Socket Option

Introduction

- Ways to get and set the socket option that affect a socket
  - ***getsockopt , setsockopt***

  function=>IPv4 and IPv6 multicasting options

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 interprete the option(generic, IPv4, IPv6, TCP)
- ***optval*** => pointer to a variable from which the new value of option is fetched by ***getsockopt,*** or into which the current value of the option is stored by ***setsockopt.***
- ***optlen*** => the size of the option variable.

socket state

- We must set that option for the listening socket => because connected socket is not returned to a server by ***accept*** until the three way handshake is completed by the TCP layer.

Generic socket option

- **SO_BROADCAST** =>enable or disable the ability of the process to send broadcast message.(**only datagram socket** : Ethernet, token ring..). You cannot broadcast on a point-to-point link or any connection-based transport protocol such as SCTP or TCP.

- **SO_DEBUG** =>kernel keep track of detailed information about **all packets sent or received by TCP** (only supported by TCP)

- **SO_DONTROUTE**=>outgoing packets are to bypass the normal routing mechanisms of the underlying protocol. The destination must be on a directly-connected network, and messages are directed to the appropriate network interface according to the destination address

- **SO_ERROR**=>when error occurs on a socket, the protocol module in a Berkeley-derived kernel sets a variable named **so_error for that socket. Process can obtain the value of so_error by fetching the SO_ERROR socket option**

SO_KEEPALIVE

- **SO_KEEPALIVE**=>wait 2hours, and then TCP automatically sends a ***keepalive probe*** to the peer.
  - Peer response
    - ACK(everything OK)
    - RST(peer crashed and rebooted):ECONNRESET
    - no response:ETIMEOUT =>socket closed
  - example: Rlogin, Telnet…

SO_LINGER

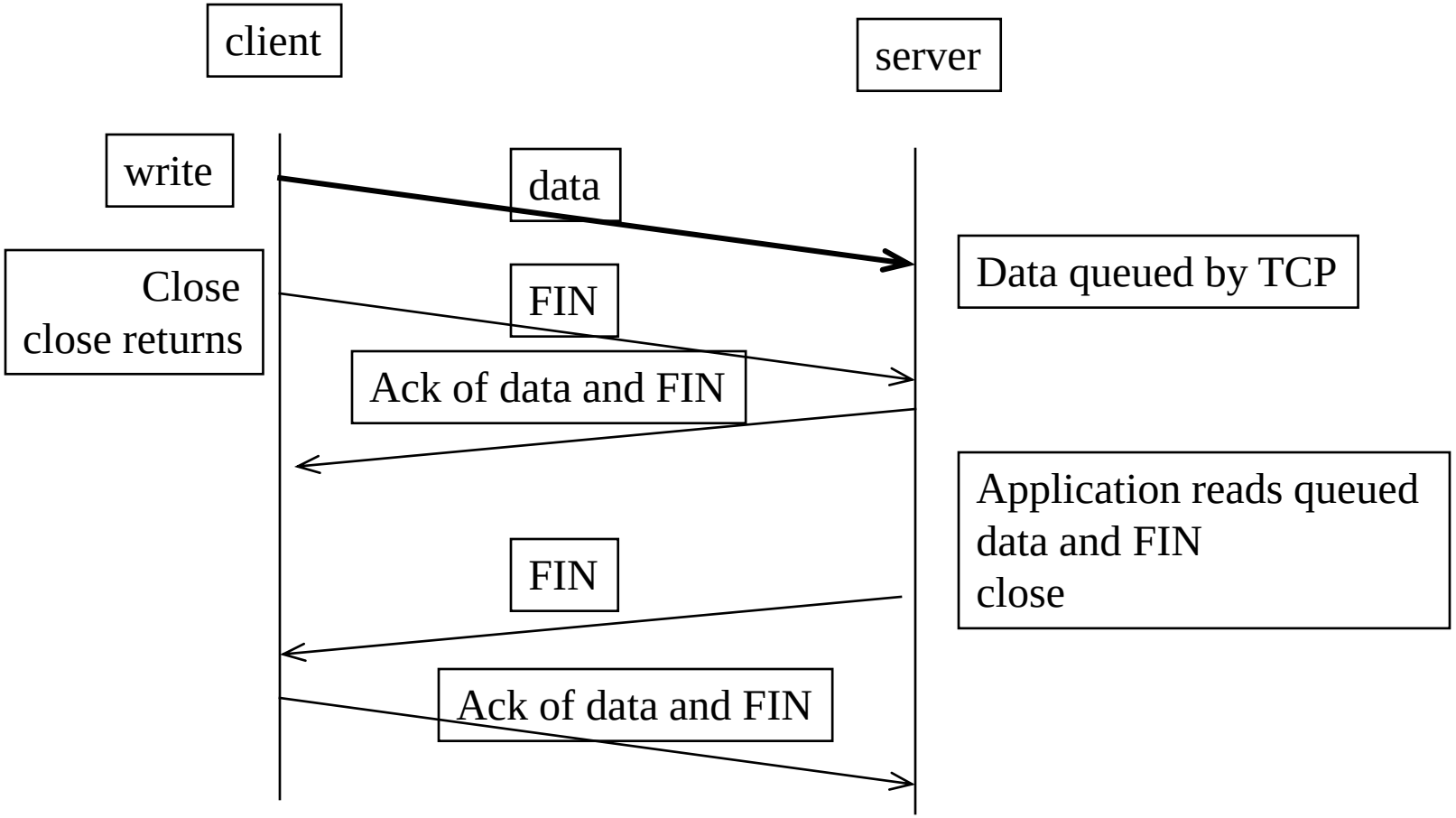
- **SO_LINGER** =>specify how the ***close*** function operates for a connection-oriented protocol(default:close returns immediately)
  - struct linger{

    int l_onoff; /* 0 = off, nonzero = on */

    int l_linger; /*linger time : second*/

    };
- ***l_onoff*** = 0 : turn off , ***l_linger*** is ignored
- ***l_onoff*** = nonzero and ***l_linger*** is 0:TCP abort the connection, discard any remaining data in send buffer.
- ***l_onoff*** = nonzero and ***l_linger*** is nonzero : process wait until remained data sending, or until linger time expired. If socket has been set nonblocking it will not wait for the ***close*** to complete, even if linger time is nonzero.

SO_LINGER



Default operation of close:it returns immediately

- A way to know that the peer application has read the data
  - use an application-level ack or application ACK
  - **client**

```
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
```

SO_RCVBUF , SO_SNDBUF

- let us change the default send-buffer, receive-buffer size.
  - Default TCP send and receive buffer size :
    - 4096bytes
    - 8192-61440 bytes
  - Default UDP buffer size : 9000bytes, 40000 bytes
- SO_RCVBUF option must be setting before connection established.
- TCP socket buffer size should be at least three times the MSSs

SO_RCVLOWAT , SO_SNDLOWAT

- Every socket has a receive low-water mark and send low-water mark.(used by select function)
- **Receive low-water mark:**
  - the amount of data that must be in the socket receive buffer for select to return “readable”.
  - Default receive low-water mark : 1 for TCP and UDP
- **Send low-water mark:**
  - the amount of available space that must exist in the socket send buffer for select to return “writable”
  - Default send low-water mark : 2048 for TCP
  - UDP send buffer never change because dose not keep a copy of send datagram.

SO_RCVTIMEO, SO_SNDTIMEO

- allow us to place a timeout on socket receives and sends.
- Default disabled

SO_REUSEADDR, SO_REUSEPORT

- Allow a listening server to start and bind its well known port even if previously established connection exist that use this port as their local port.
- Allow multiple instance of the same server to be started on the same port, as long as each instance binds a different local IP address.
- Allow a single process to bind the same port to multiple sockets, as long as each bind specifies a different local IP address.
- Allow completely duplicate bindings : multicasting

SO_TYPE

- Return the socket type.
- Returned value is such as SOCK_STREAM, SOCK_DGRAM...

SO_USELOOPBACK

- This option applies only to sockets in the routing domain(AF_ROUTE).
- The socket receives a copy of everything sent on the socket.

IPv4 socket option

- ***Level*** => IPPROTO_IP
- IP_HDRINCL => If this option is set for a raw IP socket, we must build our IP header for all the datagrams that we send on the raw socket.(chapter 26)

IPv4 socket option

- IP_OPTIONS=>allows us to set IP option in IPv4 header.(chapter 24)
- IP_RECVDSTADDR=>This socket option causes the destination IP address of a received UDP datagram to be returned as ancillary data by ***recvmsg***.(chapter20)

IP_RECVIF

- Cause the index of the interface on which a UDP datagram is received to be returned as ancillary data by recvmsg.(chapter20)

IP_TOS

- lets us set the type-of-service(TOS) field in IP header for a TCP or UDP socket.
- If we call getsockopt for this option, the current value that would be placed into the TOS(type of service) field in the IP header is returned.(figure A.1)

IP_TTL

- We can set and fetch the default TTL(time to live field, figure A.1).

TCP socket option

- There are five socket option for TCP, but three are new with Posix.1g and not widely supported.
- Specify the level as IPPROTO_TCP.

TCP_KEEPALIVE

- This is new with Posix.1g
- It specifies the idle time in second for the connection before TCP starts sending keepalive probe.
- Default 2hours
- this option is effective only when the SO_KEEPALIVE socket option enabled.

TCP_MAXRT

- This is new with Posix.1g.
- It specifies the amount of time in seconds before a connection is broken once TCP starts retransmitting data.
  - 0 : use default
  - -1:retransmit forever
  - positive value:rounded up to next transmission time

TCP_MAXSEG

- This allows us to fetch or set the maximum segment size(MSS) for TCP connection.

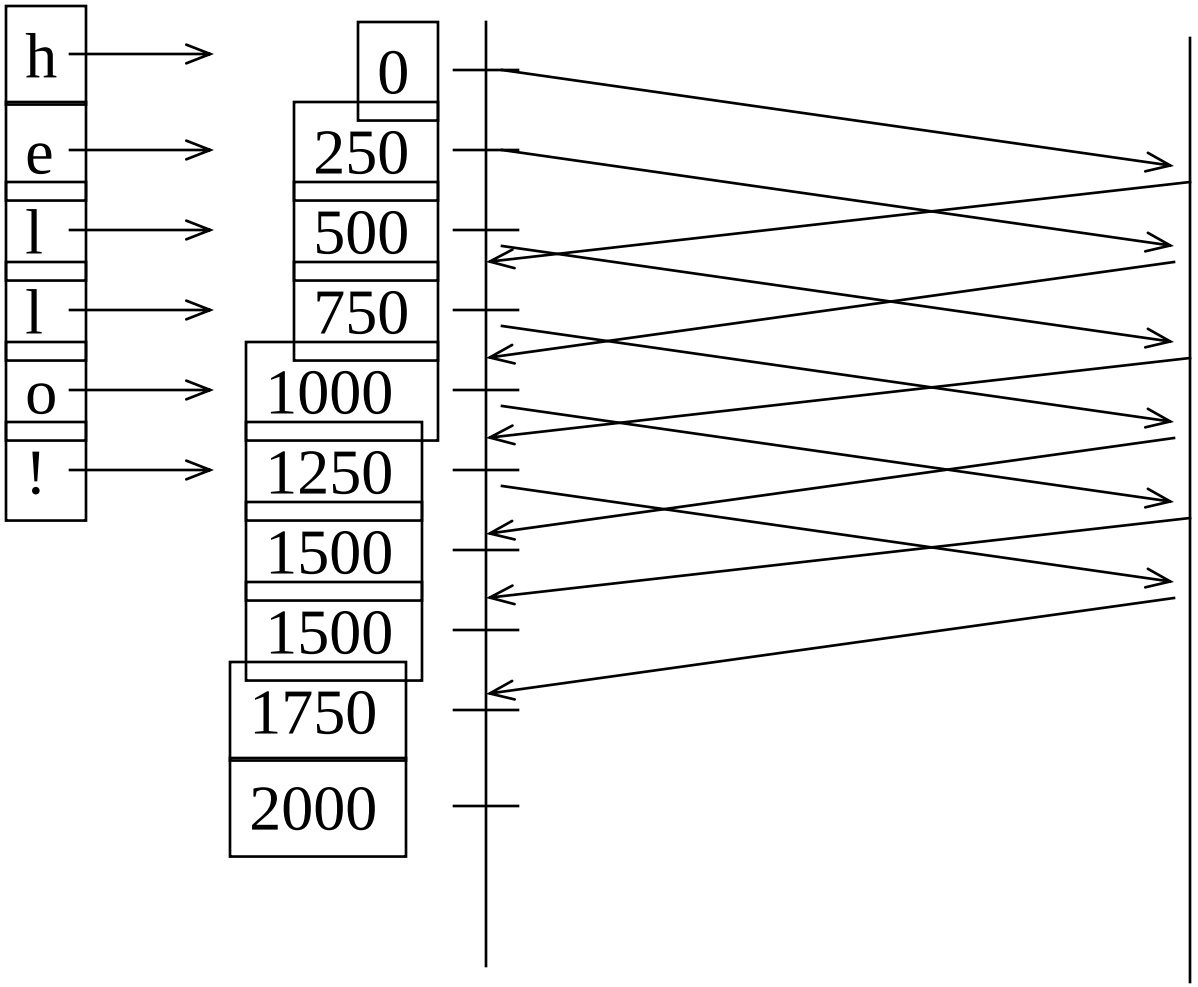
TCP_NODELAY

- This option disables TCP's ***Nagle algorithm***.
  (default this algorithm enabled)
- purpose of the ***Nagle algorithm***.
  ==>prevent a connection from having multiple small packets outstanding at any time.
- Small packet => any packet smaller than MSS.

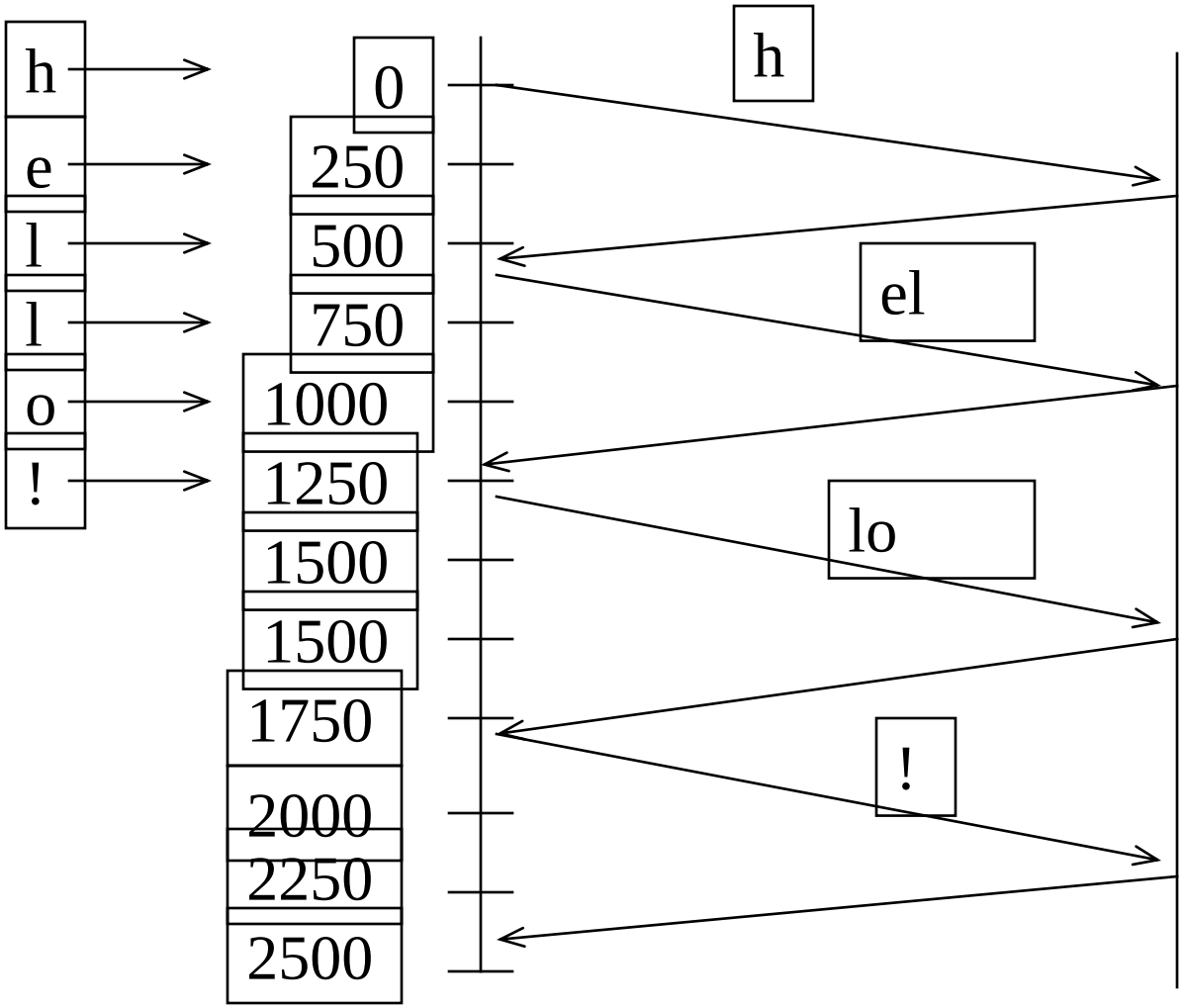
Nagle algorithm

- Default enabled.
- Reduce the number of small packet on the WAN.
- If given connection has outstanding data , then no small packet data will be sent on connection until the existing data is acknowledged.

Nagle algorithm disabled



Nagle algorithm enabled



Example: Self Practice

sockopt/checkopts.c

```
 1 #include     "unp.h"
 2 #include     <netinet/tcp.h>      /* for TCP_xxx defines */

 3 union val {
 4   int                 i_val;
 5   long                l_val;
 6   char                c_val[10];
 7   struct linger       linger_val;
 8   struct timeval      timeval_val;
 9 } val;

10 static char *sock_str_flag(union val *, int);
11 static char *sock_str_int(union val *, int);
12 static char *sock_str_linger(union val *, int);
13 static char *sock_str_timeval(union val *, int);

14 struct sock_opts {
15   char      *opt_str;
16   int        opt_level;
17   int        opt_name;
18   char    *(*opt_val_str)(union val *, int);
19 } sock_opts[] = {
20     "SO_BROADCAST",      SOL_SOCKET,  SO_BROADCAST,    sock_str_flag,
21     "SO_DEBUG",          SOL_SOCKET,  SO_DEBUG,        sock_str_flag,
22     "SO_DONTROUTE",      SOL_SOCKET,  SO_DONTROUTE,    sock_str_flag,
23     "SO_ERROR",          SOL_SOCKET,  SO_ERROR,        sock_str_int,
24     "SO_KEEPALIVE",      SOL_SOCKET,  SO_KEEPALIVE,    sock_str_flag,
25     "SO_LINGER",         SOL_SOCKET,  SO_LINGER,       sock_str_linger,
26     "SO_OOBINLINE",      SOL_SOCKET,  SO_OOBINLINE,    sock_str_flag,
27     "SO_RCVBUF",         SOL_SOCKET,  SO_RCVBUF,       sock_str_int,
28     "SO_SNDBUF",         SOL_SOCKET,  SO_SNDBUF,       sock_str_int,
29     "SO_RCVLOWAT",       SOL_SOCKET,  SO_RCVLOWAT,     sock_str_int,
30     "SO_SNDLOWAT",       SOL_SOCKET,  SO_SNDLOWAT,     sock_str_int,
31     "SO_RCVTIMEO",       SOL_SOCKET,  SO_RCVTIMEO,     sock_str_timeval,
32     "SO_SNDTIMEO",       SOL_SOCKET,  SO_SNDTIMEO,     sock_str_timeval,
33     "SO_REUSEADDR",      SOL_SOCKET,  SO_REUSEADDR,    sock_str_flag,
34 #ifdef  SO_REUSEPORT
35     "SO_REUSEPORT",      SOL_SOCKET,  SO_REUSEPORT,    sock_str_flag,
36 #else
37     "SO_REUSEPORT",      0,           0,               NULL,
38 #endif
39     "SO_TYPE",           SOL_SOCKET,  SO_TYPE,         sock_str_int,
40     "SO_USELOOPBACK",    SOL_SOCKET,  SO_USELOOPBACK,  sock_str_flag,
41     "IP_TOS",            IPPROTO_IP,  IP_TOS,          sock_str_int,
42     "IP_TTL",            IPPROTO_IP,  IP_TTL,          sock_str_int,
43     "TCP_MAXSEG",        IPPROTO_TCP, TCP_MAXSEG,      sock_str_int,
44     "TCP_NODELAY",       IPPROTO_TCP, TCP_NODELAY,     sock_str_flag,
45     NULL,                0,           0,               NULL
46 };
```

sockopt/checkopts.c

Figure 7.2 Declarations for our program to check the socket options.

sockopt/checkopts.c

```
47 int
48 main(int argc, char **argv)
49 {
50     int     fd, len;
51     struct sock_opts *ptr;

52     fd = Socket(AF_INET, SOCK_STREAM, 0);

53     for (ptr = sock_opts; ptr->opt_str != NULL; ptr++) {
54         printf("%s: ", ptr->opt_str);
55         if (ptr->opt_val_str == NULL)
56             printf("(undefined)\n");
57         else {
58             len = sizeof(val);
59             if (getsockopt(fd, ptr->opt_level, ptr->opt_name,
60                            &val, &len) == -1) {
61                 err_ret("getsockopt error");
62             } else {
63                 printf("default = %s\n", (*ptr->opt_val_str) (&val, len));
64             }
65         }
66     }
67     exit(0);
68 }
```

sockopt/checkopts.c

Figure 7.3 `main` function to check all socket options.