NAME

getsockopt, setsockopt – get and set options on sockets

SYNOPSIS

cc [*flag* ...] *file* ... **–lsocket –lnsl** [*library* ...]

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

int getsockopt(int *s*, **int** *level*, **int** *optname*, **char** **optval*, **int** **optlen*);
int setsockopt(int *s*, **int** *level*, **int** *optname*, **const char** **optval*, **int** *optlen*);

MT-LEVEL

Safe

DESCRIPTION

getsockopt( ) and **setsockopt( )** manipulate 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, *level* is the protocol number of the protocol that controls the option. For example, to indicate that an option is to be interpreted by the TCP protocol, *level* is set to the TCP protocol number (see **getprotobyname**(3N)).

The parameters *optval* and *optlen* are used to access option values for **setsockopt( )**. For **getsockopt( ),** they identify a buffer in which the value(s) for the requested option(s) are to be returned. For **getsockopt( )**, *optlen* is a value-result parameter, initially containing the size of the buffer pointed to by *optval*, and modified on return to indicate the actual size of the value returned. Use a 0 *optval* if no option value is to be supplied or returned.

optname and any specified options are passed uninterpreted to the appropriate protocol module for interpretation. The include file **<sys/socket.h>** contains definitions for the socket-level options described below. Options at other protocol levels vary in format and name.

Most socket-level options take an int for *optval*. For **setsockopt( )**, the *optval* parameter should be non-zero to enable a boolean option, or zero if the option is to be disabled. **SO_LINGER** uses a **struct linger** parameter that specifies the desired state of the option and the linger interval (see below). **struct linger** is defined in **<sys/socket.h>**. **struct linger** contains the following members:

l_onoff	on = 1/off = 0
l_linger	linger time, in seconds

The following options are recognized at the socket level. Except as noted, each may be examined with **getsockopt( )** and set with **setsockopt( )**.

SO_DEBUG	enable/disable recording of debugging information
SO_REUSEADDR	enable/disable local address reuse
SO_KEEPALIVE	enable/disable keep connections alive
SO_DONTROUTE	enable/disable routing bypass for outgoing messages
SO_LINGER	linger on close if data is present

SO_BROADCAST	enable/disable permission to transmit broadcast messages
SO_OOBINLINE	enable/disable reception of out-of-band data in band
SO_SNDBUF	set buffer size for output
SO_RCVBUF	set buffer size for input
SO_TYPE	get the type of the socket (get only)
SO_ERROR	get and clear error on the socket (get only)

SO_DEBUG enables debugging in the underlying protocol modules. **SO_REUSEADDR** indicates that the rules used in validating addresses supplied in a **bind**(3N) call should allow reuse of local addresses. **SO_KEEPALIVE** enables the periodic transmission of messages on a connected socket. If the connected party fails to respond to these messages, the connection is considered broken and processes using the socket are notified using a **SIGPIPE** signal. **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 a 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( )** 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 the **setsockopt( )** call when **SO_LINGER** is requested). If **SO_LINGER** is disabled and a **close( )** 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. 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( )** or **read( )** calls without the **MSG_OOB** flag.

SO_SNDBUF and **SO_RCVBUF** are options that 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 Internet protocols place an absolute limit of 64 Kbytes on these values for **UDP** and **TCP** sockets.

Finally, **SO_TYPE** and **SO_ERROR** are options used only with **getsockopt( )**. **SO_TYPE** returns the type of the socket (for example, **SOCK_STREAM**). It is useful for servers that inherit sockets on startup. **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.

RETURN VALUES

If successful, **getsockopt( )** returns **0**; otherwise, it returns –**1** and sets **errno** to indicate the error.

ERRORS

The call succeeds unless:

EBADF The argument *s* is not a valid file descriptor.

ENOMEM There was insufficient memory available for the operation to complete.

ENOPROTOOPT The option is unknown at the level indicated.

ENOSR There were insufficient STREAMS resources available for the operation to complete.

ENOTSOCK The argument *s* is not a socket.

SEE ALSO

close(2), **ioctl**(2), **bind**(3N), **getprotobyname**(3N), **socket**(3N)