

Advanced Sockets Programming

Ref: Chapter 7,11,21,22

- Socket Options
 - Posix name/address conversion
 - Out-of-Band Data
 - Signal Driven I/O
- It's important to know about some of these topics, although it might not be apparent how and when to use them.
 - Details are in the book - we are just trying to get some idea of what can be done.

Socket Options

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

Option types

- Many socket options are Boolean flags indicating whether some feature is enabled (1) or disabled (0).
- Other options are associated with more complex types including `int`, `timeval`, `in_addr`, `sockaddr`, etc.

Netprog 2000 - Advanced Sockets Programming

4

Read-Only Socket Options

- Some options are readable only (we can't set the value).

Netprog 2000 - Advanced Sockets Programming

5

Setting and Getting option values

`getsockopt ()` gets the current value of a socket option.

`setsockopt ()` is used to set the value of a socket option.

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

Netprog 2000 - Advanced Sockets Programming

6

getsockopt()

```
int getsockopt( int sockfd,  
               int level,  
               int optname,  
               void *opval,  
               socklen_t *optlen);
```

level specifies whether the option is a general option or a protocol specific option (what level of code should interpret the option).

setsockopt()

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

General Options

- Protocol independent options.
- Handled by the generic socket system code.
- Some general options are supported only by specific types of sockets (SOCK_DGRAM, SOCK_STREAM).

Some Generic Options

SO_BROADCAST
SO_DONTROUTE
SO_ERROR
SO_KEEPALIVE
SO_LINGER
SO_RCVBUF, SO_SNDBUF
SO_REUSEADDR

Netprog 2000 - Advanced Sockets Programming

10

SO_BROADCAST

- Boolean option: enables/disables sending of broadcast messages.
- Underlying DL layer must support broadcasting!
- Applies only to SOCK_DGRAM sockets.
- Prevents applications from inadvertently sending broadcasts (OS looks for this flag when broadcast address is specified).

Netprog 2000 - Advanced Sockets Programming

11

SO_DONTROUTE

- Boolean option: enables bypassing of normal routing.
- Used by routing daemons.

Netprog 2000 - Advanced Sockets Programming

12

SO_ERROR

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

Netprog 2000 - Advanced Sockets Programming

13

SO_KEEPALIVE

- Boolean option: enabled means that STREAM sockets should send a *probe* to peer if no data flow for a "long time".
- Used by TCP - allows a process to determine whether peer process/host has crashed.
- Consider what would happen to an open telnet connection without keepalive.

Netprog 2000 - Advanced Sockets Programming

14

SO_LINGER

Value is of 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.

Netprog 2000 - Advanced Sockets Programming

15

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 `read()`!).

shutdown() VS SO_LINGER

- The book shows how you can use `shutdown()` to find out when the peer process has read all the sent data.

SO_RCVBUF SO_SNDBUF

- Integer values options - change the receive and send buffer sizes.
- Can be used with `STREAM` and `DGRAM` sockets.
- With `TCP`, this option effects the window size used for flow control - must be established before connection is made.

SO_REUSEADDR

- Boolean option: enables binding to an address (port) that is already in use.
- Used by servers that are transient - allows binding a passive socket to a port currently in use (with active sockets) by other processes.

SO_REUSEADDR

- Can be used to establish separate servers for the same service on different interfaces (or different IP addresses on the same interface).
- Virtual Web Servers can work this way.

IP Options (IPv4)

- IP_HDRINCL: used on raw IP sockets when we want to build 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.

another TCP socket option

- `TCP_NODELAY`: can disable TCP's Nagle algorithm that delays sending small packets if there is unACK'd data pending.
- `TCP_NODELAY` also disables delayed ACKS (TCP ACKs are cumulative).

Socket Options Summary

- This was just an overview
 - there are many details associated with the options described.
 - There are many options that haven't been described.
 - Our text is one of the best sources of information about socket options.

Posix Name/Adress Conversion

- We've seen `gethostbyname` and `gethostbyaddr` - these are protocol dependent.
 - Not part of sockets library.
- Posix includes protocol *independent* functions:

`getaddrinfo()` `getnameinfo()`

`getaddrinfo`, `getnameinfo`

- These functions provide name/address conversions as part of the sockets library.
- In the future it will be important to write code that can run on many protocols (IPV4, IPV6), but for now these functions are not widely available.
 - It's worth seeing how they work even though we probably can't use them yet!

Why `getaddrinfo()`?

- Puts protocol dependence in library (where it belongs).
 - Same code can be used for many protocols (IPV4, IPV6)
 - re-entrant function - `gethostbyname` is not!
 - Important to threaded applications.

getaddrinfo()

```
int getaddrinfo(
    const char *hostname,
    const char *service,
    const struct addrinfo* hints,
    struct addrinfo **result);
```

getaddrinfo() replaces both
gethostbyname() and getservbyname()

getaddrinfo() parameters

hostname is a hostname or an address string
(dotted decimal string for IP).

service is a service name or a decimal port
number string.

struct addrinfo

```
struct addrinfo {
    int      ai_flags;
    int      ai_family;
    int      ai_socktype;
    int      ai_protocol;
    size_t   ai_addrlen;
    char     *ai_canonname;
    struct sockaddr *ai_addr;
    struct addrinfo *ai_next;
};
```

Linked list!

getaddrinfo() hints

`hints` is an `addrinfo *` (can be `NULL`) that can contain:

- `ai_flags` (`AI_PASSIVE`, `AI_CANONNAME`)
- `ai_family` (`AF_XXX`)
- `ai_socktype` (`SOCK_XXX`)
- `ai_protocol` (`IPPROTO_TCP`, etc.)

getaddrinfo() result

`result` is returned with the address of a pointer to an `addrinfo` structure that is the head of a linked list.

It is possible to get multiple structures:

- multiple addresses associated with the `hostname`.
- The `service` is provided for multiple socket types.

addrinfo usage

```
ai_flags
ai_family
ai_socktype
ai_protocol
ai_addrlen
ai_canonname
ai_addr
ai_next
```

Used in call to `socket()`

Used in call to `bind()`, `connect()`
or `sendto()`

```
ai_flags
ai_family
ai_socktype
ai_protocol
ai_addrlen
ai_canonname
ai_addr
ai_next .....
```

getnameinfo()

```
int getnameinfo(  
    const struct sockaddr *sockaddr,  
    socklen_t addrlen  
    char *host,  
    size_t hostlen,  
    char *serv,  
    size_t servlen,  
    int flags);
```

`getnameinfo()` looks up a hostname and a service name given a `sockaddr`

Netprog 2000 - Advanced Sockets Programming

34

Out-of-Band Date

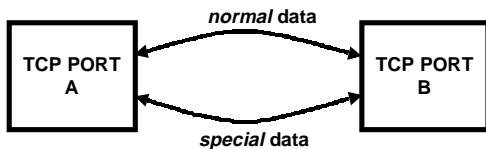
- Ever been on a date, gone to a dance club and the band doesn't show up?
 - This is becoming a serious problem:
 - The number of Internet dating services is growing exponentially.
 - The number of bands is not growing.
 - RFC 90210 proposes some short term solutions (until the number of bands can be increased).

Netprog 2000 - Advanced Sockets Programming

35

Out-of-Band *Data*

- TCP (and other transport layers) provide a mechanism for delivery of "high priority" data ahead of "normal data".
- We can almost think of this as 2 streams:



Netprog 2000 - Advanced Sockets Programming

36

TCP OOB Data

- TCP supports something like OOB data using URGENT MODE (a bit is set in a TCP segment header).
- A TCP segment header field contains an indication of the location of the urgent data in the stream (the byte number).

Netprog 2000 - Advanced Sockets Programming

37

Sending OOB Data

```
send(sd, buff, 1, MSG_OOB);
```

Use `send()` to put a single byte of urgent data in a TCP stream.

The TCP layer adds some segment header info to let the other end know there is some OOB data.

Netprog 2000 - Advanced Sockets Programming

38

Receiving OOB Data

- The TCP layer generates a `SIGURG` signal in the receiving process.
- `select()` will tell you an exception condition is present.

Netprog 2000 - Advanced Sockets Programming

39

Reading URG data (a.k.a. *re-urg-e-dating*)

- Depending on how things are set up:
 - the data can be read using `recv()` with a `MSG_OOB` flag set.
 - The data can be read *inline* and the receiving process can monitor the *out-of-band-mark* for the connection (using `socketmark()`)

Netprog 2000 - Advanced Sockets Programming

40

So what?

- OOB Data might be used:
 - a heartbeat between the client and server to detect early failure (example in the book).
 - A way to communicate an exceptional condition to a peer even when flow control has stopped the sender.

Netprog 2000 - Advanced Sockets Programming

41

Singles Driven IOU

- Another problem with Internet Dating services is the lack of single drivers in many metropolitan areas.
 - Neither participant can pick up the other.
 - Dating protocols degrade to involve online communication only.
 - Proxy drivers (running TAXI protocol) get overloaded and refuse IOU packets.

Netprog 2000 - Advanced Sockets Programming

42

Signal Driven I/O

- We can tell the kernel to send us a **SIGIO** signal whenever something happens to a socket descriptor.
- The signal handler must determine what conditions caused the signal and take appropriate action.

Netprog 2000 - Advanced Sockets Programming

43

Signal Driven UDP

- SIGIO occurs whenever:
 - an incoming datagram arrives.
 - An asynchronous error occurs.
 - Could be ICMP error (unreachable, invalid address, etc).
- Could allow process to handle other tasks and still watch for incoming UDP messages.

Netprog 2000 - Advanced Sockets Programming

44

Signal-Driven Example

- Real life signal-driven UDP example described in the book:
 - NTP Network Time Protocol.
 - Used to record timestamp of arrival of UDP datagram.

Netprog 2000 - Advanced Sockets Programming

45

Signal Driven TCP (very rare)

- **SIGIO** occurs whenever:
 - an incoming connection has completed.
 - Disconnect request initiated.
 - Disconnect request completed.
 - Half a connection shutdown.
 - Data has arrived.
 - Data has been sent (indicating there is buffer space)
 - asynchronous error

Netprog 2000 · Advanced Sockets Programming

46
