Socket Options

Harshad B. Prajapati
Associate Professor
Information Technology Department,
Dharmsinh Desai University, Nadiad

Socket Options

- They are used to enable/disable features to get more control on behavior of sockets.
- They are used to change default value of some feature
- Two functions
  - getsockopt
  - setsockopt

getsockopt and setsockopt function

```
#include <sys/socket.h>
int getsockopt(int sockfd, int level, int optname, void *optval,
    socklent_t *optlen);
int setsockopt(int sockfd, int level , int optname, const void
    *optval, socklent_t  optlen);
```

- sockfd: open socket descriptor
- level: code in the system to interpret the option(generic, IPv4, IPv6, TCP)
  - level includes general socket option (SOL_SOCKET) and protocol-specific option (IP, TCP, etc).

getsockopt and setsockopt function

- optval: pointer to a variable from which the new value of option is fetched by setsockopt, or into which the current value of the option is stored by getsockopt.
  - Option value can be of different types : int, in_addr, timeval, …
  - that is the reason we use the void pointer.
- optlen: the size of the option variable.

getsockopt and setsockopt function

- Two types of options
  - Binary option
    - Used to enable or disable certain feature (flag option).
    - optval is integer type.
      - In getsockopt, The returned optval 0 means option is disabled and 1 means it is enabled.
      - In setsockopt, optval 0 is used to disable the option and nonzero is used to enable the option.
  - Value option
    - Uses specific values.
    - Used to pass/fetch values, structures, etc.

Socket options

General socket options (level = SOL_SOCKET)

- Optnames:
  - SO_BROADCAST(int): permit sending broadcast datagram
  - SO_ERROR: can only be got (not set), reset the error
  - SO_KEEPALIVE: for TCP only, automatic send keepalive message when inactive for 2 hours (can be modified).
  - SO_LINGER: for TCP only, determines the behavior when a close is called.
  - SO_RCVBUF, SO_SNDBUF: send and receive buffer size.

Socket options

IP options:
- Allows packets sent through a socket to have certain behavior,
- Level = IPPROTO_IP
- E.g., manipulating IP header fields

Socket options

TCP options:
- Level = IPPROTO_TCP
- Optnames:
  - TCP_KEEPALIVE: set the time
  - TCP_NODELAY: wait to ack or not (enable)

Inheritance of Socket options

- The following socket options are inherited by a connected TCP socket from the listening socket.
  - SO_KEEPALIVE
  - SO_LINGER
  - SO_RCVBUF
  - SO_SNDBUF
- When to set these options?
  - The connected socket is returned to a server by accept until the three-way handshake is completed by the TCP layer.
  - Therefore if something should happen before 3WH completes ?
  - To set these option(s), set them for the listening socket.

Generic Socket Options

- These options are protocol independent. However, certain options apply to only certain types of sockets (E.g., SO_BROADCAST).
- They are handled by a protocol independent code within the kernel. (Not by one particular protocol module)

Generic Socket Options (SO_BROADCAST)

- It enables or disables the ability of the process to send broadcast message on broadcast links.
  - Can work only with datagram socket.
  - Moreover only on network that supports the concept of broadcast.
    - E.g Ethernet, token ring..)
- Example:

int broadcastFlag=1;

setsockopt(sd,SOL_SOCKET,SO_BROADCAST,&broadcastFlag,sizeof(broadcastFlag));

- If the destination address is a broadcast address, and this socket option is not set**,** EACCES is returned.

Generic Socket Options (SO_ERROR)

- when error occurs on a socket, the protocol module in a Berkeley-derived kernel sets one of standard UNIX Exxx values to a variable named so_error for that socket. Not errno, it is error of a process.
- It is called *pending error* for the socket.
- The process can be immediately notified of the error in one of two ways
  - If the process is blocked in a call to select on the socket
  - If the process is using signal-driven I/O, the SIGIO signal is generated for either the process or the process group.
- The process can then obtain the value of so_error by fetching the SO_ERROR socket option.

Generic Socket Options (SO_ERROR)

- If so_error is nonzero when the process calls read
  - And there is no data to return, read returns -1 with errno set to the value of so_error. The value of so_error is then reset to 0.
  - If there is data queued for the socket, that data is returned by read instead of error condition.
- If so_error is nonzero when the process calls write, -1 is returned with errno set to the value of so_error. The so_error is reset to 0.

Generic Socket Options (SO_KEEPALIVE)

- It is used to detect peer host crash.
- When the keepalive option is set for a TCP socket.
  - And there is no data exchange for 2hours,
  - then TCP automatically sends a keep-alive probe to the peer.
- Possible peer responses of keep-alive probe
  - ACK(everything OK)
  - RST(peer crashed and rebooted):ECONNRESET
  - no response to keep alive probe.

Generic Socket Options (SO_KEEPALIVE)

- Peer response-1: ACK
  - The peer TCP responds with the expected ACK.
  - Everything OK, and application is not notified.
  - TCP will send another probe following another 2 hours of inactivity.

Generic Socket Options (SO_KEEPALIVE)

- Peer response-2: RST
  - The peer TCP responds with RST.
  - It indicates that peer host has crashed and rebooted.
  - Socket's pending error is set to ECONNRESET.
    - When do we get this error? See SO_ERROR.
  - And the socket is closed.

Generic Socket Options (SO_KEEPALIVE)

- Peer response-3: No response to keep alive probe.
  - Berkely derived TCPs send eight additional probes, 75 seconds apart, trying to elicit response.
  - TCP will give up if no response within 11 minutes and 15 seconds after sending the first probe.
  - If there is no response at all to TCP's keepalive probes,
    - Socket's pending error is set to ETIMEDOUT and the socket is closed.

      Or
    - If the socket receives an ICMP error in response to one of keepalive probes, the corresponding error is returned instead. (EHOSTUNREACH).

Generic Socket Options (SO_KEEPALIVE)

- Can we change inactivity duration (i.e., rather than 2 hours, can we specify some other period)?
  - Most kernels maintain these parameters on a per-kernel basis.
  - So, changing the inactivity period from 2 hours to 15 minutes will affect all sockets on the host.

Generic Socket Options (SO_KEEPALIVE)

- Use of SO_KEEPALIVE to detect peer host crash.
- If the peer process crashes?
  - Its TCP will send a FIN across the connection, which we can easily detect with select (through I/O multiplexing).
- If there is no response to any keep-alive probes?
  - We are not guaranteed that peer host has crashed.
  - It could be possible that some intermediate router failed for 15 minutes (for example,), and our probe sending period gets completely overlapped by this 15 minutes period.

Generic Socket Options (SO_KEEPALIVE)

- Practical usage
- This option is normally used by servers.
- Servers use the option as they spend most of their time for waiting for input across the TCP connection.
- If the client host crashes, the server process will never know about it.
- And the server will continually wait for input that can never arrive.
- This is called half-open connection. The keep-alive option will detect these half-open connections and terminate them.

Generic Socket Options (SO_KEEPALIVE)

Scenario	Peer process crashes	Peer host crashes	Peer host is unreachable
Our TCP is actively sending data	•Peer TCP sends FIN (detect using select) •If our TCP sends another segment, peer TCP responds with RST. •If our TCP sends yet another segment, our TCP sends us SIGPIPE.	Our TCP will time out and our socket's pending error is set to ETIMEDOUT	Our TCP will time out and our socket's pending error is set to EHOSTUNREACH
Our TCP is actively receiving data	Peer TCP will send FIN, which we will read as end-of-file	We will stop receiving data	We will stop receiving data
Connection is idle, Keepalive is set.	Peer TCP sends FIN (detect using select)	9 probes sent Error: ETIMEDOUT	9 probes sent Error: EHOSTUNREACH
Connection is idle, Keepalive not set	Peer TCP sends FIN (detect using select)	(Nothing)	(Nothing)

Generic Socket Options (SO_LINGER)

- Linger means gradually dying.
- Using this option, we can specify how the close function operates for a connection oriented protocol.
- By default, close returns immediately, but if there is any data still remaining in the socket send buffer, the system will try to deliver the data to the peer.
- We can change this default behavior by passing properly initialized struct linger structure to setsockopt function.

Generic Socket Options (SO_LINGER)

```
struct linger{
        int l_onoff; /* 0 = off, nonzero = on */
        int l_linger; /*linger time : second*/
};
```

- l_onoff = 0 : turn off the option , l_linger is ignored.
  - We get default behavior.
- l_onoff = nonzero and l_linger is 0:
  - TCP aborts the connection when the close is called.
  - TCP discard any remaining data in the socket send buffer and sends RST to the peer, not the normal four packet connection termination sequence.
  - Moreover, TCP's TIME_WAIT state is avoided. There is possibility of another incarnation of this connection gets created within 2MSL seconds (old duplicates from earlier connection arrive at new connection)

Generic Socket Options (SO_LINGER)

```
struct linger{
        int l_onoff; /* 0 = off, nonzero = on */
        int l_linger; /*linger time : second*/
        };
```

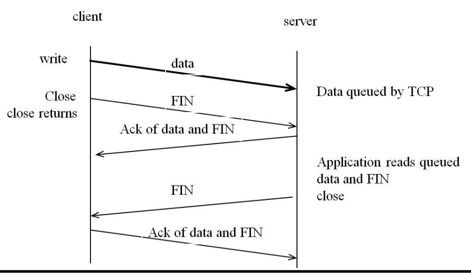
- l_onoff = nonzero and l_linger is nonzero :
  - Kernel will linger until socket is closed.
  - If there is any data still remaining in socket send buffer, the process is put to sleep until
    - Either all the data is sent and acknowledge by peer TCP
    - or until the linger time expires.
  - If socket has been set nonblocking, it will not wait for the close to complete, even if linger time is nonzero.

Generic Socket Options (SO_LINGER)

- The application should check the return value from close while using linger socket option.
- If the linger time expires before the remaining data is sent and acknowledged, close returns EWOULDBLOCK and any remaining data in the send buffer is discarded.

Generic Socket Options (SO_LINGER)

- Default operation of close. It returns immediately

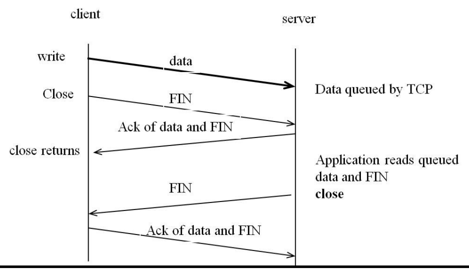


Generic Socket Options (SO_LINGER)

- In default behavior, there exists two problems.
  - It is possible that the client's close can return before the server reads the remaining data in its socket receive buffer.
  - It is possible that the server host crashes before server application reads this remaining data, and the client application will never know this.

Generic Socket Options (SO_LINGER)

- By setting so_linger socket option, the client can make sure that its sent data and FIN have been acknowledged by the peer TCP. But still, the problem 2 exists.



Generic Socket Options (SO_LINGER)

- Successful return from close, with SO_LINGER option set, only tells us that the data we sent and our FIN have been acknowledged by the peer TCP.
- This does not tell us whether the peer application has read the data.

Generic Socket Options (SO_LINGER)

- How to make sure that the peer application has read our data?
  - Use of shutdown
  - Use of application level acknowledgement of data.
- shutdown

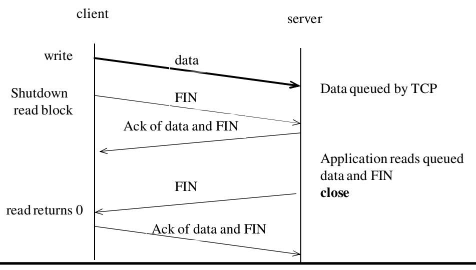
```
#include <sys/socket.h>
int shutdown(int s, int how);
```

(On success, zero is returned. On error, -1 is returned, and errno is set appropriately.)

  - If how = SHUT_RD, further receiving will not be allowed.
  - If how = SHUT_WR, further sending will not be allowed.
  - If how = SHUT_RDWR, further receiving and sending will not be allowed.

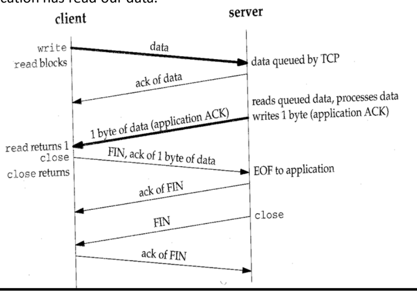
Generic Socket Options (SO_LINGER)

- **Use of shutdown** to know the peer application has read our data. Call shutdown with how=SHUT_WR instead of close and then wait for the peer to close its end of the connection.



Generic Socket Options (SO_LINGER)

- **Use of application level acknowledgement of data** to know the peer application has read our data.



Generic Socket Options (SO_LINGER)

- **Use of application level acknowledgement of data** to know the peer application has read our data.
- Use ack of 1 byte.
- Client code

char ack;

write(sockfd, data, nbytes); /* data from client to server */

N=read(sockfd, &ack, 1); /* wait for application-level ack */

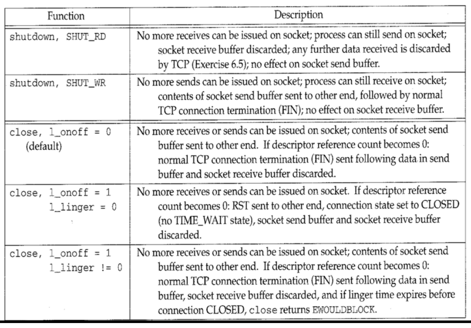
- Server

nbytes=read(sockfd, buff, sizeof(buff)); /*data from client */

/* server verifies it received the correct amount of data from the client */

write(sockfd, “ ”, 1); /* server’s ACK back to client */

Generic Socket Options (SO_LINGER)-Summary



Function	Description
`shutdown, SHUT_RD`	No more receives can be issued on socket; process can still send on socket; socket receive buffer discarded; any further data received is discarded by TCP (Exercise 6.5); no effect on socket send buffer.
`shutdown, SHUT_WR`	No more sends can be issued on socket; process can still receive on socket; contents of socket send buffer sent to other end, followed by normal TCP connection termination (FIN); no effect on socket receive buffer.
`close, l_onoff = 0` (default)	No more receives or sends can be issued on socket; contents of socket send buffer sent to other end. If descriptor reference count becomes 0: normal TCP connection termination (FIN) sent following data in send buffer and socket receive buffer discarded.
`close, l_onoff = 1` `l_linger = 0`	No more receives or sends can be issued on socket. If descriptor reference count becomes 0: RST sent to other end, connection state set to CLOSED (no TIME_WAIT state), socket send buffer and socket receive buffer discarded.
`close, l_onoff = 1` `l_linger != 0`	No more receives or sends can be issued on socket; contents of socket send buffer sent to other end. If descriptor reference count becomes 0: normal TCP connection termination (FIN) sent following data in send buffer, socket receive buffer discarded, and if linger time expires before connection CLOSED, `close` returns `EWOULDBLOCK`.

Generic Socket Options (SO_RCVBUF and SO_SNDBUF)

- The receive buffers are used by TCP and UDP to hold received data until it is read by the application.
- TCP
  - The available room in socket receive buffer is the window that the TCP advertizes to the other end.
  - The TCP socket receive buffer can not overflow because peer is not allowed to send data beyond the advertized window. I.e. TCP’s flow control.
  - the receiving TCP discards the data, if the peer ignores the advertized window and sends data beyond the window.
  - Default TCP send and receive buffer size : 4096bytes
  - Newer systems use 8192-61440 bytes

Generic Socket Options (SO_RCVBUF and SO_SNDBUF)

- UDP
  - UDP has no flow control.
  - When a datagram arrives that will not fit in the socket receive buffer, that datagram is discarded.
  - Default UDP send buffer size : 9000bytes, Default UDP receive buffer size 40000 bytes

Generic Socket Options (SO_RCVBUF and SO_SNDBUF)

- The ordering of function calls is important when setting the size of the TCP socket receive buffer.
- TCP's window scale option is exchanged with the peer on the SYN segments when the connection is established.
  - For Clients: SO_RCVBUF socket option must be set before calling connect.
  - For Servers: SO_RCVBUF socket option must be set for the listening socket before calling listen.
- TCP socket buffer size should be at least three times the MSSs for the connection.
- The TCP socket buffer sizes should also be an even multiple of the MSS for the connection.

TCP socket options (TCP_KEEPALIVE)

- TCP_KEEPALIVE: Specify the idle time in seconds.
- TCP_NODELAY: disable Nagle algorithm, to reduce the number of small packets

TCP socket options (TCP_NODELAY)

- This option disables TCP's Nagle algorithm. (By default this algorithm is enabled).
- The purpose of the Nagle algorithm is to reduce the number of small packets outstanding at any time on a WAN.
- What is a small packet?
  - Any packet smaller than MSS.
- The Nagle algorithm:
  - If a connection has outstanding data (data sent but yet not been acknowledged), then no small packets will be sent on the connection until the existing data is acknowledged.

TCP socket options (TCP_NODELAY)

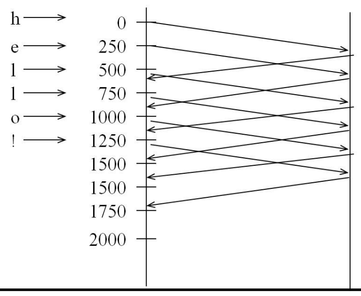
- The two common applications that generate small packets are
  - Rlogin and
  - Telnet clients.
- They send each keystroke as a separate packet.
- We do not notice the Nagle algorithm with these clients on a fast LAN.
  - Because the time required for a small packet to be acknowledged is typically a few milliseconds, far less than the time between two successive characters that human being type.
- But on a WAN (where it takes a second for a small packet to be acknowledged), we can notice a delay in the character echoing.

TCP socket options (TCP_NODELAY)

- Consider a situation
  - We type the six-character string "hello!" to either the Rlogin or Telnet client, with exactly 250 ms between each character.
  - Assume, the RTT to the server is 600 ms. And the server immediately sends back the echo of the character.
  - We assume that the ACK of the client's character is sent back to the client along with the character echo.
  - And we ignore (in displaying it on a diagram) the ACKs that the client sends for the server's echo.
- Lets see this scenario with (i) Nagle algorithm disabled and (ii) Nagle algorithm enabled.

TCP socket options (TCP_NODELAY)

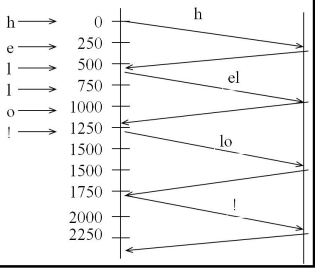
- Nagle algorithm disabled. I.e., TCP_NODELAY is enabled.



TCP socket options (TCP_NODELAY)

Nagle algorithm enabled. I.e., TCP_NODELAY is disabled

- The first character is sent as a packet.
- But next two characters are not sent, as the connection has a small packet outstanding.
- At time 600, when the ACK of the first packet is received, along with the echo of the first character, these two characters are sent.



TCP socket options (TCP_NODELAY)

- The Nagle algorithm often interacts with another TCP algorithm: the Delayed ACK algorithm.
- Delayed ACK algorithm
  - When TCP receives data, it does not send ACK immediately.
  - It wait some small amount of time (50-200 ms) with the hope that in this small amount of time there will be data to send back to the peer.
  - This is done so that the ACK can piggyback with the data and can have saving of one TCP segment.
  - This is normally the case with the Rlogin and Telnet clients. (servers typically echo each character sent by the client)

TCP socket options (TCP_NODELAY)

- Clients whose servers do not generate traffic in the reverse direction?
  - These clients can detect noticeable delays. Because the client TCP will not send any data to the server until the server's delayed ACK timer expires. As, after expiry of delayed ACK timer, client TCP would receive ACK of previous data
  - Clients can disable Nagle algorithm using TCP_NODELAY option.

TCP socket options (TCP_NODELAY)

- Client that sends a single logical request to its server in small packets
  - Such client interacts badly with the Nagle algorithm and TCP's delayed ACK .
- Example.
  - A client sends 400 bytes request (4 bytes request type followed by 396 bytes of request data) to the server using two write calls.
  - The second write will not be sent by the client TCP until the server TCP acknowledges the 4-byte write.
  - Also, the server application cannot operate on 4 bytes of data until it receives the remaining 396 bytes of the data. The server TCP will delay the ACK of the 4 bytes of data.

TCP socket options (TCP_NODELAY)

Client that sends a single logical request to its server in small packets.

- Possible Solutions:
  - Use writev (vector write) instead of two write calls.
  - Copy the 4 byte of data and the 396 of data into a single buffer and call write once for this buffer.
  - Set TCP_NODELAY socket option and continue to call write two time. (It is least desirable solution for the situation.)

Socket Options

- Read Chapter 7 of Stevens, Vol1.