Distributed Computing Systems

Sockets

Outline

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

Socket Basics (1 of 2)

- An end-point for an Internet network connection
  - what application layer "plugs into"

User Application
Socket
Operating System
Transport Layer
Network Layer

- User sees "descriptor" - integer index or object handle
  - like: `FILE *`, or file index from `open()`
  - returned by `socket()` call (more later)
  - programmer cares about Application Programming Interface (API)

Socket Basics (2 of 2)

- 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
  - c1: `206.62.226.35,p21+198.69.10.2,p1500`
  - c2: `206.62.226.35,p21+198.69.10.2,p1499`

Ports

- Numbers (below is typical, since vary by OS):
  - `0-1023` "reserved", must be root
  - `1024-5000` "ephemeral", temporary use
  - Above `5000` for general use
    - (`50,000` is specified max)
- Well-known, reserved services (see `/etc/services` in Unix). E.g.,

FTP	21
HTTP	80
IMAP	220
World of Warcraft	1119 & 3724

Transport Layer

- *UDP*: User Datagram Protocol
  - no acknowledgements
  - no retransmissions
  - out of order, duplicates possible
  - connectionless
- *TCP*: Transmission Control Protocol
  - reliable (in order, all arrive, no duplicates)
  - flow control
  - connection-based
- Note, TCP ~95% of all flows and packets on Internet
  - (What applications don't (always) use UDP?)
  - (What protocol for distributed shell?)

Outline

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

Socket Details Mini-Outline

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

Beej's Guide to Network Programming

- Project 2 → Includes links to samples
  - TCP Server and TCP Client (both in C)
- Addresses and Sockets
- Examples (`talk-tcp`, `listen-tcp`, …)
- Misc stuff
  - `setsockopt(),getsockopt()`
  - `fcntl()`

Addresses and Sockets

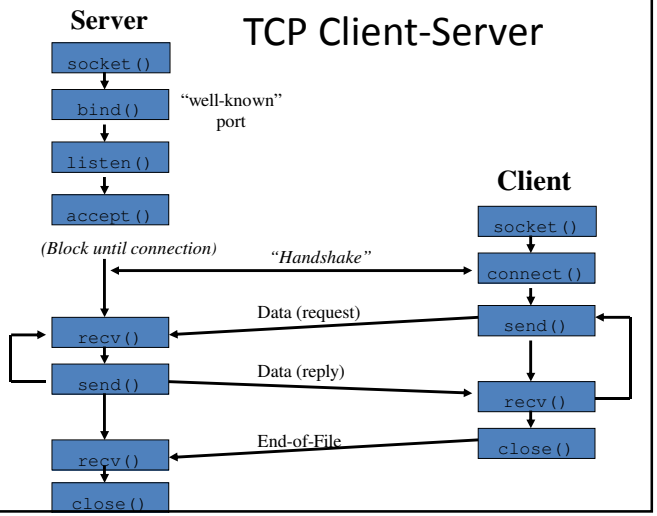
- Structure to hold address information
- Functions pass address from user to OS
  - `bind()`
  - `connect()`
  - `sendto()`
- Functions pass address from OS to user
  - `accept()`
  - `recvfrom()`

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 (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 `(-rwsr-xr-x root 2014 /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 local protocol address ("name") to socket

- *sockfd* is socket descriptor from `socket()`
- *myaddr* is 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` (unless multiple nics)
- *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 (to passive) for TCP server

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

accept()

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

Return next completed connection.

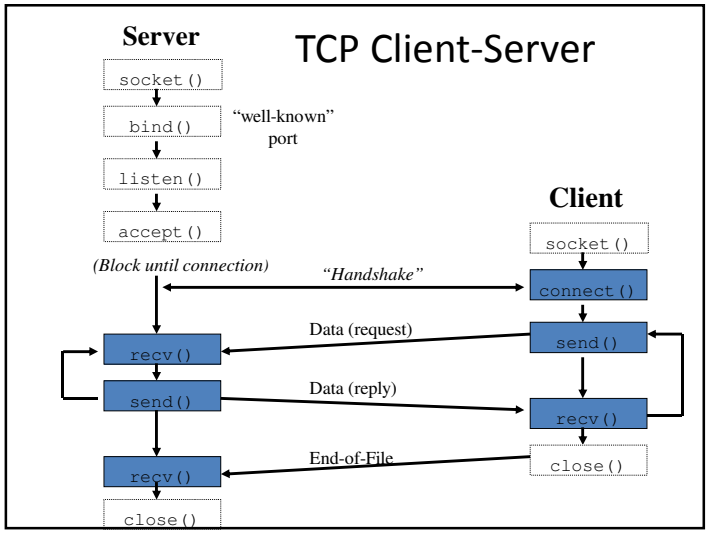
- *sockfd* is socket descriptor from `socket()`
- *cliaddr* and *addrlen* return protocol address from client
- returns brand new descriptor, created by OS
- note, if create new process or thread, can create concurrent server

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



connect()

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

Connect to server.

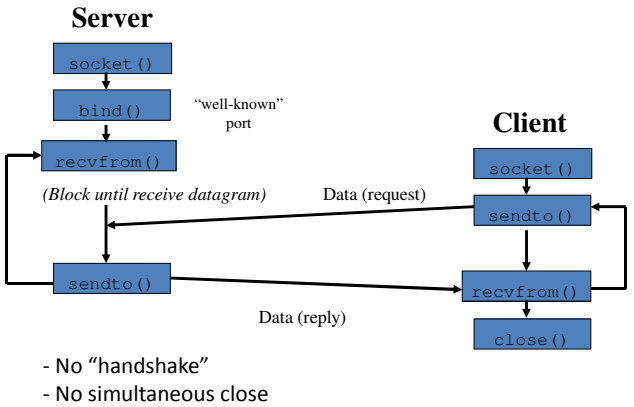
- *sockfd* is socket descriptor from `socket()`
- *servaddr* is pointer to structure with:
  - *port number* and *IP address*
  - must be specified (unlike `bind()`)
- *addrlen* is length of structure
- client doesn't need `bind()`
  - OS will pick ephemeral port
- returns socket descriptor if ok, -1 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 return less than `mbytes`)
  - MSG_DONTROUTE (bypass routing table)



- No "handshake"
- No simultaneous close

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
  - `sendto` 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

Outline

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

Socket Options (TCP)

- TCP_KEEPALIVE
  - idle time before close (2 hours, default)
- TCP_MAXRT
  - set timeout value
- TCP_NODELAY
  - disable Nagle Algorithm
  - won't buffer data for larger chunk, but sends immediately

`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!

Final Notes – Distributed Shell

- TCP (not UDP)
- *Does* need to handle more than one client at a time (a *concurrent* server)
- Refer to sample code online (`talk`, `listen`)
- Recommendation:
  - Develop shell independently of sockets