ASSIGNMENT-2

Name: Sayantan Roy

Roll Number: GCECTB-R18-3025

Subject: Computer Network Lab

Dept: CSE

Year:3rd

ASSIGNMENT -2

SOCKET PROGRAMMING

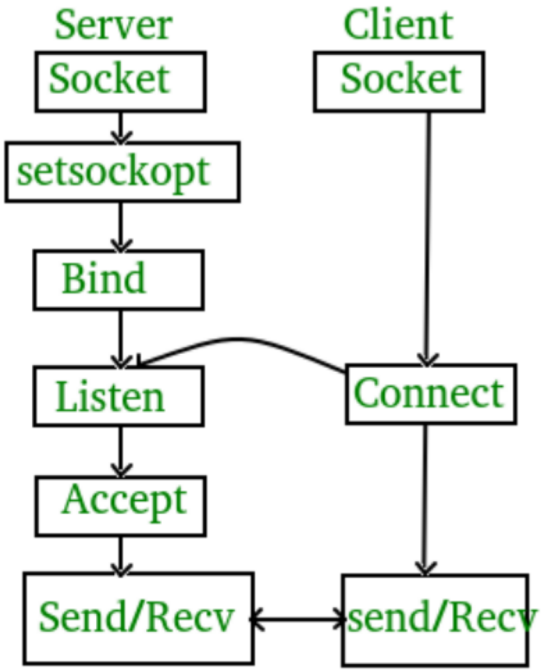
IN C

ASSIGNMENT 2

Socket Programming in C

What is socket programming?

Socket programming is a way of connecting two nodes on a network to communicate with each other. One socket(node) listens on a particular port at an IP, while other socket reaches out to the other to form a connection. Server forms the listener socket while client reaches out to the server.



Stages for server:

Socket creation:

int sockfd = socket(domain, type, protocol)

sockfd:socket descriptor, an integer (like a file-handle)
domain:integer, communication domain e.g., AF_INET (IPv4 protocol) , AF_INET6 (IPv6 protocol)

type:communication type:

SOCK_STREAM: TCP(reliable, connection oriented)

SOCK_DGRAM: UDP(unreliable, connectionless)

protocol:Protocol value for Internet Protocol(IP), which is 0. This is the same number which appears on protocol field in the IP header of a packet. (man protocols for more details)

Setsockopt:

int setsockopt(int sockfd, int level, int optname,

const void *optval, socklen_t optlen);

This helps in manipulating options for the socket referred by the file descriptor sockfd. This is completely optional, but it helps in reuse of address and port. Prevents error such as: “address already in use”.

•

•**Bind:**

```
int bind(int sockfd, const struct sockaddr *addr,
                                socklen_t addrlen);
```

After creation of the socket, bind function binds the socket to the address and port number specified in addr(custom data structure). In the example code, we bind the server to the localhost, hence we use INADDR_ANY to specify the IP address.

•**Listen:**

```
int listen(int sockfd, int backlog);
```

It puts the server socket in a passive mode, where it waits for the client to approach the server to make a connection. The backlog, defines the maximum length to which the queue of pending connections for sockfd may grow. If a connection request arrives when the queue is full, the client may receive an error with an indication of ECONNREFUSED.

Accept:

int new_socket= accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);

It extracts the first connection request on the queue of pending connections for the listening socket, sockfd, creates a new connected socket, and returns a new file descriptor referring to that socket. At this point, connection is established between client and server, and they are ready to transfer data.

Stages for Client:

- **Socket connection**: Exactly same as that of server's socket creation
- **Connect:**

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

The connect() system call connects the socket referred to by the file descriptor sockfd to the address specified by addr. Server's address and port is specified in addr.

Implementation:

Here we are exchanging on hello message between server and client to demonstrate the client/server model.

If we are creating a connection between client and server using TCP then it has few functionality like, TCP is suited for applications that require high reliability, and transmission time is relatively less critical. It is used by other protocols like HTTP, HTTPs, FTP, SMTP, Telnet. TCP rearranges data packets in the order specified. There is absolute guarantee that the data transferred remains intact and arrives in the same order in which it was sent. TCP does Flow Control and requires three packets to set up a socket connection, before any user data can be sent. TCP handles reliability and congestion control. It also does error checking and error recovery. Erroneous packets are retransmitted from the source to the destination.

HERE IS A TCP/IP CLIENT SERVER PROGRAM:

SERVERSIDE PROGRAM:

```
#include <stdio.h>
#include <netdb.h>
#include <netinet/in.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#define MAX 80
#define PORT 8080
#define SA struct sockaddr

// Function designed for chat between client and server.
void func(int sockfd)
{
	char buff[MAX];
	int n;
	// infinite loop for chat
	for (;;) {
		bzero(buff, MAX);

		// read the message from client and copy it in buffer
		read(sockfd, buff, sizeof(buff));
		// print buffer which contains the client contents
```

```
            printf("From client: %s\t To client : ", buff);
            bzero(buff, MAX);
            n = 0;
            // copy server message in the buffer
            while ((buff[n++] = getchar()) != '\n')
                ;

            // and send that buffer to client
            write(sockfd, buff, sizeof(buff));

            // if msg contains "Exit" then server exit and chat ended.
            if (strncmp("exit", buff, 4) == 0) {
                printf("Server Exit...\n");
                break;
            }
        }
}

// Driver function
int main()
{
        int sockfd, connfd, len;
        struct sockaddr_in servaddr, cli;

        // socket create and verification
        sockfd = socket(AF_INET, SOCK_STREAM, 0);
```

```
if (sockfd == -1) {
        printf("socket creation failed...\n");
        exit(0);
}
else
        printf("Socket successfully created..\n");
bzero(&servaddr, sizeof(servaddr));

// assign IP, PORT
servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
servaddr.sin_port = htons(PORT);

// Binding newly created socket to given IP and verification
if ((bind(sockfd, (SA*)&servaddr, sizeof(servaddr))) != 0) {
        printf("socket bind failed...\n");
        exit(0);
}
else
        printf("Socket successfully binded..\n");

// Now server is ready to listen and verification
if ((listen(sockfd, 5)) != 0) {
        printf("Listen failed...\n");
        exit(0);
}
```

```
        else
                printf("Server listening..\n");
        len = sizeof(cli);

        // Accept the data packet from client and verification
        connfd = accept(sockfd, (SA*)&cli, &len);
        if (connfd < 0) {
                printf("server acccept failed...\n");
                exit(0);
        }
        else
                printf("server acccept the client...\n");

        // Function for chatting between client and server
        func(connfd);

        // After chatting close the socket
        close(sockfd);
}
```

TCP CLIENT SIDE PROGRAM:

```
#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#define MAX 80
```

```
#define PORT 8080
#define SA struct sockaddr
void func(int sockfd)
{
	char buff[MAX];
	int n;
	for (;;) {
		bzero(buff, sizeof(buff));
		printf("Enter the string : ");
		n = 0;
		while ((buff[n++] = getchar()) != '\n')
			;
		write(sockfd, buff, sizeof(buff));
		bzero(buff, sizeof(buff));
		read(sockfd, buff, sizeof(buff));
		printf("From Server : %s", buff);
		if ((strncmp(buff, "exit", 4)) == 0) {
			printf("Client Exit...\n");
			break;
		}
	}
}

int main()
{
	int sockfd, connfd;
```

```
struct sockaddr_in servaddr, cli;

// socket create and varification
sockfd = socket(AF_INET, SOCK_STREAM, 0);
if (sockfd == -1) {
        printf("socket creation failed...\n");
        exit(0);
}
else
        printf("Socket successfully created..\n");
bzero(&servaddr, sizeof(servaddr));

// assign IP, PORT
servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr = inet_addr("127.0.0.1");
servaddr.sin_port = htons(PORT);

// connect the client socket to server socket
if (connect(sockfd, (SA*)&servaddr, sizeof(servaddr)) != 0) {
        printf("connection with the server failed...\n");
        exit(0);
}
else
        printf("connected to the server..\n");

// function for chat
```

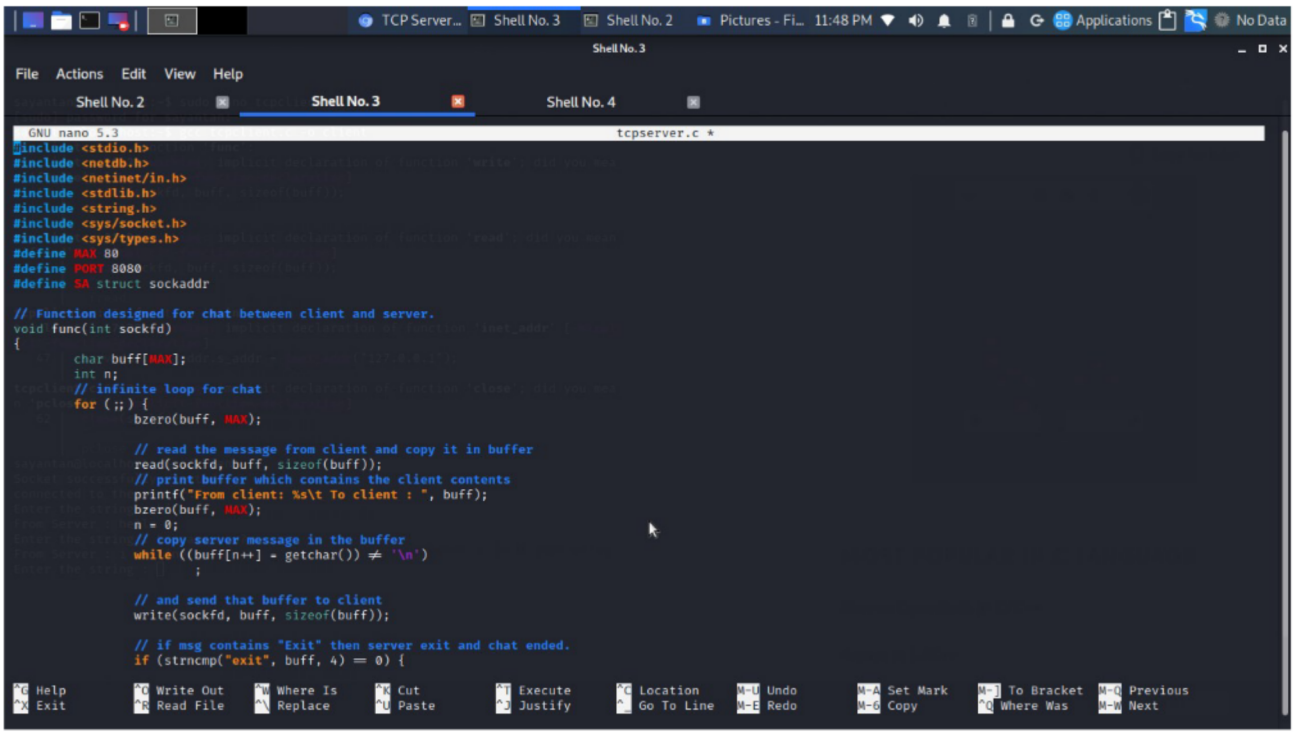
```
        func(sockfd);


        // close the socket
        close(sockfd);
}
```

Compilation –
Server side:
gcc server.c -o server
./server

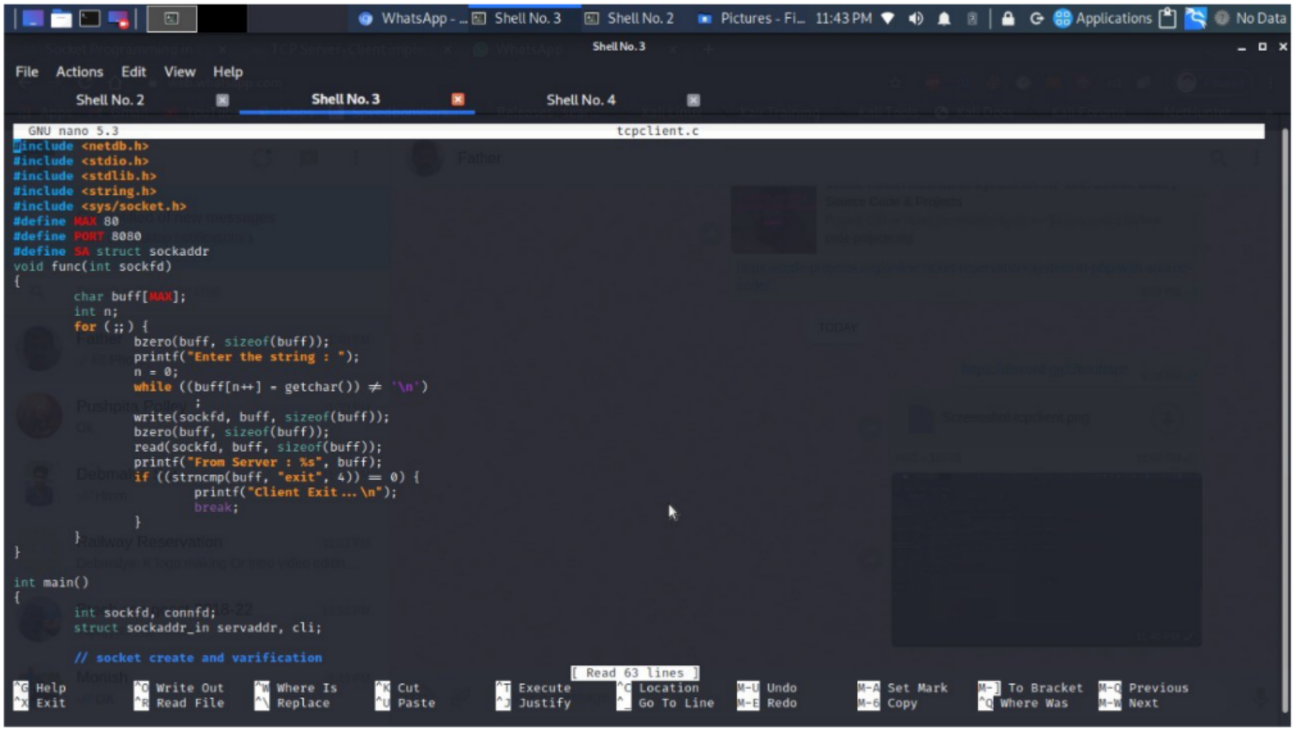
Client side:
gcc client.c -o client
./client

Output –
Server side:
Socket successfully created..

```
Socket successfully binded..
Server listening..
server acccept the client...
From client: hi
     To client : hello
From client: exit
     To client : exit
Server Exit...
```

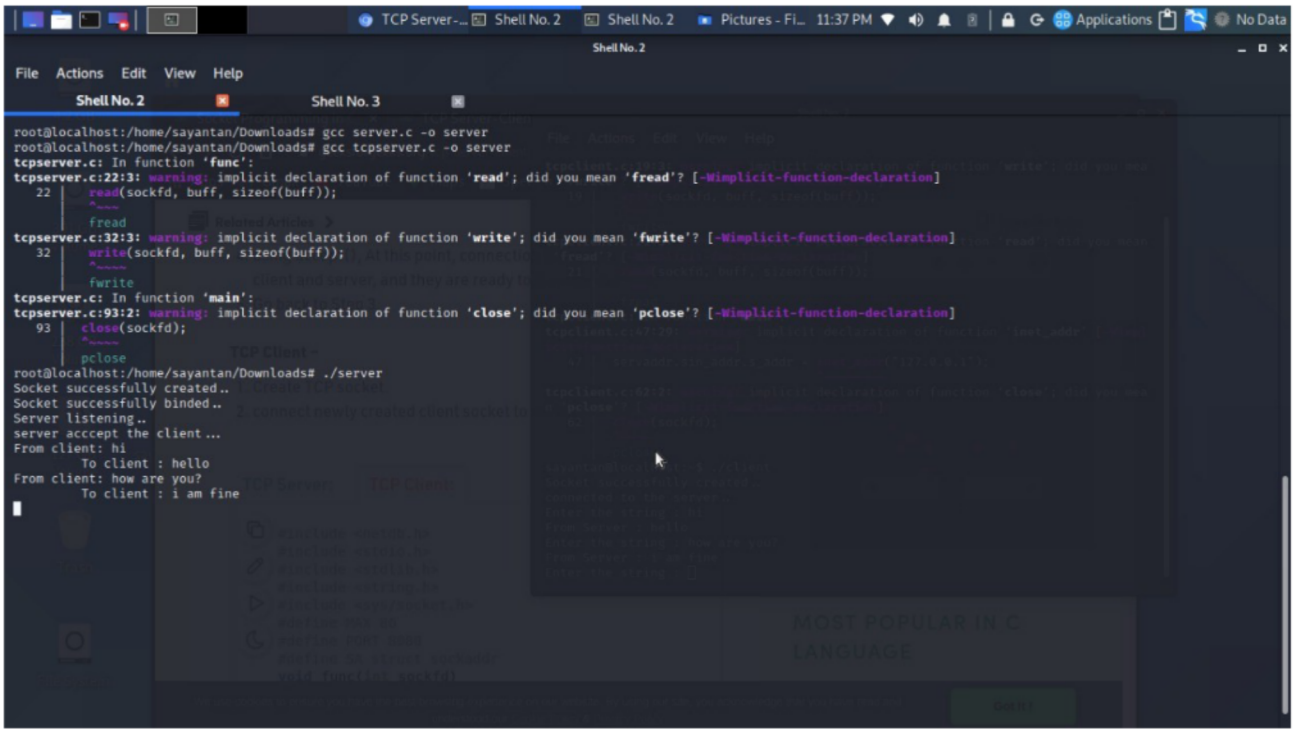


TCPSERVER.C



TCPCLIENT.C

TCP SERVER WORKING:



TCP CLIENT WORKING: