Introduction to Socket Programming

Why Socket Programming?
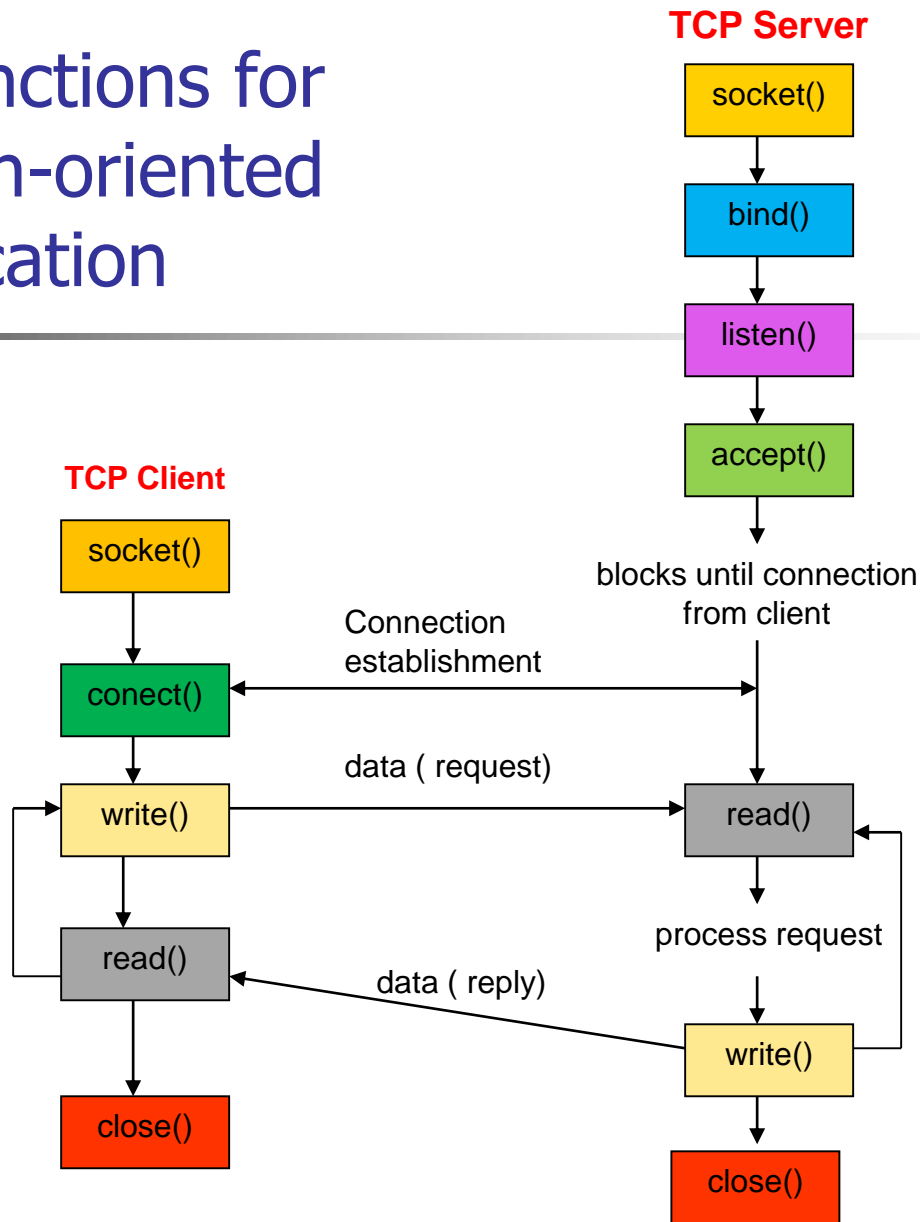
- To build any network application
  - Web browser
  - FTP

Client – Server model

- Server – provider of information
- Client – seeker of information
- Eg. Apache server – web browser

Socket functions for connection-oriented communication

TCP Server

socket()

bind()

listen()

accept()

blocks until connection from client

TCP Client

socket()

conect()

Connection establishment

write()

data (request)

read()

process request

read()

data (reply)

write()

close()

close()

4

Data structures

Defined by including the <netinet/in.h> header

struct sockaddr
{
unsigned short *sa_family*;
// address family, AF_xxx
 	
char *sa_data*[14];
// 14 bytes of protocol address
};
	// IPv4 AF_INET sockets:
	
	struct sockaddr_in
	{
	short *sin_family*;
	// e.g. AF_INET, AF_INET6
	unsigned short *sin_port*;
	// e.g. htons(3490)
	struct in_addr *sin_addr*;
	// see struct in_addr, below
	char *sin_zero*[8];
	// zero this if you want to
	      };
		struct in_addr
		{
		unsigned long *s_addr*;
		// load with inet_pton()
		};

Choice of Port number

- Choose a port number that is registered for general use, from 1024 to 49151
  - Do not use ports 1 to 1023. These ports are reserved for use by the Internet Assigned Numbers Authority (IANA)
  - Avoid using ports 49152 through 65535. These are dynamic ports that operating systems use randomly. If you choose one of these ports, you risk a potential port conflict

6

Byte ordering

- Byte ordering or Endianess is the attribute of a system which indicates whether integers are stored / represented left to right or right to left.

- Example 1: short int x = 0xAABB (hex)

  This can be stored in memory as 2 adjacent bytes as either (0xaa , 0xbb) or as (0xbb, 0xaa).

  Big Endian:

  Byte Value :   [0xAA] [0xBB]
  Memory   :   [0][1]

  Little Endian:

  Byte Value :   [0xBB] [0xAA]
  Memory   :   [0][1]

- All Network data is sent in Big Endian format.
- In the networking world we call this representation as Network Byte Order and native representation on the host as Host Byte Order.
- We convert all data into Network Byte Order before transmission.

Other functions

- Byte Ordering:

  Host Byte Order to Network Byte Order:
  htons() , htonl()
  Network Byte Order to Host Byte Order:
  ntohs() , ntohl()

- IP Address format:

  Ascii dotted to Binary: inet_aton()
  Binary to Ascii dotted: inet_ntoa()

Socket()

- int s = socket(domain, type, protocol);
where
  - s: socket descriptor, an integer (like a file-handle)
  - domain: integer, communication domain
    - e.g., AF_INET (IPv4 protocol)
    - **Note. We'll use AF_INET**
  - type: communication type
    - SOCK_STREAM: reliable, 2-way, connection-based service
    - SOCK_DGRAM: unreliable, connectionless
    - **Note. We'll use SOCK_STREAM**
  - protocol: **We'll set to 0**

Bind()

- The bind function assigns a local protocol address to a socket.
- The protocol address is the combination of either a 32-bit IPV4 address or a 128-bit IPV6 address, along with a 16-bit port number

#include <sys/socket.h>

int bind(int sockfd, struct sockaddr *address, int addr_len)

- sockfd: a socket descriptor returned by the socket()
- *address: a pointer to a protocol-specific address.
- addrlen: the size of the socket address structure

- Returns on success: 0, on error: -1

Listen()

- The listen function is called only by a TCP server to converts an unconnected socket into a passive socket.

  #include <sys/socket.h>

  int listen (int sockfd, int backlog);

- sockfd: a socket descriptor
- backlog: maximum number of connections that the kernel should queue for this socket

- Returns on success: 0, on error: -1

Accept()

- The accept function is called by the TCP server to return the next completed connection

#include<sys/socket.h>

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

- sockfd: socket descriptor
- *cliaddr: used to return the protocol address of the connected peer process
- *addrlen: length of the address

- Returns on success: a new (connected)socket descriptor, on error:-1

Connect()

- The connect function is used by a TCP client to establish a connection with a TCP server

#include<sys/socket.h>

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

- sockfd: a socket descriptor
- *servaddr: a pointer to a socket address structure
- addrlen: the size of the socket address structure

- Returns on success: 0, on error: -1

Read()

- The read function is used to receive data from the specified socket

#include <unistd.h>

int read(int sockfd, const void * buf, int nbytes);

- sockfd: a socket descriptor
- buf: buffer to store the data.
- nbytes: size of the buffer

- Returns: number of bytes read if OK,0 on EOF, -1 on error

Write()

- The write function is used to send the data through the specified socket

#include <unistd.h>

int write(int sockfd, const void * buf, int nbytes);

- sockfd: a socket descriptor
- buf: buffer to store the data.
- nbytes: size of the buffer

- Returns: number of bytes written if OK,0 on EOF, -1 on error

Close()

- The close function is used to close a socket and terminate a connection

  #include <unistd.h>

  int close (int sockfd);

- sockfd: This socket descriptor is no longer useable

- Returns on success: 0, on error: -1