NAME

getsockopt, **setsockopt** - get and set options on sockets

LIBRARY

Standard C Library (libc, -lc)

SYNOPSIS

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

int
getsockopt(*int s, int level, int optname, void * restrict optval, socklen_t * restrict optlen*);

int
setsockopt(*int s, int level, int optname, const void *optval, socklen_t optlen*);

DESCRIPTION

The **getsockopt**() and **setsockopt**() system calls manipulate the *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 the protocol number of the appropriate protocol controlling the option is supplied. For example, to indicate that an option is to be interpreted by the TCP protocol, *level* should be set to the protocol number of TCP; see getprotoent(3).

The *optval* and *optlen* arguments are used to access option values for **setsockopt**(). For **getsockopt**() they identify a buffer in which the value for the requested option(s) are to be returned. For **getsockopt**(), *optlen* is a value-result argument, initially containing the size of the buffer pointed to by *optval*, and modified on return to indicate the actual size of the value returned. If no option value is to be supplied or returned, *optval* may be NULL.

The *optname* argument and any specified options are passed uninterpreted to the appropriate protocol module for interpretation. The include file <*sys/socket.h*> contains definitions for socket level options, described below. Options at other protocol levels vary in format and name; consult the appropriate entries in section 4 of the manual.

Most socket-level options utilize an *int* argument for *optval*. For **setsockopt**(), the argument should be non-zero to enable a boolean option, or zero if the option is to be disabled. SO_LINGER uses a *struct linger* argument, defined in <*sys/socket.h*>, which specifies the desired state of the option and the linger

interval (see below). SO_SNDTIMEO and SO_RCVTIMEO use a *struct timeval* argument, defined in <*sys/time.h*>.

The following options are recognized at the socket level. For protocol-specific options, see protocol manual pages, e.g. ip(4) or tcp(4). Except as noted, each may be examined with **getsockopt**() and set with **setsockopt**().

SO_DEBUG	enables recording of debugging information
SO_REUSEADDR	enables local address reuse
SO_REUSEPORT	enables duplicate address and port bindings
SO_REUSEPORT_LB	enables duplicate address and port bindings with load balancing
SO_KEEPALIVE	enables keep connections alive
SO_DONTROUTE	enables routing bypass for outgoing messages
SO_LINGER	linger on close if data present
SO_BROADCAST	enables permission to transmit broadcast messages
SO_OOBINLINE	enables reception of out-of-band data in band
SO_SNDBUF	set buffer size for output
SO_RCVBUF	set buffer size for input
SO_SNDLOWAT	set minimum count for output
SO_RCVLOWAT	set minimum count for input
SO_SNDTIMEO	set timeout value for output
SO_RCVTIMEO	set timeout value for input
SO_ACCEPTFILTER	set accept filter on listening socket
SO_NOSIGPIPE	controls generation of SIGPIPE for the socket
SO_TIMESTAMP	enables reception of a timestamp with datagrams
SO_BINTIME	enables reception of a timestamp with datagrams
SO_ACCEPTCONN	get listening status of the socket (get only)
SO_DOMAIN	get the domain of the socket (get only)
SO_TYPE	get the type of the socket (get only)
SO_PROTOCOL	get the protocol number for the socket (get only)
SO_PROTOTYPE	SunOS alias for the Linux SO_PROTOCOL (get only)
SO_ERROR	get and clear error on the socket (get only)
SO_RERROR	enables receive error reporting
SO_SETFIB	set the associated FIB (routing table) for the socket (set only)

The following options are recognized in FreeBSD:

SO_LABEL	get MAC label of the socket (get only)
SO_PEERLABEL	get socket’s peer’s MAC label (get only)
SO_LISTENQLIMIT	get backlog limit of the socket (get only)

SO_LISTENQLEN	get complete queue length of the socket (get only)
SO_LISTENINCQLEN	get incomplete queue length of the socket (get only)
SO_USER_COOKIE	set the ’so_user_cookie’ value for the socket (uint32_t, set only)
SO_TS_CLOCK	set specific format of timestamp returned by SO_TIMESTAMP
SO_MAX_PACING_RATE	set the maximum transmit rate in bytes per second for the socket
SO_NO_OFFLOAD	disables protocol offloads
SO_NO_DDP	disables direct data placement offload

SO_DEBUG enables debugging in the underlying protocol modules.

SO_REUSEADDR indicates that the rules used in validating addresses supplied in a bind(2) system call should allow reuse of local addresses.

SO_REUSEPORT allows completely duplicate bindings by multiple processes if they all set SO_REUSEPORT before binding the port. This option permits multiple instances of a program to each receive UDP/IP multicast or broadcast datagrams destined for the bound port.

SO_REUSEPORT_LB allows completely duplicate bindings by multiple sockets if they all set SO_REUSEPORT_LB before binding the port. Incoming TCP and UDP connections are distributed among the participating listening sockets based on a hash function of local port number, and foreign IP address and port number. A maximum of 256 sockets can be bound to the same load-balancing group.

SO_KEEPALIVE enables the periodic transmission of messages on a connected socket. Should the connected party fail to respond to these messages, the connection is considered broken and processes using the socket are notified via a SIGPIPE signal when attempting to send data.

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 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(2) 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 seconds in the **setsockopt**() system call when SO_LINGER is requested). If SO_LINGER is disabled and a close(2) 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.

Broadcast was a privileged operation in earlier versions of the system.

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(2) or read(2) calls without the MSG_OOB flag. Some protocols always behave as if this option is set.

SO_SNDBUF and SO_RCVBUF are options to 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 system places an absolute maximum on these values, which is accessible through the sysctl(3) MIB variable "kern.ipc.maxsockbuf".

SO_SNDLOWAT is an option to set the minimum count for output operations. Most output operations process all of the data supplied by the call, delivering data to the protocol for transmission and blocking as necessary for flow control. Nonblocking output operations will process as much data as permitted subject to flow control without blocking, but will process no data if flow control does not allow the smaller of the low water mark value or the entire request to be processed. A select(2) operation testing the ability to write to a socket will return true only if the low water mark amount could be processed. The default value for SO_SNDLOWAT is set to a convenient size for network efficiency, often 1024.

SO_RCVLOWAT is an option to set the minimum count for input operations. In general, receive calls will block until any (non-zero) amount of data is received, then return with the smaller of the amount available or the amount requested. The default value for SO_RCVLOWAT is 1. If SO_RCVLOWAT is set to a larger value, blocking receive calls normally wait until they have received the smaller of the low water mark value or the requested amount. Receive calls may still return less than the low water mark if an error occurs, a signal is caught, or the type of data next in the receive queue is different from that which was returned.

SO_SNDTIMEO is an option to set a timeout value for output operations. It accepts a *struct timeval* argument with the number of seconds and microseconds used to limit waits for output operations to complete. If a send operation has blocked for this much time, it returns with a partial count or with the error EWOULDBLOCK if no data were sent. In the current implementation, this timer is restarted each time additional data are delivered to the protocol, implying that the limit applies to output portions ranging in size from the low water mark to the high water mark for output.

SO_RCVTIMEO is an option to set a timeout value for input operations. It accepts a *struct timeval* argument with the number of seconds and microseconds used to limit waits for input operations to complete. In the current implementation, this timer is restarted each time additional data are received by the protocol, and thus the limit is in effect an inactivity timer. If a receive operation has been blocked for this much time without receiving additional data, it returns with a short count or with the error EWOULDBLOCK if no data were received.

SO_SETFIB can be used to over-ride the default FIB (routing table) for the given socket. The value must be from 0 to one less than the number returned from the sysctl *net.fibs*.

SO_USER_COOKIE can be used to set the uint32_t so_user_cookie field in the socket. The value is an uint32_t, and can be used in the kernel code that manipulates traffic related to the socket. The default value for the field is 0. As an example, the value can be used as the skipto target or pipe number in **ipfw/dummynet**.

SO_ACCEPTFILTER places an accept_filter(9) on the socket, which will filter incoming connections on a listening stream socket before being presented for accept(2). Once more, listen(2) must be called on the socket before trying to install the filter on it, or else the **setsockopt**() system call will fail.

```
struct  accept_filter_arg {
        char    af_name[16];
        char    af_arg[256-16];
};
```

The *optval* argument should point to a *struct accept_filter_arg* that will select and configure the accept_filter(9). The *af_name* argument should be filled with the name of the accept filter that the application wishes to place on the listening socket. The optional argument *af_arg* can be passed to the accept filter specified by *af_name* to provide additional configuration options at attach time. Passing in an *optval* of NULL will remove the filter.

The SO_NOSIGPIPE option controls generation of the SIGPIPE signal normally sent when writing to a connected socket where the other end has been closed returns with the error EPIPE.

If the SO_TIMESTAMP or SO_BINTIME option is enabled on a SOCK_DGRAM socket, the recvmsg(2) call may return a timestamp corresponding to when the datagram was received. However, it may not, for example due to a resource shortage. The *msg_control* field in the *msghdr* structure points to a buffer that contains a *cmsghdr* structure followed by a *struct timeval* for SO_TIMESTAMP and *struct bintime* for SO_BINTIME. The *cmsghdr* fields have the following values for TIMESTAMP by default:

```
cmsg_len = CMSG_LEN(sizeof(struct timeval));
cmsg_level = SOL_SOCKET;
cmsg_type = SCM_TIMESTAMP;
```

and for SO_BINTIME:

```
cmsg_len = CMSG_LEN(sizeof(struct bintime));
```

```
cmsg_level = SOL_SOCKET;
cmsg_type = SCM_BINTIME;
```

Additional timestamp types are available by following SO_TIMESTAMP with SO_TS_CLOCK, which requests a specific timestamp format to be returned instead of SCM_TIMESTAMP when SO_TIMESTAMP is enabled. These SO_TS_CLOCK values are recognized in FreeBSD:

SO_TS_REALTIME_MICRO
: realtime (SCM_TIMESTAMP, struct timeval), default

SO_TS_BINTIME
: realtime (SCM_BINTIME, struct bintime)

SO_TS_REALTIME
: realtime (SCM_REALTIME, struct timespec)

SO_TS_MONOTONIC
: monotonic time (SCM_MONOTONIC, struct timespec)

SO_ACCEPTCONN, SO_TYPE, SO_PROTOCOL (and its alias SO_PROTOTYPE) and SO_ERROR are options used only with **getsockopt**(). SO_ACCEPTCONN returns whether the socket is currently accepting connections, that is, whether or not the listen(2) system call was invoked on the socket. SO_TYPE returns the type of the socket, such as SOCK_STREAM; it is useful for servers that inherit sockets on startup. SO_PROTOCOL returns the protocol number for the socket, for AF_INET and AF_INET6 address families. 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. SO_RERROR indicates that receive buffer overflows should be handled as errors. Historically receive buffer overflows have been ignored and programs could not tell if they missed messages or messages had been truncated because of overflows. Since programs historically do not expect to get receive overflow errors, this behavior is not the default.

SO_LABEL returns the MAC label of the socket. SO_PEERLABEL returns the MAC label of the socket’s peer. Note that your kernel must be compiled with MAC support. See mac(3) for more information.

SO_LISTENQLIMIT returns the maximal number of queued connections, as set by listen(2). SO_LISTENQLEN returns the number of unaccepted complete connections. SO_LISTENINCQLEN returns the number of unaccepted incomplete connections.

SO_MAX_PACING_RATE instruct the socket and underlying network adapter layers to limit the transfer rate to the given unsigned 32-bit value in bytes per second.

SO_NO_OFFLOAD disables support for protocol offloads. At present, this prevents TCP sockets from

using TCP offload engines. SO_NO_DDP disables support for a specific TCP offload known as direct data placement (DDP). DDP is an offload supported by Chelsio network adapters that permits reassembled TCP data streams to be received via zero-copy in user-supplied buffers using aio_read(2).

RETURN VALUES

Upon successful completion, the value 0 is returned; otherwise the value -1 is returned and the global variable *errno* is set to indicate the error.

ERRORS

The **getsockopt**() and **setsockopt**() system calls succeed unless:

[EBADF] The argument *s* is not a valid descriptor.

[ENOTSOCK] The argument *s* is a file, not a socket.

[ENOPROTOOPT] The option is unknown at the level indicated.

[EFAULT] The address pointed to by *optval* is not in a valid part of the process address space. For **getsockopt**(), this error may also be returned if *optlen* is not in a valid part of the process address space.

[EINVAL] Installing an accept_filter(9) on a non-listening socket was attempted.

[ENOMEM] A memory allocation failed that was required to service the request.

The **setsockopt**() system call may also return the following error:

[ENOBUFS] Insufficient resources were available in the system to perform the operation.

SEE ALSO

ioctl(2), listen(2), recvmsg(2), socket(2), getprotoent(3), mac(3), sysctl(3), ip(4), ip6(4), sctp(4), tcp(4), protocols(5), sysctl(8), accept_filter(9), bintime(9)

HISTORY

The **getsockopt**() and **setsockopt**() system calls appeared in 4.2BSD.

BUGS

Several of the socket options should be handled at lower levels of the system.