Multimedia Computing

Sockets

Outline

- Socket basics
- Socket details (TCP and UDP)
- Socket options
- Final notes

Socket Basics

- An end-point for a IP network connection
  - what the application layer "plugs into"
  - programmer cares about Application Programming Interface (API)
- End point determined by two things:
  - Host address: IP address is Network Layer
  - Port number: is Transport Layer
- Two end-points determine a connection: socket pair
  - ex: 206.62.226.35,p21 + 198.69.10.2,p1500
  - ex: 206.62.226.35,p21 + 198.69.10.2,p1499

Ports

- Numbers (vary in BSD, Solaris, Linux):
  - 0-1023 "reserved", must be root
  - 1024 - 5000 "ephemeral"
  - however, many systems allow > 3977 ports
    - (50,000 is correct number)
- Well-known, reserved services /etc/services:
  - ftp 21/tcp
  - telnet 23/tcp
  - finger 79/tcp
  - snmp 161/udp

Sockets and the OS

User Application
Socket
Operating System
Transport Layer

- User sees "descriptor", integer index
  - like: `FILE *`, or file index from `open()`
  - returned by `socket()` call (more later)

Transport Layer

- UDP: User Datagram Protocol
  - no acknowledgements
  - no retransmissions
  - out of order, duplicate possible
  - connectionless
- TCP: Transmission Control Protocol
  - reliable (in order, all arrive, no duplicates)
  - flow control
  - connection
  - duplex

Outline

- Socket basics
- *Socket details (TCP and UDP)*
- Socket options
- Final notes

Socket Details Outline

Unix Network Programming, W. Richard Stevens, 2nd edition, ©1998, Prentice Hall

- Addresses and Sockets
- TCP client-server (talk-tcp, listen-tcp)
- UDP client-server (talk-udp, listen-udp)
- Misc stuff
  - setsockopt(), getsockopt()
  - fcntl()

Addresses and Sockets

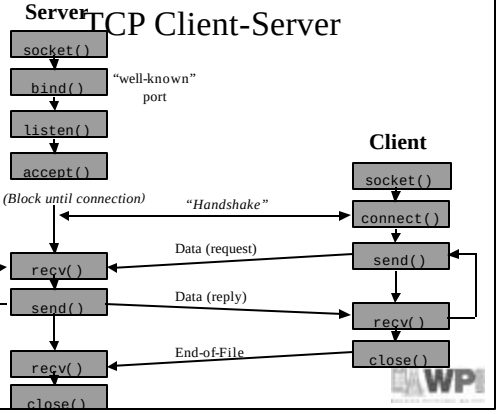
- Structure to hold address information
- Functions pass address from user to OS
  `bind()`
  `connect()`
  `sendto()`
- Functions pass address from OS to user
  `accept()`
  `recvfrom()`

Socket Address Structure

```
struct in_addr {
  in_addr_t s_addr;          /* 32-bit IPv4 addresses */
};
struct sockaddr_in {
  unit8_t      sin_len;      /* length of structure */
  sa_family_t  sin_family;   /* AF_INET */
  in_port_t    sin_port;     /* TCP/UDP Port num */
  struct in_addr sin_addr;   /* IPv4 address (above) */
  char sin_zero[8];          /* unused */
}
```

- Are also "generic" and "IPv6" socket structures

TCP Client-Server



`socket()`

`int socket(int family, int type, int protocol);`

Create a socket, giving access to transport layer service.

- *family* is one of
  - AF_INET (IPv4), AF_INET6 (IPv6), AF_LOCAL (local Unix),
  - AF_ROUTE (access to routing tables), AF_KEY (new, for encryption)
- *type* is one of
  - SOCK_STREAM (TCP), SOCK_DGRAM (UDP)
  - SOCK_RAW (for special IP packets, PING, etc. Must be root)
    - setuid bit `(-rws--x--x root 1997 /sbin/ping*)`
- *protocol* is 0 (used for some raw socket options)
- upon success returns socket descriptor
  - Integer, like file descriptor
  - Return -1 if failure

bind()

```
int bind(int sockfd, const struct sockaddr *myaddr,
      socklen_t addrlen);
```

Assign a local protocol address ("name") to a socket.

- *sockfd* is socket descriptor from `socket()`
- *myaddr* is a pointer to address struct with:
  - *port number* and *IP address*
  - if port is 0, then host will pick ephemeral port
    - not usually for server (exception RPC port-map)
  - IP address != INADDR_ANY (multiple net cards)
- *addrlen* is length of structure
- returns 0 if ok, -1 on error
  - EADDRINUSE ("Address already in use")

listen()

```
int listen(int sockfd, int backlog);
```

Change socket state for TCP server.

- *sockfd* is socket descriptor from `socket()`
- *backlog* is maximum number of *incomplete* connections
  - historically 5
  - rarely above 15 on a even moderate Web server!
- Sockets default to active (for a client)
  - change to passive so OS will accept connection

accept()

```
int accept(int sockfd, struct sockaddr
     cliaddr, socklen_t *addrlen);
```

Return next completed connection.

- *sockfd* is socket descriptor from `socket()`
- *cliaddr* and *addrlen* return protocol address from client
- returns brand new descriptor, created by OS
- if used with `fork()`, can create concurrent server

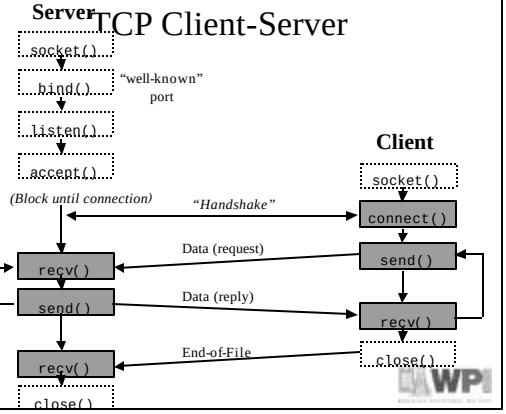
close()

```
int close(int sockfd);
```

Close socket for use.

- *sockfd* is socket descriptor from `socket()`
- closes socket for reading/writing
  - returns (doesn't block)
  - attempts to send any unsent data
  - socket option SO_LINGER
    - block until data sent
    - or discard any remaining data
  - Returns -1 if error

TCP Client-Server



connect()

```
int connect(int sockfd, const struct
sockaddr *servaddr, socklen_t addrlen);
```

Connect to server.

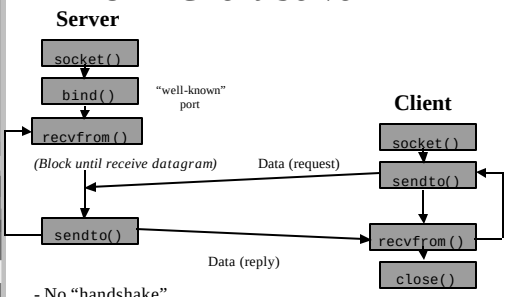
- *sockfd* is socket descriptor from `socket()`
- *servaddr* is a pointer to a structure with:
  - *port number* and *IP address*
  - must be specified (unlike `bind()`)
- *addrlen* is length of structure
- client doesn't need `bind()`
  - OS will pick ephemeral port
- returns socket descriptor if ok, -1 on error

Sending and Receiving

```
int recv(int sockfd, void *buff, size_t
  mbytes, int flags);
int send(int sockfd, void *buff, size_t
  mbytes, int flags);
```

- Same as `read()` and `write()` but for *flags*
  - MSG_DONTWAIT (this send non-blocking)
  - MSG_OOB (out of band data, 1 byte sent ahead)
  - MSG_PEEK (look, but don't remove)
  - MSG_WAITALL (don't give me less than max)
  - MSG_DONTROUTE (bypass routing table)

UDP Client-Server



- No "handshake"
- No simultaneous close
- No `fork()` for concurrent servers!

Sending and Receiving

```
int recvfrom(int sockfd, void *buff, size_t mbytes, int
  flags, struct sockaddr *from, socklen_t *addrlen);
int sendto(int sockfd, void *buff, size_t mbytes, int
  flags, const struct sockaddr *to, socklen_t
  addrlen);
```

- Same as `recv()` and `send()` but for *addr*
  - `recvfrom` fills in address of where packet came from
  - `sento` requires address of where sending packet to

`connect()` with UDP

- Record address and port of peer
  - datagrams to/from others are not allowed
  - does not do three way handshake, or connection
  - "connect" a misnomer, here. Should be `setpeername()`
- Use `send()` instead of `sendto()`
- Use `recv()` instead of `recvfrom()`
- Can change connect or unconnect by repeating `connect()` call
- (Can do similar with `bind()` on receiver)

Why use connected UDP?

- Send two datagrams unconnected:
  - connect the socket
  - output first dgram
  - unconnect the socket
  - connect the socket
  - ouput second dgram
  - unconnect the socket
- Send two datagrams connected:
  - connect the socket
  - output first dgram
  - ouput second dgram

Socket Options

- setsockopt(), getsockopt()
- SO_LINGER
  - upon close, discard data or block until sent
- SO_RCVBUF, SO_SNDBUF
  - change buffer sizes
  - for TCP is "pipeline", for UDP is "discard"
- SO_RCVLOWAT, SO_SNDLOWAT
  - how much data before "readable" via select()
- SO_RCVTIMEO, SO_SNDTIMEO
  - timeouts

Socket Options (TCP)

- TCP_KEEPALIVE
  - idle time before close (2 hours, default)
- TCP_MAXRT
  - set timeout value
- TCP_NODELAY
  - disable Nagle Algorithm

fcntl()

- 'File control' but used for sockets, too
- Signal driven sockets
- Set socket owner
- Get socket owner
- Set socket non-blocking

```
flags = fcntl(sockfd, F_GETFL, 0);
flags |= O_NONBLOCK;
fcntl(sockfd, F_SETFL, flags);
```

- Beware not getting flags before setting!