Socket Options

Socket options

- Various attributes that are used to determine the behavior of sockets
- Setting options tells the OS/Protocol stack the desired behavior
- Support for generic options (apply to all sockets) and protocol specific options

Socket options

- Many socket options are Boolean flags indicating whether some feature is enabled (1) or disabled (0)
- Other options are associated with more complex types, e.g., in_addr, sockaddr, etc
- Some options are *read-only* and the value can't be set, e.g., SO_ERROR

Socket options (2)

- Many options available and depend on the OS
- Let's look at some sampling

Generic options

- Protocol independent options
- Handled by the generic socket system code in the kernel
- Some generic options are supported only by specific types of sockets
  - E.g., SO_BROADCAST applies only to UDP sockets

Managing socket options – 3 ways

- `getsockopt()` **and** `setsockopt()`
- `fcntl()`
- `ioctl()`

getsockopt()and setsockopt()

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

int getsockopt(
   int sockfd,
   int level,     /* SOL_SOCKET, IPPROT_IP, IPPROTO_TCP etc */
   int optname,   /* option name */
   void *optval, /* option value */
   socklen_t *optlen); /* data length of option */

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

getsockopt()

- *level* specifies whether the option is a general option or a protocol specific option
  - What level of code should interpret the option
  - E.g., SOL_SOCKET request applies to the socket layer
  - IPv4 (IPPROTO_IP)
  - IPv6 (IPPROTO_IPV6)
  - ICMP (IPPROTO_ICMP)
  - TCP (IPPROTO_TCP)
  - …

setsockopt()

- *optval* is pointer to where to get the current value
- *optlen* specifies size of the optval variable

Some generic options

SO_BROADCAST
SO_DONTROUTE
SO_ERROR
SO_KEEPALIVE
SO_LINGER
SO_RCVBUF,SO_SNDBUF
SO_REUSEADDR

SO_BROADCAST

- Boolean option: enables/disables ability to send broadcast messages.
- Underlying link layer must support broadcasting!
- Applies only to SOCK_DGRAM sockets.
- Kernel prevents applications from sending to broadcast address if this option is not set.

SO_DONTROUTE

- Boolean option: enables bypassing of normal routing.
- Used by routing daemons to bypass the routing table and force a packet be sent out a particular interface.

SO_ERROR

- Integer value option.
- The value is an error indicator value (similar to errno).
- Readable (by get) only!
- Reading (by calling getsockopt()) clears any pending error.

SO_KEEPALIVE

- Boolean option: enabled means that TCP sockets should send a probe to peer if no data flow for a “long time”.
  - Default two hours, send a probe
  - Set as system-wide basis
- Used by TCP, allows a process to determine if:
  - Peer host has crashed, or
  - There is a network outage
- Consider what would happen to an open telnet connection without keepalive.

SO_LINGER

- Value type

```
struct linger {
  int l_onoff;    /* 0 = off */
  int l_linger;   /* time in seconds */
};
```

- Used to control whether and how long a call to close() will wait for pending ACKs
- Connection-oriented sockets only

SO_LINGER Usage

- By default, calling close() on a TCP socket will return immediately
- The closing process has no way of knowing whether or not the peer received all data
- Setting SO_LINGER means the closing process can determine that the peer machine has received the data
- But not that the data has been actually read() by the application

SO_LINGER

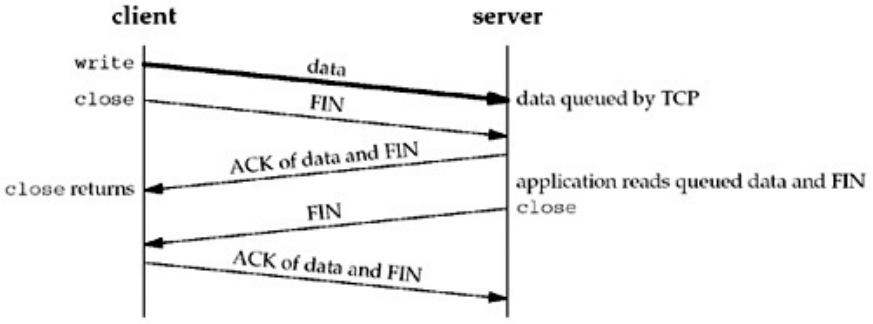
- *l_onoff* is zero, default behavior; close() returns immediately
- *l_onoff* is non-zero, *l_linger* is zero: TCP aborts with a reset
  - Avoids TIME_WAIT state
- *l_linger* non-zero, kernel puts process to sleep until:
  - All data is acked by peer, or
  - The linger time expires
- If expired, close() returns with an error and the remaining data is discarded

SO_LINGER vs shutdown()

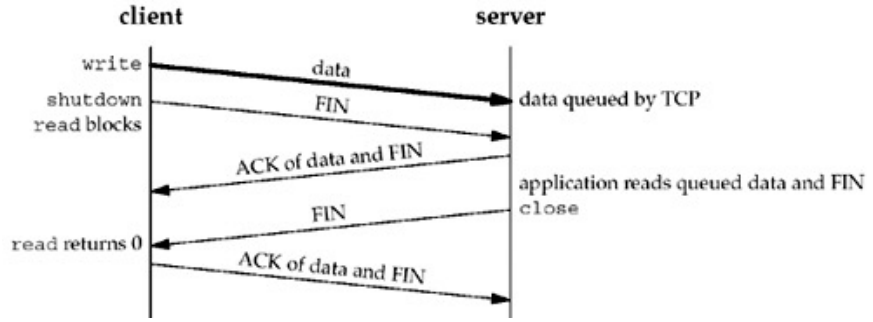
- SO_LINGER is used with close()
  - To know peer TCP acked our FIN
- Alternatively, use shutdown()
  - E.g., client can know when the server process has read all the sent data, by waiting for the server to do the close()
  - Client uses SHUT_WR instead of close()
- How parameter
  - SHUT_RD, further receives will be disallowed
  - SHUT_WR, further sends will be disallowed
  - SHUT_RDWR, further sends and receives will be disallowed

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

int  shutdown(int socket, int how);
```



Close with SO_LINGER: ACK of client FIN Fig. 7.8



Using shutdown to know peer application read data and FIN Fig. 7.10

SO_RCVBUF and SO_SNDBUF

- Integer value options to change the receive and send buffer sizes
- Can be used with TCP and UDP sockets
- With TCP, this option effects the window size used for flow control; must be set before connection is made

SO_RCVTIMEO
SO_SNDTIMEO

- Takes pointer to timeval structure, the same as the one in select()
- Disable timeout by setting its values to zero
- The receive one affects five input functions:
  - read(), readv(), recv(), recvfrom(), recvmsg()
- The send one affect five output functions
  - write(), writev(), send(), sendto(), sendmsg()

SO_REUSEADDR

- Boolean option: enables binding to a port that is already in use
- Used by listening servers that are transient and are restarted after spawning a child
  - Allows binding a listen socket to a port still in use by an active connection in another process (i.e., child)

SO_REUSEADDR (cont.)

- Can be used to establish separate servers for the same port
  - One server could be on the wildcard address
  - Others could bind to a different IP aliases for an interface
- Processes use it to receive with multicast
  - I.e., normally supported by UDP sockets

IPv4 Options

- Used with “raw” IP sockets
- IP_HDRINCL: used on raw IP sockets when we are building the IP header ourselves
- IP_TOS: allows us to set the Type-of-Service field in an IP header
- IP_TTL: allows us to set the Time-to-Live field in an IP header

TCP Socket Options

- TCP_KEEPALIVE: set the idle time used when SO_KEEPALIVE is enabled
- TCP_MAXSEG: set the maximum segment size sent by a TCP socket

TCP Socket Options (cont'd)

- TCP_NODELAY:
  - Disables TCP's Nagle algorithm that delays sending small packets if there is unACKed data pending
  - Also disables delayed ACKs (i.e., the cumulative ACKs)

Socket Options Summary

- This is just an overview
- There are many details associated with the options described
- There are many options that haven't been described

fcntl()

```
int fcntl(int fd, int cmd, long arg);
```

Posix way for miscellaneous file control operations:

- Non-blocking I/O (O_NONBLOCK, F_SETFL)
- Signal-driven I/O (O_ASYNC, F_SETFL)
- Set socket owner (F_SETOWN), i.e., (the process ID or process group ID) to receive a signal

Set non-blocking (e.g.)

```
int flags;

/* set a socket as nonblocking */
if  ((flags = fcntl (fd, F_GETFL, 0)) < 0)
    perror("F_GETFL error");
flags |= O_NONBLOCK;
if (fcntl(fd, F_SETFL, flags) < 0)
    perror("F_SETFL error");
```

ioctl()

- Used as the system interface for everything that didn't fit into some other nicely defined category
- Very OS implementation dependent
- POSIX has gotten rid of certain functionality settings by creating specific wrapper functions

```
#include <unistd.h>
int ioctl(int fd,
          int request,
          ... /* void *arg */);
```

ioctl()

- The third argument is always a pointer, but the type depends on the request
- Six different categories of requests
  - Socket operations
  - File operations
  - Interface operations
  - ARP cache operations
  - Routing table operations
  - STREAMS system

ioctl() (e.g.)

A common use for network programming is to obtain information on all of the host's interfaces

- E.g., does an interface support broadcasting or multicasting, etc
- Request and result pointer arguments

Obtain the interface configuration from the kernel

- Check the interface capabilities and details
- E.g., is the interface broadcast or multicast capable?

```
sfd = socket(AF_INET, SOCK_DGRAM, 0);
struct ifconf ifc;
ifc.ifc_buf = (caddr_t)&reqbuf[0]; // buffer
ifc.ifc_len = bufsize;  // buffer size
int rc = ioctl(sfd, SIOCGIFCONF, &ifc);
```

Socket operations

Operation	fcntl	ioctl	Routing socket	POSIX
Set socket for nonblocking I/O	F_SETFL, O_NONBLOCK	FIONBIO		fcntl
Set socket for signal-driven I/O	F_SETFL, O_ASYNC	FIOASYNC		fcntl
Set socket owner	F_SETOWN	SIOCSPGRP or FIOSETOWN		fcntl
Get socket owner	F_GETOWN	SIOCGPGRP or FIOGETOWN		fcntl
Get # bytes in socket receive buffer		FIONREAD		
Test for socket at out-of-band mark		SIOCATMARK		sockatmark
Obtain interface list		SIOCGIFCONF	sysctl	
Interface operations		SIOC[GS]IF*xxx*		
ARP cache operations		SIOC*x*ARP	RTM_*xxx*	
Routing table operations		SIOC*xxx*RT	RTM_*xxx*	

Summary of fcntl, ioctl, routing socket operation

- The first six operations can be applied to sockets by any process
- The second two (interface operations) are less common, but are still general-purpose
- The last two (ARP and routing table) are issued by administrative programs such as ifconfig and route