive packets and to do other

socket operations. For more information see their respective manual pages.

socket(2) creates a socket, connect(2) connects a socket to a remote socket address, the

bind(2) function binds a socket to a local socket address, listen(2) tells the socket that

new connections shall be accepted, and accept(2) is used to get a new socket with a new

incoming connection. socketpair(2) returns two connected anonymous sockets (implemented

only for a few local families like AF_UNIX)

send(2), sendto(2), and sendmsg(2) send data over a socket, and recv(2), recvfrom(2),

recvmsg(2) receive data from a socket. poll(2) and select(2) wait for arriving data or a

readiness to send data. In addition, the standard I/O operations like write(2), writev(2), sendfile(2), read(2), and readv(2) can be used to read and write data.

getsockname(2) returns the local socket address and getpeername(2) returns the remote socket address. getsockopt(2) and setsockopt(2) are used to set or get socket layer or protocol options. ioctl(2) can be used to set or read some other options.

close(2) is used to close a socket. shutdown(2) closes parts of a full-duplex socket con? nection.

Seeking, or calling pread(2) or pwrite(2) with a nonzero position is not supported on sockets.

It is possible to do nonblocking I/O on sockets by setting the O_NONBLOCK flag on a socket file descriptor using fcntl(2). Then all operations that would block will (usually) re? turn with EAGAIN (operation should be retried later); connect(2) will return EINPROGRESS error. The user can then wait for various events via poll(2) or select(2).

??

? I/O events ?

??

?Event ? Poll flag ? Occurrence ?

??

?Read ? POLLIN ? New data arrived. ?

??

?Read ? POLLIN ? A connection setup has been completed (for ?

? ? ? connection-oriented sockets) ?

??

?Read ? POLLHUP ? A disconnection request has been initiated ?

? ? ? by the other end. ?

??

?Read ? POLLHUP ? A connection is broken (only for connec? ?

? ? ? tion-oriented protocols). When the socket ?

? ? ? is written SIGPIPE is also sent. ?

??

?Write ? POLLOUT ? Socket has enough send buffer space for ?

? ? ? writing new data. ?

??

?Read/Write ? POLLIN | ? An outgoing connect(2) finished. ?

?   ? POLLOUT   ?   ?

??

?Read/Write ? POLLERR   ? An asynchronous error occurred.   ?

??

?Read/Write ? POLLHUP   ? The other end has shut down one direction. ?

??

?Exception ? POLLPRI   ? Urgent data arrived. SIGURG is sent then. ?

??

An alternative to poll(2) and select(2) is to let the kernel inform the application about events via a SIGIO signal. For that the O_ASYNC flag must be set on a socket file de?scriptor via fcntl(2) and a valid signal handler for SIGIO must be installed via sigac?tion(2). See the Signals discussion below.

Socket address structures

Each socket domain has its own format for socket addresses, with a domain-specific address structure. Each of these structures begins with an integer "family" field (typed as sa_family_t) that indicates the type of the address structure. This allows the various system calls (e.g., connect(2), bind(2), accept(2), getsockname(2), getpeername(2)), which are generic to all socket domains, to determine the domain of a particular socket address.

To allow any type of socket address to be passed to interfaces in the sockets API, the type struct sockaddr is defined. The purpose of this type is purely to allow casting of domain-specific socket address types to a "generic" type, so as to avoid compiler warnings about type mismatches in calls to the sockets API.

In addition, the sockets API provides the data type struct sockaddr_storage. This type is suitable to accommodate all supported domain-specific socket address structures; it is large enough and is aligned properly. (In particular, it is large enough to hold IPv6 socket addresses.) The structure includes the following field, which can be used to iden?tify the type of socket address actually stored in the structure:

```
sa_family_t ss_family;
```

The sockaddr_storage structure is useful in programs that must handle socket addresses in a generic way (e.g., programs that must deal with both IPv4 and IPv6 socket addresses).

Socket options

The socket options listed below can be set by using setsockopt(2) and read with getsock?opt(2) with the socket level set to SOL_SOCKET for all sockets. Unless otherwise noted, optval is a pointer to an int.

SO_ACCEPTCONN

Returns a value indicating whether or not this socket has been marked to accept connections with listen(2). The value 0 indicates that this is not a listening socket, the value 1 indicates that this is a listening socket. This socket option is read-only.

SO_ATTACH_FILTER (since Linux 2.2), SO_ATTACH_BPF (since Linux 3.19)

Attach a classic BPF (SO_ATTACH_FILTER) or an extended BPF (SO_ATTACH_BPF) program to the socket for use as a filter of incoming packets. A packet will be dropped if the filter program returns zero. If the filter program returns a nonzero value which is less than the packet's data length, the packet will be truncated to the length returned. If the value returned by the filter is greater than or equal to the packet's data length, the packet is allowed to proceed unmodified.

The argument for SO_ATTACH_FILTER is a sock_fprog structure, defined in <linux/fil?ter.h>:

```
struct sock_fprog {
    unsigned short      len;
    struct sock_filter *filter;
};
```

The argument for SO_ATTACH_BPF is a file descriptor returned by the bpf(2) system call and must refer to a program of type BPF_PROG_TYPE_SOCKET_FILTER.

These options may be set multiple times for a given socket, each time replacing the previous filter program. The classic and extended versions may be called on the same socket, but the previous filter will always be replaced such that a socket never has more than one filter defined.

Both classic and extended BPF are explained in the kernel source file Documenta?tion/networking/filter.txt

SO_ATTACH_REUSEPORT_CBPF, SO_ATTACH_REUSEPORT_EBPF

For use with the SO_REUSEPORT option, these options allow the user to set a classic BPF (SO_ATTACH_REUSEPORT_CBPF) or an extended BPF (SO_ATTACH_REUSEPORT_EBPF) pro?gram which defines how packets are assigned to the sockets in the reuseport group (that is, all sockets which have SO_REUSEPORT set and are using the same local ad?dress to receive packets).

The BPF program must return an index between 0 and N-1 representing the socket which should receive the packet (where N is the number of sockets in the group). If the BPF program returns an invalid index, socket selection will fall back to the plain SO_REUSEPORT mechanism.

Sockets are numbered in the order in which they are added to the group (that is, the order of bind(2) calls for UDP sockets or the order of listen(2) calls for TCP sockets). New sockets added to a reuseport group will inherit the BPF program. When a socket is removed from a reuseport group (via close(2)), the last socket in the group will be moved into the closed socket's position.

These options may be set repeatedly at any time on any socket in the group to re? place the current BPF program used by all sockets in the group.

SO_ATTACH_REUSEPORT_CBPF takes the same argument type as SO_ATTACH_FILTER and SO_ATTACH_REUSEPORT_EBPF takes the same argument type as SO_ATTACH_BPF.

UDP support for this feature is available since Linux 4.5; TCP support is available since Linux 4.6.

SO_BINDTODEVICE

Bind this socket to a particular device like ?eth0?, as specified in the passed in? terface name. If the name is an empty string or the option length is zero, the socket device binding is removed. The passed option is a variable-length null-ter? minated interface name string with the maximum size of IFNAMSIZ. If a socket is bound to an interface, only packets received from that particular interface are processed by the socket. Note that this works only for some socket types, particu? larly AF_INET sockets. It is not supported for packet sockets (use normal bind(2) there).

Before Linux 3.8, this socket option could be set, but could not retrieved with getsockopt(2). Since Linux 3.8, it is readable. The optlen argument should con? tain the buffer size available to receive the device name and is recommended to be IFNAMSIZ bytes. The real device name length is reported back in the optlen argu? ment.

SO_BROADCAST

Set or get the broadcast flag. When enabled, datagram sockets are allowed to send packets to a broadcast address. This option has no effect on stream-oriented sock? ets.

SO_BSDCOMPAT

Enable BSD bug-to-bug compatibility. This is used by the UDP protocol module in Linux 2.0 and 2.2. If enabled, ICMP errors received for a UDP socket will not be passed to the user program. In later kernel versions, support for this option has been phased out: Linux 2.4 silently ignores it, and Linux 2.6 generates a kernel warning (printk()) if a program uses this option. Linux 2.0 also enabled BSD bug-to-bug compatibility options (random header changing, skipping of the broadcast flag) for raw sockets with this option, but that was removed in Linux 2.2.

SO_DEBUG

Enable socket debugging. Allowed only for processes with the CAP_NET_ADMIN capa? bility or an effective user ID of 0.

SO_DETACH_FILTER (since Linux 2.2), SO_DETACH_BPF (since Linux 3.19)

These two options, which are synonyms, may be used to remove the classic or ex? tended BPF program attached to a socket with either SO_ATTACH_FILTER or SO_AT? TACH_BPF. The option value is ignored.

SO_DOMAIN (since Linux 2.6.32)

Retrieves the socket domain as an integer, returning a value such as AF_INET6. See socket(2) for details. This socket option is read-only.

SO_ERROR

Get and clear the pending socket error. This socket option is read-only. Expects an integer.

SO_DONTROUTE

Don't send via a gateway, send only to directly connected hosts. The same effect can be achieved by setting the MSG_DONTROUTE flag on a socket send(2) operation. Expects an integer boolean flag.

SO_INCOMING_CPU (gettable since Linux 3.19, settable since Linux 4.4)

Sets or gets the CPU affinity of a socket. Expects an integer flag.

```
int cpu = 1;
setsockopt(fd, SOL_SOCKET, SO_INCOMING_CPU, &cpu,
           sizeof(cpu));
```

Because all of the packets for a single stream (i.e., all packets for the same 4-tuple) arrive on the single RX queue that is associated with a particular CPU, the typical use case is to employ one listening process per RX queue, with the in?

coming flow being handled by a listener on the same CPU that is handling the RX queue. This provides optimal NUMA behavior and keeps CPU caches hot.

SO_INCOMING_NAPI_ID (gettable since Linux 4.12)

Returns a system-level unique ID called NAPI ID that is associated with a RX queue on which the last packet associated with that socket is received.

This can be used by an application to split the incoming flows among worker threads based on the RX queue on which the packets associated with the flows are received. It allows each worker thread to be associated with a NIC HW receive queue and ser? vice all the connection requests received on that RX queue. This mapping between a app thread and a HW NIC queue streamlines the flow of data from the NIC to the ap? plication.

SO_KEEPALIVE

Enable sending of keep-alive messages on connection-oriented sockets. Expects an integer boolean flag.

SO_LINGER

Sets or gets the SO_LINGER option. The argument is a linger structure.

```
struct linger {
    int l_onoff;    /* linger active */
    int l_linger;   /* how many seconds to linger for */
};
```

When enabled, a close(2) or shutdown(2) will not return until all queued messages for the socket have been successfully sent or the linger timeout has been reached. Otherwise, the call returns immediately and the closing is done in the background. When the socket is closed as part of exit(2), it always lingers in the background.

SO_LOCK_FILTER

When set, this option will prevent changing the filters associated with the socket. These filters include any set using the socket options SO_ATTACH_FILTER, SO_AT? TACH_BPF, SO_ATTACH_REUSEPORT_CBPF, and SO_ATTACH_REUSEPORT_EBPF.

The typical use case is for a privileged process to set up a raw socket (an opera? tion that requires the CAP_NET_RAW capability), apply a restrictive filter, set the SO_LOCK_FILTER option, and then either drop its privileges or pass the socket file descriptor to an unprivileged process via a UNIX domain socket.

Once the SO_LOCK_FILTER option has been enabled, attempts to change or remove the filter attached to a socket, or to disable the SO_LOCK_FILTER option will fail with the error EPERM.

SO_MARK (since Linux 2.6.25)

Set the mark for each packet sent through this socket (similar to the netfilter MARK target but socket-based). Changing the mark can be used for mark-based rout? ing without netfilter or for packet filtering. Setting this option requires the CAP_NET_ADMIN capability.

SO_OOBINLINE

If this option is enabled, out-of-band data is directly placed into the receive data stream. Otherwise, out-of-band data is passed only when the MSG_OOB flag is set during receiving.

SO_PASSCRED

Enable or disable the receiving of the SCM_CREDENTIALS control message. For more information see unix(7).

SO_PASSSEC

Enable or disable the receiving of the SCM_SECURITY control message. For more in? formation see unix(7).

SO_PEEK_OFF (since Linux 3.4)

This option, which is currently supported only for unix(7) sockets, sets the value of the "peek offset" for the recv(2) system call when used with MSG_PEEK flag.

When this option is set to a negative value (it is set to -1 for all new sockets), traditional behavior is provided: recv(2) with the MSG_PEEK flag will peek data from the front of the queue.

When the option is set to a value greater than or equal to zero, then the next peek at data queued in the socket will occur at the byte offset specified by the option value. At the same time, the "peek offset" will be incremented by the number of bytes that were peeked from the queue, so that a subsequent peek will return the next data in the queue.

If data is removed from the front of the queue via a call to recv(2) (or similar) without the MSG_PEEK flag, the "peek offset" will be decreased by the number of bytes removed. In other words, receiving data without the MSG_PEEK flag will cause the "peek offset" to be adjusted to maintain the correct relative position in the queued data, so that a subsequent peek will retrieve the data that would have been retrieved had the data not been removed.

For datagram sockets, if the "peek offset" points to the middle of a packet, the data returned will be marked with the MSG_TRUNC flag.

The following example serves to illustrate the use of SO_PEEK_OFF. Suppose a stream socket has the following queued input data:

```
aabbccddeeff
```

The following sequence of recv(2) calls would have the effect noted in the com? ments:

```
int ov = 4;                  // Set peek offset to 4
setsockopt(fd, SOL_SOCKET, SO_PEEK_OFF, &ov, sizeof(ov));
recv(fd, buf, 2, MSG_PEEK);  // Peeks "cc"; offset set to 6
recv(fd, buf, 2, MSG_PEEK);  // Peeks "dd"; offset set to 8
recv(fd, buf, 2, 0);         // Reads "aa"; offset set to 6
recv(fd, buf, 2, MSG_PEEK);  // Peeks "ee"; offset set to 8
```

SO_PEERCRED

Return the credentials of the peer process connected to this socket. For further details, see unix(7).

SO_PEERSEC (since Linux 2.6.2)

Return the security context of the peer socket connected to this socket. For fur? ther details, see unix(7) and ip(7).

SO_PRIORITY

Set the protocol-defined priority for all packets to be sent on this socket. Linux uses this value to order the networking queues: packets with a higher priority may be processed first depending on the selected device queueing discipline. Setting a priority outside the range 0 to 6 requires the CAP_NET_ADMIN capability.

SO_PROTOCOL (since Linux 2.6.32)

Retrieves the socket protocol as an integer, returning a value such as IP? PROTO_SCTP. See socket(2) for details. This socket option is read-only.

SO_RCVBUF

Sets or gets the maximum socket receive buffer in bytes. The kernel doubles this value (to allow space for bookkeeping overhead) when it is set using setsockopt(2), and this doubled value is returned by getsockopt(2). The default value is set by the /proc/sys/net/core/rmem_default file, and the maximum allowed value is set by the /proc/sys/net/core/rmem_max file. The minimum (doubled) value for this option is 256.

SO_RCVBUFFORCE (since Linux 2.6.14)

Using this socket option, a privileged (CAP_NET_ADMIN) process can perform the same task as SO_RCVBUF, but the rmem_max limit can be overridden.

SO_RCVLOWAT and SO_SNDLOWAT

Specify the minimum number of bytes in the buffer until the socket layer will pass the data to the protocol (SO_SNDLOWAT) or the user on receiving (SO_RCVLOWAT). These two values are initialized to 1. SO_SNDLOWAT is not changeable on Linux (setsockopt(2) fails with the error ENOPROTOOPT). SO_RCVLOWAT is changeable only since Linux 2.4.

Before Linux 2.6.28 select(2), poll(2), and epoll(7) did not respect the SO_RCVLOWAT setting on Linux, and indicated a socket as readable when even a single byte of data was available. A subsequent read from the socket would then block un?
til SO_RCVLOWAT bytes are available. Since Linux 2.6.28, select(2), poll(2), and epoll(7) indicate a socket as readable only if at least SO_RCVLOWAT bytes are available.

SO_RCVTIMEO and SO_SNDTIMEO

Specify the receiving or sending timeouts until reporting an error. The argument is a struct timeval. If an input or output function blocks for this period of time, and data has been sent or received, the return value of that function will be the amount of data transferred; if no data has been transferred and the timeout has been reached, then -1 is returned with errno set to EAGAIN or EWOULDBLOCK, or EIN?
PROGRESS (for connect(2)) just as if the socket was specified to be nonblocking. If the timeout is set to zero (the default), then the operation will never timeout. Timeouts only have effect for system calls that perform socket I/O (e.g., read(2), recvmsg(2), send(2), sendmsg(2)); timeouts have no effect for select(2), poll(2), epoll_wait(2), and so on.

SO_REUSEADDR

Indicates that the rules used in validating addresses supplied in a bind(2) call should allow reuse of local addresses. For AF_INET sockets this means that a socket may bind, except when there is an active listening socket bound to the ad?
dress. When the listening socket is bound to INADDR_ANY with a specific port then

it is not possible to bind to this port for any local address. Argument is an in?
teger boolean flag.

SO_REUSEPORT (since Linux 3.9)

Permits multiple AF_INET or AF_INET6 sockets to be bound to an identical socket ad?
dress. This option must be set on each socket (including the first socket) prior
to calling bind(2) on the socket. To prevent port hijacking, all of the processes
binding to the same address must have the same effective UID. This option can be
employed with both TCP and UDP sockets.

For TCP sockets, this option allows accept(2) load distribution in a multi-threaded
server to be improved by using a distinct listener socket for each thread. This
provides improved load distribution as compared to traditional techniques such us?
ing a single accept(2)ing thread that distributes connections, or having multiple
threads that compete to accept(2) from the same socket.

For UDP sockets, the use of this option can provide better distribution of incoming
datagrams to multiple processes (or threads) as compared to the traditional tech?
nique of having multiple processes compete to receive datagrams on the same socket.

SO_RXQ_OVFL (since Linux 2.6.33)

Indicates that an unsigned 32-bit value ancillary message (cmsg) should be attached
to received skbs indicating the number of packets dropped by the socket since its
creation.

SO_SELECT_ERR_QUEUE (since Linux 3.10)

When this option is set on a socket, an error condition on a socket causes notifi?
cation not only via the exceptfds set of select(2). Similarly, poll(2) also re?
turns a POLLPRI whenever an POLLERR event is returned.

Background: this option was added when waking up on an error condition occurred
only via the readfds and writefds sets of select(2). The option was added to allow
monitoring for error conditions via the exceptfds argument without simultaneously
having to receive notifications (via readfds) for regular data that can be read
from the socket. After changes in Linux 4.16, the use of this flag to achieve the
desired notifications is no longer necessary. This option is nevertheless retained
for backwards compatibility.

SO_SNDBUF

Sets or gets the maximum socket send buffer in bytes. The kernel doubles this value (to allow space for bookkeeping overhead) when it is set using setsockopt(2), and this doubled value is returned by getsockopt(2). The default value is set by the /proc/sys/net/core/wmem_default file and the maximum allowed value is set by the /proc/sys/net/core/wmem_max file. The minimum (doubled) value for this option is 2048.

SO_SNDBUFFORCE (since Linux 2.6.14)

Using this socket option, a privileged (CAP_NET_ADMIN) process can perform the same task as SO_SNDBUF, but the wmem_max limit can be overridden.

SO_TIMESTAMP

Enable or disable the receiving of the SO_TIMESTAMP control message. The timestamp control message is sent with level SOL_SOCKET and a cmsg_type of SCM_TIMESTAMP. The cmsg_data field is a struct timeval indicating the reception time of the last packet passed to the user in this call. See cmsg(3) for details on control mes? sages.

SO_TIMESTAMPNS (since Linux 2.6.22)

Enable or disable the receiving of the SO_TIMESTAMPNS control message. The time? stamp control message is sent with level SOL_SOCKET and a cmsg_type of SCM_TIMES? TAMPNS. The cmsg_data field is a struct timespec indicating the reception time of the last packet passed to the user in this call. The clock used for the timestamp is CLOCK_REALTIME. See cmsg(3) for details on control messages.

A socket cannot mix SO_TIMESTAMP and SO_TIMESTAMPNS: the two modes are mutually ex? clusive.

SO_TYPE

Gets the socket type as an integer (e.g., SOCK_STREAM). This socket option is read-only.

SO_BUSY_POLL (since Linux 3.11)

Sets the approximate time in microseconds to busy poll on a blocking receive when there is no data. Increasing this value requires CAP_NET_ADMIN. The default for this option is controlled by the /proc/sys/net/core/busy_read file.

The value in the /proc/sys/net/core/busy_poll file determines how long select(2) and poll(2) will busy poll when they operate on sockets with SO_BUSY_POLL set and no events to report are found.

In both cases, busy polling will only be done when the socket last received data from a network device that supports this option.

While busy polling may improve latency of some applications, care must be taken when using it since this will increase both CPU utilization and power usage.

Signals

When writing onto a connection-oriented socket that has been shut down (by the local or the remote end) SIGPIPE is sent to the writing process and EPIPE is returned. The signal is not sent when the write call specified the MSG_NOSIGNAL flag.

When requested with the FIOSETOWN fcntl(2) or SIOCSPGRP ioctl(2), SIGIO is sent when an I/O event occurs. It is possible to use poll(2) or select(2) in the signal handler to find out which socket the event occurred on. An alternative (in Linux 2.2) is to set a real-time signal using the F_SETSIG fcntl(2); the handler of the real time signal will be called with the file descriptor in the si_fd field of its siginfo_t. See fcntl(2) for more information.

Under some circumstances (e.g., multiple processes accessing a single socket), the condi? tion that caused the SIGIO may have already disappeared when the process reacts to the signal. If this happens, the process should wait again because Linux will resend the sig? nal later.

/proc interfaces

The core socket networking parameters can be accessed via files in the directory /proc/sys/net/core/.

rmem_default

contains the default setting in bytes of the socket receive buffer.

rmem_max

contains the maximum socket receive buffer size in bytes which a user may set by using the SO_RCVBUF socket option.

wmem_default

contains the default setting in bytes of the socket send buffer.

wmem_max

contains the maximum socket send buffer size in bytes which a user may set by using the SO_SNDBUF socket option.

message_cost and message_burst

configure the token bucket filter used to load limit warning messages caused by ex? ternal network events.

netdev_max_backlog

Maximum number of packets in the global input queue.

optmem_max

Maximum length of ancillary data and user control data like the iovecs per socket.

Ioctls

These operations can be accessed using ioctl(2):

error = ioctl(ip_socket, ioctl_type, &value_result);

SIOCGSTAMP

Return a struct timeval with the receive timestamp of the last packet passed to the user. This is useful for accurate round trip time measurements. See setitimer(2) for a description of struct timeval. This ioctl should be used only if the socket options SO_TIMESTAMP and SO_TIMESTAMPNS are not set on the socket. Otherwise, it returns the timestamp of the last packet that was received while SO_TIMESTAMP and SO_TIMESTAMPNS were not set, or it fails if no such packet has been received, (i.e., ioctl(2) returns -1 with errno set to ENOENT).

SIOCSPGRP

Set the process or process group that is to receive SIGIO or SIGURG signals when I/O becomes possible or urgent data is available. The argument is a pointer to a pid_t. For further details, see the description of F_SETOWN in fcntl(2).

FIOASYNC

Change the O_ASYNC flag to enable or disable asynchronous I/O mode of the socket. Asynchronous I/O mode means that the SIGIO signal or the signal set with F_SETSIG is raised when a new I/O event occurs.

Argument is an integer boolean flag. (This operation is synonymous with the use of fcntl(2) to set the O_ASYNC flag.)

SIOCGPGRP

Get the current process or process group that receives SIGIO or SIGURG signals, or 0 when none is set.

Valid fcntl(2) operations:

FIOGETOWN

The same as the SIOCGPGRP ioctl(2).

FIOSETOWN

The same as the SIOCSPGRP ioctl(2).

VERSIONS

SO_BINDTODEVICE was introduced in Linux 2.0.30. SO_PASSCRED is new in Linux 2.2. The /proc interfaces were introduced in Linux 2.2. SO_RCVTIMEO and SO_SNDTIMEO are supported since Linux 2.3.41. Earlier, timeouts were fixed to a protocol-specific setting, and could not be read or written.

NOTES

Linux assumes that half of the send/receive buffer is used for internal kernel structures; thus the values in the corresponding /proc files are twice what can be observed on the wire.

Linux will allow port reuse only with the SO_REUSEADDR option when this option was set both in the previous program that performed a bind(2) to the port and in the program that wants to reuse the port. This differs from some implementations (e.g., FreeBSD) where only the later program needs to set the SO_REUSEADDR option. Typically this difference is invisible, since, for example, a server program is designed to always set this option.

SEE ALSO

wireshark(1), bpf(2), connect(2), getsockopt(2), setsockopt(2), socket(2), pcap(3), ad?
dress_families(7), capabilities(7), ddp(7), ip(7), ipv6(7), packet(7), tcp(7), udp(7), unix(7), tcpdump(8)

COLOPHON

This page is part of release 5.10 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/.