CS 640 Introduction to Computer Networks

Lecture 2

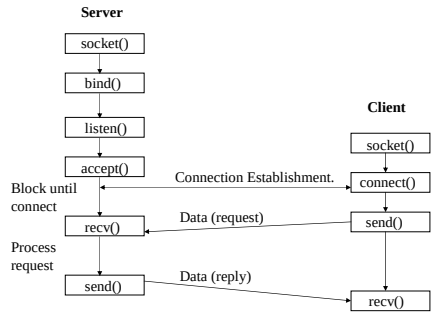
Today's lecture

- Application programming interface (sockets)
- Performance metrics

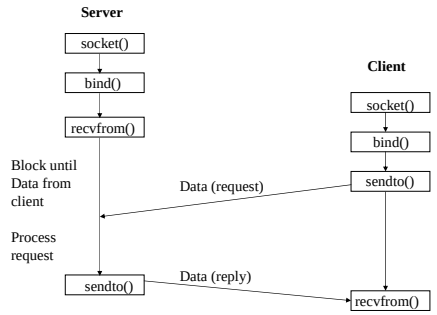
Berkeley Sockets

- Networking protocols are implemented as part of the OS
  - The networking API exported by most OS's is the *socket interface*
  - Originally provided by BSD 4.1c ~1982.
- The principal abstraction is a socket
  - Point where an application attaches to the network
  - Operations: creating connections, attaching to network, sending/receiving data, closing.

Connection-oriented example (TCP)



Connectionless example (UDP)



Ports (multiplexing)

- How does the OS know whether one wants to connect to the web server or the email server?
- How does the OS know which process to deliver the data to?
- 16 bit port numbers are used
  - Both source and destination have a port number
  - Servers have well known port numbers <1024
- How can the OS tell TCP packets from UDP?
  - Protocol number is part of IP header

Socket call

- Means by which an application attached to the network
- int socket(int family, int type, int protocol)
- *family*: address family (protocol family)
  - AF_UNIX, AF_INET, AF_NS, AF_IMPLINK
- *type*: semantics of communication
  - SOCK_STREAM, SOCK_DGRAM, SOCK_RAW
  - Not all combinations of family and type are valid
- *protocol*: Usually set to 0 but can be set to specific value.
  - Family and type usually imply the protocol
- Return value is a *handle* for new socket

Bind call

- Binds a new socket to the specified address
- int bind(int socket, struct sockaddr *address, int addr_len)
- *socket*: newly created socket handle
- *address*: data structure with *local* address
  - IP address and port number (demux keys)
    - Can use well known port or unique port

Listen call

- Connection-oriented servers use it to indicate they are willing to receive connections
- Int listen(int socket, int backlog)
- *socket*: handle of newly creates socket
- *backlog*: number of connection requests that can be queued by the system while waiting for server to execute accept call.

Accept call

- After *listen*, the accept call performs a *passive open* (server prepared to accept connects).
- int accept(int socket, struct sockaddr *address, int addr_len)
- It blocks until a remote client carries out a connection request
- When it does return, it returns with a *new* socket that corresponds with new connection and the address contains the clients address

Connect call

- Client executes an *active open* of a connection
- Int connect(int socket, struct sockaddr *address, int addr_len)
- Call does not return until the three-way TCP handshake is complete
- Address field has remote system's address
- Client OS usually selects random, unused port

send(to), recv(from)

- After connection has been made, application uses send/recv to data
- int send(int socket, char *message, int msg_len, int flags)
  - Send specified message using specified socket
- int recv(int scoket, char *buffer, int buf_len, int flags)
  - Receive message from specified socket into specified buffer

Performance Metrics

- Bandwidth: physical property of link
- Throughput: actual data transmitted per time unit
  - notation
    - KB = 2^{10} bytes
    - Mbps = 10^6 bits per second
- Latency (delay)
  - time to send message from point A to point B
  - one-way versus round-trip time (RTT)

    Latency = Propagation + Transmit
    Propagation = Distance / Speed (of light)
    Transmit = Size / Bandwidth
- Delays on Internet much greater (queuing)

Bandwidth versus Latency

- Relative importance
- Assume propagation delay is 100 ms
- Transfer 1 Kb, bw 1 Mbps
  - Latency: 100 + 1 (transmission delay) = 101 ms
- Transfer 1 Mb
  - Latency 100 + 1000 (transmission delay) = 1100 ms