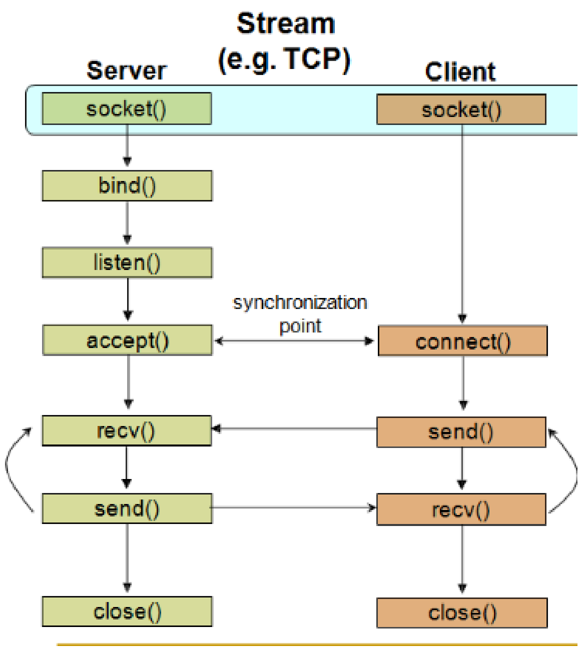
SOCKET PROGRAMMING

(study the connection using TCP and UDP after that try to implement the logic)

1. TCP Connection:



Server:

- Intialize sock_addr structure
- Create socket :socket()
- Set the configuration for severaddress (sock_addr structure)
- Bind server socket to specific IP address and port number :bind()
- Listen for incoming connections :listen()
- Accept the incoming connection: accept()
- Send or receive data
- Close the socket

```
#include<stdio.h>
#include<arpa/inet.h>
#include <unistd.h>
#define port 5000
void main()
{
        struct sockaddr_in serveraddr,newaddr; //initializing the structure for
storing serveraddress configuration and also newaddress to store the
address of the client
```

```
        int sersocket,newsocket,s,size;
        char buffer[100];


        sersocket=socket(PF_INET,SOCK_STREAM,0);//create a socket and if
socket created successfully it will return a positive value


        if(sersocket>0)//checking whether the socket is created
                printf("\nserver socket is created");


//Configuring the server address
        serveraddr.sin_family= PF_INET;  //used to specify the communication
domain
        serveraddr.sin_port= htons(port); //htons() convert the host byte order
to network byte order(big-endian)

        serveraddr.sin_addr.s_addr=htonl(INADDR_ANY);//htonl() is used to convert 32
bit host byte order to network byte order (big-endian)


        s=bind(sersocket,(struct sockaddr *)&serveraddr,sizeof(serveraddr));//Bind server
socket to specific IP address and port number, it will return 0 is the bind is successful


        if(s==0)//check whether the bind is successful
                printf("\nbind success");


        listen(sersocket,1); //Listen for incoming connections


        size=sizeof(newaddr);
        printf("\nserver ready");

        newsocket=accept(sersocket,(struct sockaddr *)&newaddr,&size);//accept the
incoming connection, it returns a positive value it the connection is accepted
```

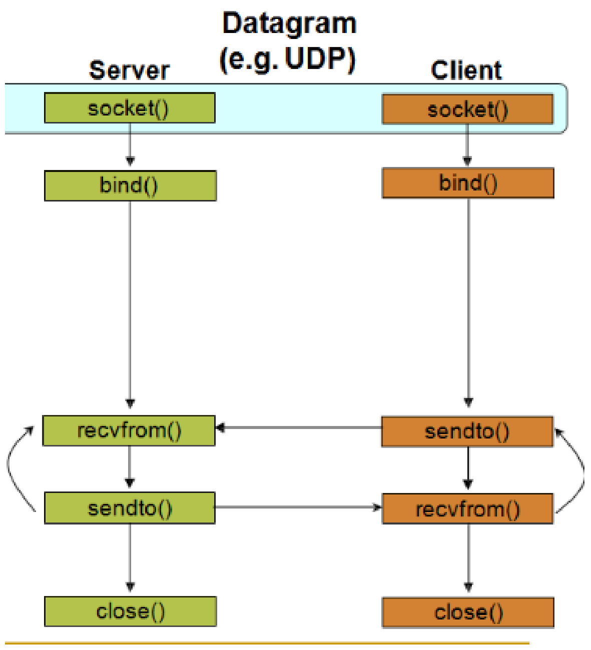
```
        if(newsocket>0) //check whether the connection is accepted
                printf("\naccepted");

//After the connection you can write the code to send or receive any data to the client
using send() or recv()

        recv(newsocket,buffer,1024,0);
        printf("\ndata received from client: %s\n",buffer);

printf("\nEnter string:");
        scanf("%s",buffer);
        send(newsocket,buffer,sizeof(buffer),0);


        close(sersocket);
}
```

Client:

- Intialize sock_addr structure
- Create socket :socket()
- Set the configuration for severaddress (sock_addr structure)
- Establish a connection to the remote server :connect()
- Send or receive data
- Close the socket/connection

```
#include<stdio.h>
#include<arpa/inet.h>
#include <unistd.h>
#define port 5000

void main()
{
      struct sockaddr_in serveraddr; //initializing the structure for
storing serveraddress configuration



      int clisocket;
      char buffer[100];


      clisocket=socket(PF_INET,SOCK_STREAM,0);//create a socket and if
socket created successfully it will return a positive value




      if(clisocket>0)//checking whether the socket is created
            printf("\nclient socket created");
//Configuring the server address




      serveraddr.sin_family= PF_INET; //used to specify the
communication domain

      serveraddr.sin_port= htons(port);//htons() convert the host byte
order to network byte order(big-endian)

      serveraddr.sin_addr.s_addr=inet_addr("127.0.0.1");// mention the
ip address, inet_addr() is used to convert the dotted ip address into
32 bit binary representation




      connect(clisocket,(struct
sockaddr*)&serveraddr,sizeof(serveraddr)); //used to connect to the
server




//After the connection you can write the code to  send or receive any
data using send() or  recv() to the server

      printf("\nEnter string:");
      scanf("%s",buffer);
      send(clisocket,buffer,sizeof(buffer),0);

recv(clisocket,buffer,1024,0);
printf("\ndata received from server: %s\n",buffer);



      close(clisocket);
}
```

2. UDP Connection



Server:

- Intialize sock_addr structure
- Create socket :socket()
- Set the configuration for severaddress (sock_addr structure)
- Bind server socket to specific IP address and port number :bind()
- Send or receive data (using sendto() and recvfrom())
- Close the socket

Same thing as TCP connection the only difference is there is no accept() and listen() it is directly sending the data after the configuration of the server address as it is a wireless connection

```
#define port 4000
void main()
{
	struct sockaddr_in serveraddr,newaddr;
	int sersocket,s,size;
	char buffer[100];
	sersocket=socket(AF_INET,SOCK_DGRAM,0);
	if(sersocket>0)
		printf("\nServer socket created");
	serveraddr.sin_family=AF_INET;
	serveraddr.sin_port=htons(port);
	serveraddr.sin_addr.s_addr=htonl(INADDR_ANY);
	s=bind(sersocket,(struct
sockaddr*)&serveraddr,sizeof(serveraddr));
	if(s==0)
		printf("\nBind success");
//Here you can mention the code to send or receive data from the server

      size=sizeof(newaddr);
      recvfrom(sersocket,buffer,1024,0,(struct
sockaddr*)&newaddr,&size);
      printf("\nData recieved from client:%s\n",buffer);

printf("\nEnter string:");
      scanf("%s",buffer);
      sendto(sersocket,buffer,sizeof(buffer),0,(struct
sockaddr*)&serveraddr,sizeof(serveraddr));

      close(sersocket);
}
```

Client:

- Intialize sock_addr structure
- Create socket :socket()
- Set the configuration for severaddress (sock_addr structure)
- Send or receive data :sendto() or recvfrom()
- Close

Same thing as TCP connection the only difference is there is no connect() it is directly sending the data after the configuration of the server address as it is a wireless connection

```
#include<stdio.h>
#include<arpa/inet.h>
#include <unistd.h>
#define port 4000
void main()
{
      struct sockaddr_in serveraddr, newaddr;
      int clisocket;
      char buffer[100];
      clisocket=socket(AF_INET,SOCK_DGRAM,0);
      if(clisocket>0)
            printf("\nClient socket created");
      serveraddr.sin_family=AF_INET;
      serveraddr.sin_port=htons(port);
      serveraddr.sin_addr.s_addr=inet_addr("127.0.0.1");

//Here you can mention the code to send or receive data from the client
      printf("\nEnter string:");
      scanf("%s",buffer);
      sendto(clisocket,buffer,sizeof(buffer),0,(struct
sockaddr*)&serveraddr,sizeof(serveraddr));
size=sizeof(newaddr);
      recvfrom(clisocket,buffer,1024,0,(struct
sockaddr*)&newaddr,&size);
      printf("\nData recieved from server:%s\n",buffer);


      close(clisocket);
}
```

```
To access the manual page of different commands use: man [commands]
Eg: man socket
```

1. socket():
   - Syntax: Create a new socket.
   - Parameters:
     - domain: The communication domain or address family for the socket (e.g., AF_INET for IPv4).
     - type: The type of socket (e.g., SOCK_STREAM for TCP or SOCK_DGRAM for UDP).
     - protocol: The specific protocol to be used with the socket (usually set to 0 for default).
2. bind():
   - Syntax: Bind a socket to a specific address and port.
   - Parameters:
     - sockfd: The file descriptor of the socket to be bound.
     - addr: Pointer to a sockaddr structure containing the address and port to bind to.
     - addrlen: The size of the sockaddr structure.
3. listen():
   - Syntax: Put a socket in a passive listening mode.
   - Parameters:
     - sockfd: The file descriptor of the socket to listen on.
     - backlog: The maximum length of the queue of pending connections.
4. accept():
   - Syntax: Accept an incoming connection on a listening socket.

- Parameters:
    - sockfd: The file descriptor of the listening socket.
    - addr: Pointer to a sockaddr structure where the address of the connecting client will be stored.
    - addrlen: Pointer to a variable containing the size of the sockaddr structure.

5. connect():
    - Syntax: Initiate a connection on a socket to a remote server.
    - Parameters:
        - sockfd: The file descriptor of the socket.
        - addr: Pointer to a sockaddr structure containing the address and port of the server to connect to.
        - addrlen: The size of the sockaddr structure.
6. send():
    - Syntax: Send data over a TCP socket to the connected peer.
    - Parameters:
        - sockfd: The file descriptor of the socket.
        - buf: Pointer to the data buffer containing the data to be sent.
        - len: The length of the data to be sent.
        - flags: Optional flags to modify the behavior of the send operation.
7. recv():
    - Syntax: Receive data from a TCP socket.
    - Parameters:
        - sockfd: The file descriptor of the socket.
        - buf: Pointer to the buffer where the received data will be stored.
        - len: The maximum length of the buffer to receive data.
        - flags: Optional flags to modify the behavior of the receive operation.
8. sendto():
    - Syntax: Send data over a UDP socket to a specific destination.
    - Parameters:
        - sockfd: The file descriptor of the socket.

    - buf: Pointer to the data buffer containing the data to be sent.
    - len: The length of the data to be sent.
    - flags: Optional flags to modify the behavior of the send operation.
    - dest_addr: Pointer to a sockaddr structure specifying the destination address and port.
    - addrlen: The size of the dest_addr structure.

9. recvfrom():
    - Syntax: Receive data from a UDP socket, along with the address of the sender.
    - Parameters:
        - sockfd: The file descriptor of the socket.
        - buf: Pointer to the buffer where the received data will be stored.
        - len: The maximum length of the buffer to receive data.
        - flags: Optional flags to modify the behavior of the receive operation.
        - src_addr: Pointer to a sockaddr structure where the address of the sender will be stored.
        - addrlen: Pointer to a variable containing the size of the src_addr structure. On return, it will contain the actual size of the address.