Socket Programming

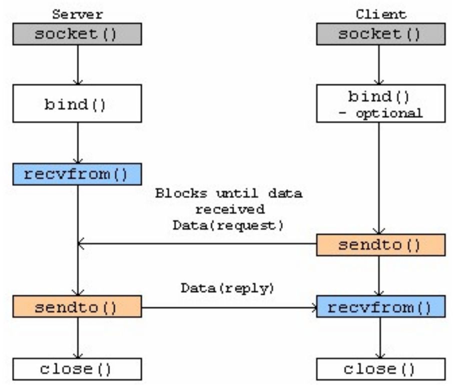
- **Introduction to Network communication models**
  Two hosts can communicate over the network using various models. One of the most common models which we experience in our day to day usage of internet is the Client-Server communication model. The other model is the peer to peer communication model. Another emerging trend is to combine both of these models to make network communication more efficient.
  - **Client-Server Communication**
    In the Client-Server model, one of the communicating hosts acts as a server while one or more than one nodes act as clients. Clients initiate communication with the server and server, depending upon services, serves the content to clients. For example, when you browse various websites, you actually act as a client and the web server which serves you web pages is called the server. In all of the instances of web surfing, it is always the job of the client to initiate the communication.
  - **Peer-Peer Communication**
    The second model is the peer-to-peer communication. In this model, two peers/nodes/users can directly communicate with each other without involving a centralized server. Any user can initiate a connection with any other node so each user can act as a server as well as a client. Bit torrent is an example of peer-peer communication.
  - **Hybrid Approach**
    To deal with the emerging trends of next generation network applications, engineers and researchers have combined both these models to gain maximum out of network communication. Skype, whatsapp and viber are few examples of hybrid architecture.

- **Introduction to Socket programming**
  A socket is an interface provided by the operating system between the application layer and the transport layer. It is used by application programmers for socket programming so that nodes running their applications communicate with other nodes. In socket programming, we use send and receive functions to send data to and receive data from remote nodes. We also need addresses of the remote nodes to communicate with them.

- **Byte Ordering**
  Local and remote addresses in network communication are a combination of a port and an IP address. Both of these are multi byte addresses. The port number is a 16 bit number while the IP address is a 32 bit number. While communicating between nodes having different architectures, there is a need to decide a standard multi-byte ordering mechanism. For this matter, researchers have decided to use big-endian architecture which is also known as network byte order. Therefore, we will have to change both the IP

Address and the port address to the network address before sending it over the network. We will also have to reverse the process on receiving node if we are interested in knowing the source address details.

- **UDP based Client Server Communication**
  UDP is a connectionless protocol on transport layer. Clients do not need to establish a communication channel before starting communication. Any client can send packets to the server at any instant of time without establishing the connection. Moreover, in UDP, a single socket can send and receive packets to and from various nodes. There is also no guarantee of data to be delivered correctly to the destination. Packets may get corrupted or lost altogether or may arrive out of order. There are few steps involved in network communication using UDP which are shown in the diagram below.



  Each of the steps in the diagram is a function in socket programming. All of these functions are explained below:

  - **int Sock = Socket(int domain, int type, int protocol)**
    This socket function is used to create a new socket for communication. It returns a new socket file descriptor which is unique for every socket. Its parameters are given in detail as follows:
    - **domain**
      AF_INET/PF_INET
    - **type**
      SOCK_DGRAM/SOCK_STREAM
    - **protocol**
      0/getprotobyname()
    - **sock**
      Socket File Descriptor. -1 is returned in case of any error.
    - **Usage**
      int sockfd = socket (AF_INET, SOCK_DGRAM, 0);

- **int bind(int sock, struct sockaddr *my_addr, int addrlen);**
  The bind function is used to assign an address to the socket. This address includes a port number and an IP address. Its parameters are explained below:
  - **sock**
    Socket file descriptor returned by socket() call.
  - **my_addr**
    Pointer to struct *sockaddr* which contains information about local domain, IP address and Port number.
  - **addrlen**
    It is simply the size of *sockaddr*.
  - **Usage**
    bind(sockfd, (struct sockaddr *)&my_addr, sizeof(struct sockaddr));

- **int sb = sendto(int sock, const void* msg, int len, unsigned int flags, const struct sockaddr *to, socklen_t to_len)**
  Once the socket is bound to an address, the socket can be used to send and receive data. sendto method is used to send data to the destination. Its parameters are explained as follows:
  - **sock**
    Socket that is created for communication by the socket() call.
  - **msg**
    Data buffer. For ease, you can use char array.
  - **len**
    Size of data buffer.
  - **Flags**
    Provides more control over data transmission.
  - **to**
    Contains information about remote peer.
  - **to_len**
    it is the size of the to structure.
  - **sb**
    Number of bytes actually sent or -1 in case of any error.

- **recvfrom (int sock, void *buf, int len, unsigned int flags, struct sockaddr* from, socklen_t from_len)**
  - **sock**
    Socket that is created for communication.
  - **buf**
    Data buffer to receive data sent by remote peer.
  - **len**
    Size of the data buffer.

    - **flags**
      Provides more control over data reception.
    - **from**
      Address information of the remote peer.
    - **from_len**
      it is simply the size of from.
    - **rb**
      Returns number of bytes received or -1 in case of any error.

  - **close(sock)**
    At the end of communication it is necessary to gracefully terminate the communication session by closing down the socket. Close socket closes the socket passed to it.

- **Lab Activity**

  Task:

  Your task is to write a simple UDP client and a UDP server. You will send your roll number and full name in a packet as data to the server. Upon receiving your packet, the server should mark your inlab marks (random) and attendance and display it. On receiving your message, the server should also send a success message to the client.

  Information you need for successful communication:
  - Server IP
  - Server port no.

  Expected Packet Data Format sent to the server:

  You will send data in the following format:

  Your_roll_number|your_name. For example, if your roll number is 2012-EE-280 and your name is Owais Ahmed, you will send data as ***2012-EE-280|Owais Ahmed***.

  Output:

  Displayed output should be in the following format:

Roll No.	Name	IP Address	Port No.
Owais Ahmed	2012-EE-280	192.168.1.4	34516

- **Submission Email ID**

  ee436s2015@gmail.com