NATIONAL INSTITUTE OF TECHNOLOGY SILCHAR

Cachar, Assam

B.Tech. Vth Sem

Subject Code: CS-311

Subject Name: Computer Network Laboratory

Submitted By:

Name : Subhojit Ghimire

Sch. Id. : 1912160

Branch : CSE – B

Q.1.: Write a Client-Server Socket Program to implement "TCP Client Server" architecture. (Description: Retrieve information (any text or system info) from TCP Server to TCP Client.

AIM: TO IMPLEMENT "TCP CLIENT SERVER" ARCHITECTURE - C++

THEORY:

1. SOCKET: It is an Application Programming Interface (API) used for InterProcess Communications (IPC), a well-defined method of connecting two processes locally or across a network.
2. CLIENT-SERVER SOCKET: The server creates a socket, attaches it to a network port addresses, then waits for the client to contact it. The client-to-server data exchange takes place when a client connects to the server through a socket. The server performs the client's request and sends the reply back to the client.
3. TCP CLIENT SERVER: TCP (Transmission Control Protocol) is a transport layer in a networking service. The client in TCP/IP connection is the device that dials the phone and the server is the device that is listening in for calls to come in.

The entire process can be broken down as:

TCP SERVER: (i) Create TCP Socket

(ii) Bind the Socket to server address

(iii) Put the server socket in passive mode and wait for the client to approach.

(iv) When the connection is established, transfe data.

TCP CLIENT: (i) Create TCP Socket
(ii) Connect newly created client socket to server.
(iii) When the connection is established, receive data

CODE:

```
// TCP SERVER

# include <iostream>
# include <cstdlib>
# include <cstring>
# include <netdb.h>
# include <sys/socket.h>
# include <unistd.h>
// # include <netinet/in.h>

#define MAX 80
#define PORT 8080

using namespace std;

void errorExit (int errorSignal) {
    cout << "FAIL: ";
    if (errorSignal == 1)  cout << "SOCKET CREATING\n";
    else if (errorSignal == 2)  cout << "SOCKET BINDING\n";
    else if (errorSignal == 3)  cout << "SERVER LISTENING\n";
    else if (errorSignal == 4)  cout << "SERVER ACCEPTING\n";
    exit (0);
}
```

```
int main () {
    // Socket Creating
    int sockfd = socket (AF_INET, SOCK_STREAM, 0);
    Struct Sockaddr_in servaddr;
    if (sockfd != -1)
            cout << "SUCCESS: SOCKET CREATED" << endl;
    else
            errorExit (1);
    bzero (&servaddr, sizeof (servaddr));

    // Assigning IP and PORT
     servaddr.sin_family = AF_INET;
     servaddr.sin_addr.s_addr = htonl (INADDR_ANY);
     // [illegible] & servaddr.sin_addr.s_addr = inet_addr ("127.0.0.1");

    // BINDING NEWLY CREATED SOCKET TO GIVEN IP
    if (!(bind (sockfd, (struct sockaddr*)&servaddr,
                        sizeof (servaddr))))
            cout << "SUCCESS: SOCKET BINDED" << endl;
    else
            errorExit (2);

    // Server Ready to Listen
    struct Sockaddr_in cli;
    unsigned int len = sizeof (cli);
    if (!(listen (sockfd, 5)))
            cout << "SUCCESS: SERVER LISTENING" << endl;
    else
            errorExit (3);
```

```
// ACCEPTING THE DATA PACKET FROM CLIENT
int connfd = accept (sockfd, (struct sockaddr *)&cli,
                                        &len);
if ( connfd >= 0)
        cout << "SUCCESS : CLIENT ACCEPTED "<<endl;
else
        errorExit (4);

// MESSAGE EXCHANGE BETWEEN CLIENT AND SERVER
char buff [MAX];
while (true) {
        bzero (buff, MAX);
        read (connfd, buff, sizeof (buff));
        cout << "CLIENT MESSAGE:  "<<buff;
        bzero (buff, MAX);

        // write server message and send to client
        cout << "WRITE SERVER MESSAGE:  ";
        for (int ii= 0; (buff [ii] = getchar ()) != '\n'; ++ii);
        write (connfd, buff, sizeof (buff));
        if (!strncmp ("exit", buff, 4))
                break;
}
cout << "SERVER EXIT " << endl;
close (sockfd);
return 0;
}
```

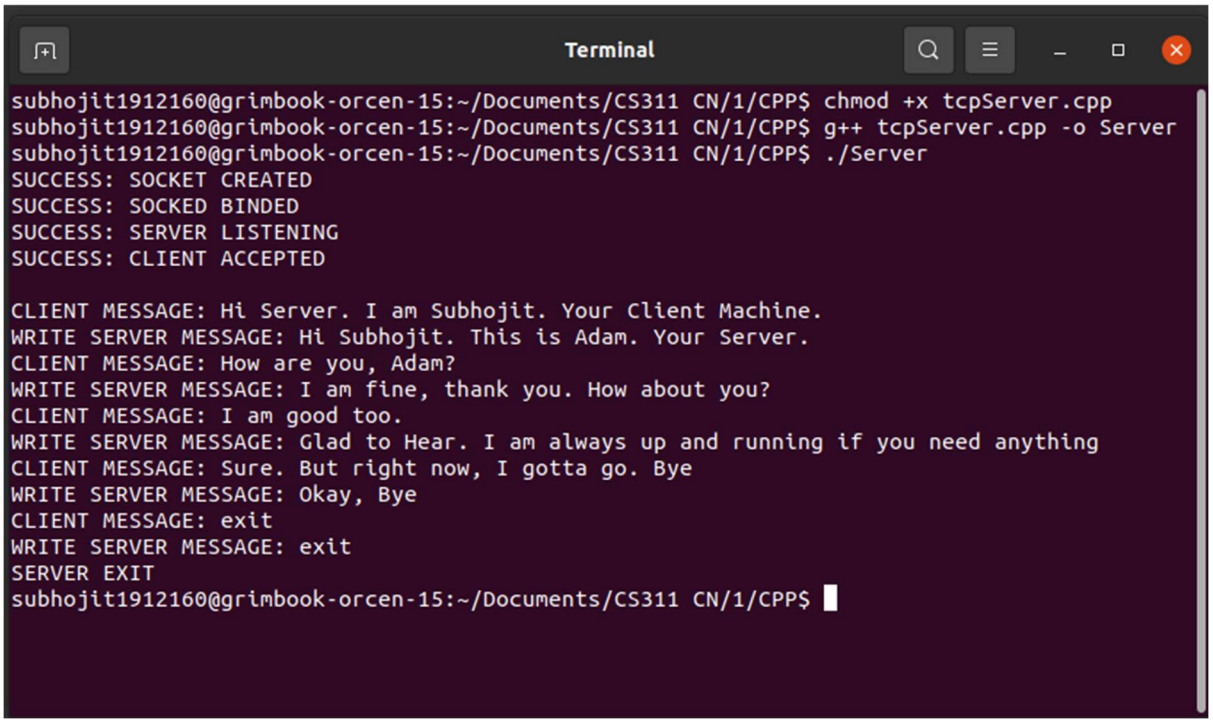
// TCP CLIENT

```
#include <iostream>
#include <cstdlib>
#include <cstring>
#include <netdb.h>
#include <unistd.h>
#include <sys/socket.h>
#include <arpa/inet.h>

#define MAX 80
#define PORT 8080

using namespace std;

void errorExit (int errorSignal) {
        cout << "FAIL: ";
        if (errorSignal == 1)   cout << "SOCKET CREATING\n";
        else if (errorSignal ==2)   cout<< "SERVER CONNECTION\n";
        exit (0);
}

int main () {
        // SOCKET CREATION
        int sockfd = socket (AF_INE, SOCK_STREAM, 0);
        struct sockaddr_in servaddr;
        if (sockfd != -1)   cout << "SUCCESS: SOCKET CREATED\n";
        else errorExit (1);
        bzero (&servaddr, sizeof(servaddr));
```

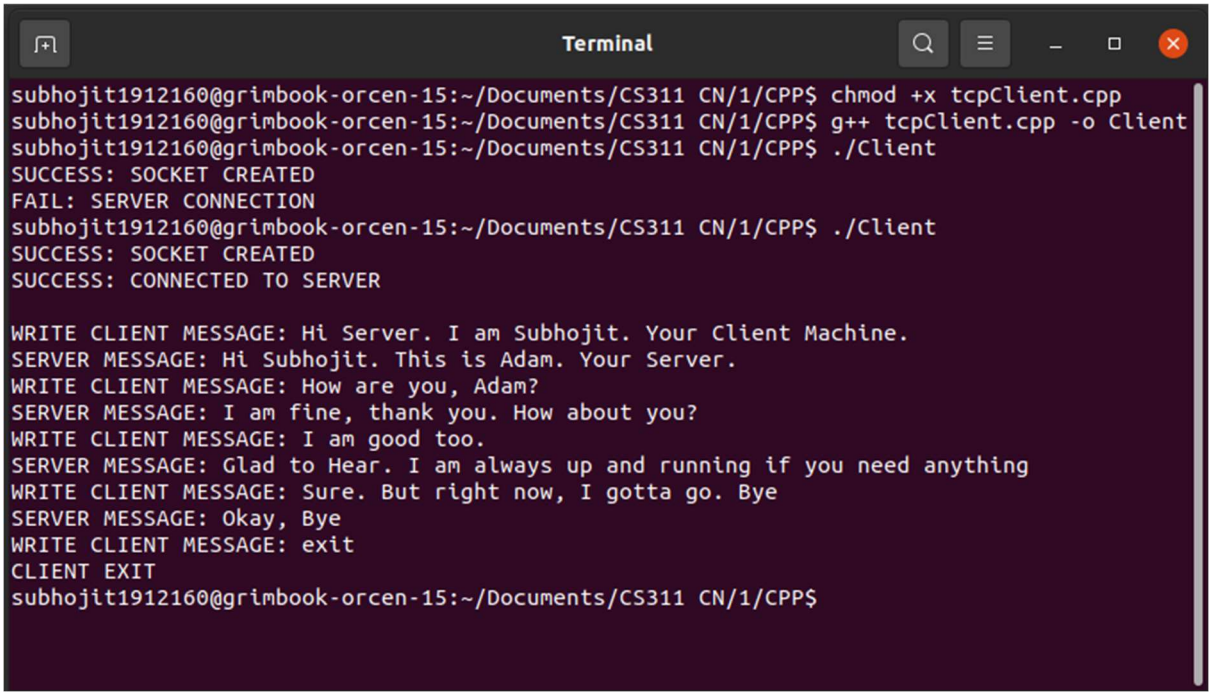
```
// ASSIGNING IP AND PORT
serv2addr.sin-family    = AF_INET;
serv2addr.sin-2ddr.s-2ddr = htonl (INADDR_ANY);
// serv2addr.sin-2ddr.s-2ddr = inet-2ddr ("127.0.0.1");
serv2addr.sin-port    = htons (PORT);

// CONNECTING TO SERVER SOCKET
if (! (connect (sockfd, (struct sock2addr*)&serv2addr,
                                    sizeof (serv2addr))))
        cout << " SUCCESS : CONNECTED TO SERVER \n";
else
        errorExit (2);

// MESSAGE EXCHANGE BETWEEN CLIENT AND SERVER.
char buff [MAX];
while (1) {
        bzero (buff, sizeof (buff));
        cout << " WRITE CLIENT MESSAGE :  ";
        for (int ii=0; (buff[ii] = getchar ()) != '\n'; ++ii);
        write (sockfd, buff, sizeof (buff));
        if (!strncmp (buff, "exit", 4));
                break;
        bzero (buff, sizeof (buff));
        read (sockfd, buff, sizeof (buff));
        cout << " SERVER MESSAGE :  " << buff;
}
cout << " CLIENT  EXIT \n";
close (sockfd);
return 0;
}
```

OUTPUT:

// TCP SERVER



// TCP CLIENT



Output Explanation:

Firstly, compile and run the TCP Server Program. This way, a server is established for any or all created clients to connect to. If there is no server, Client will fail to connect to any server. After running Server, compile and run the TCP Client Program. The created client will connect to the server. After the connection is successfully established between client and server, message exchange can take place. The messages sent by client will be visible to server, and the server can reply back with its own messages.

Q.2.: Write a Client-server Socket Program to implement "TCP Chat Server". (Description: Two client system connected to one central server for connection establishment, communication / chat have to be done in client machines.)

AIM: To IMPLEMENT "TCP CHAT SERVER" USING C++

THEORY:

1. SOCKET: It is an Application Programming Interface (API) used for InterProcess Communications (IPC), a well-defined method of connecting two processes locally or across a network.
2. CLIENT-SERVER SOCKET: The server creates a socket, attaches it to a network port addresses, then waits for the client to contact it. The client-to-server data exchange takes place when a client connects to the server through a socket. The server performs the client's request and sends the reply back to the client.
3. TCP CHAT SERVER: Even the widely used messaging application like Snapchat, WhatsApp, Twitter, Instagram etc. use transmission control protocol. In TCP based chats, the exchanging of messages take place using TCP/IP connection layer. There is a secure server that is open to all the clients for connection. Two or more clients can chat with one another by passing message through this server.

The entire process can be broken down as:

TCP SERVER: (i) Create TCP Socket.

(ii) Bind the socket to server address.

(iii) Put the server on passive mode. It will wait for the client to approach the server to make a connection.

(iv) When a connection is established between client and server, wait for client to send message.

(v) Transmit any or all messages received by the server to all the clients currently connected to the server.

TCP CLIENT: (i) Create TCP Socket.

(ii) Connect newly created client socket to server.

(iii) When the connection is established, send and receive data.

CODE:

// TCP CHAT SERVER

```
#include <iostream>
#include <netdb.h>
#include <cstdlib>
#include <cstring>
#include <sys/socket.h>
#include <pthread.h>
#include <unistd.h>
```

```
#define PORT 8080

pthread_mutex_t mutex;
int clients[20];
int nn = 0;

using namespace std;

void sendToAll (char *msg, int curr) {
        pthread_mutex_lock (&mutex);
        for (int ii = 0; ii<nn; ++ii)
                if (send (clients[ii], msg, strlen(msg),0)
                                < 0) {
                        cout<<" FAIL: MESSAGE SEND\n";
                        continue;
                }
        pthread_mutex_unlock (&mutex);
}

void *recvMsg (void *connfd) {
        int sockfd = *((int *) connfd);
        char msg[1000];
        int len;
        while ((len =recv (sockfd, msg, 1000, 0))>0){
                msg[len] = '\0';
                sendToAll (msg, sockfd);
        }
        return 0;
}
```

```
void errorExit (int errorSignal) {
        cout << "FAIL: ";
        if (errorSignal == 1)
            cout << "SOCKET CREATING \n";
        else if (errorSignal == 2)
            cout << " SOCKET BINDING \n";
        else if (errorSignal == 3)
            cout << " SERVER LISTENING \n";
        else if (errorSignal == 4)
            cout << " SERVER ACCEPTING \n";
        exit (0);
}

int main() {
    // SOCKET CREATION
    int sockfd = socket (AF_INET, SOCK_STREAM, 0);
    struct sockaddr_in servaddr;
    if (sockfd != -1)
        cout << "SUCCESS: SOCKET CREATED \n";
    else
        errorExit (1);
    bzero (&servaddr, sizeof (servaddr));

    // ASSIGNING IP AND PORT
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = htonl (INADDR_ANY);
    // servaddr.sin_addr.s_addr = inet_addr ("127.0.0.1");
    servaddr.sin_port = htons (PORT);
```

```
// Binding NEWLY CREATED SOCKET TO GIVEN IP
if (!(bind (sockfd, (struct sockaddr*) &servaddr,
                    sizeof (servaddr))))
        cout << " SUCCESS: SERVER BINDED \n";
else
        errorExit (2);

// Server ready to listen
if (!listen (sockfd, 20))
        cout<<" SUCCESS: SERVER LISTENING "<<endl;
else    errorExit (3);

// Accepting CLIENTS
pthread_t recvt;
int connfd;
while (1) {
        if ((connfd = accept (sockfd, (struct sockaddr*)
                        NULL, NULL)) >=0) {
                pthread_mutex_lock (&mutex);
                clients [nn++] = connfd;
                // Creating Thread for each Client
                pthread_create (&recvt, NULL, &recvMsg, &connfd);
                pthread_mutex_unlock (&mutex);
        }
        else    errorExit (4);
}
close (sockfd);
return 0;
}
```

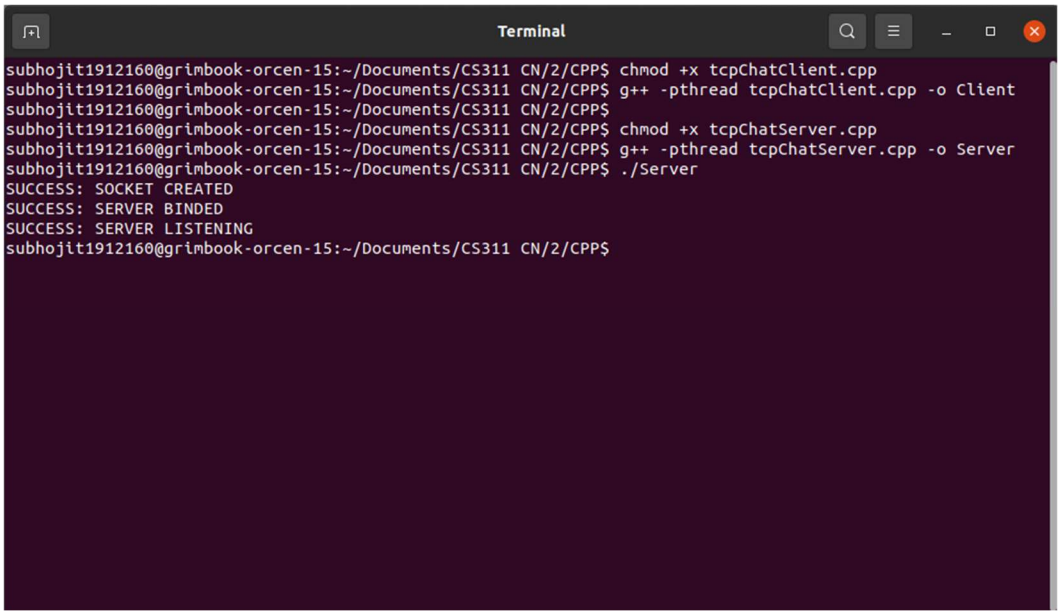
```
//TCP CHAT CLIENT

# include <iostream>
# include <cstdlib>
# include <cstring>
# include <netdb.h>
# include <arpa/inet.h>
# include <pthread.h>
# include <sys/socket.h>
# include <unistd.h>

# define PORT 8080
char msg[1000];

using namespace std;

void *recvMsg (void *connfd) {
        int sockfd = *((int *) connfd);
        int len;
        // Client Thread Always Ready to Get Message
        while ((len = recv (sockfd, msg, 1000, 0))>0){
                msg [len] = '\0';
                fputs (msg, stdout);
        }
        return 0;
}
```

```
void errorExit (int errorSignal) {
        cout << "FAIL: ";
        if ( errorSignal == 1)
                cout << "SOCKET CREATING\n";
        else if (errorSignal == 2)
                cout <<" SERVER CONNECTION\n";
        exit (0);
}

int main ( int argc , char *argv[ ]) {
        // Socket creation
        int sockfd = socket (AF_INET, SOCK_STREAM, 0);
        struct sockaddr_in servaddr;
        if (sockfd != -1)
                cout << "SUCCESS : SOCKET CREATED\n";
        else
                errorExit (1);
        bzero (&servaddr, sizeof (servaddr));

        // Assigning IP and PORT
        servaddr.sin_family = AF_INET;
        servaddr.sin_addr.s_addr = htonl (INADDR_ANY);
        // servaddr.sin_addr.s_addr = inet_addr ("127.0.0.1");
        servaddr.sin_port = htons (PORT);

        // Connecting to Server Socket
        if (!(connect (sockfd, (struct sockaddr*)&servaddr,
                                sizeof (servaddr))))
                cout << "SUCCESS: CONNECTET TO SERVER\n";
        else errorExit (2);
```

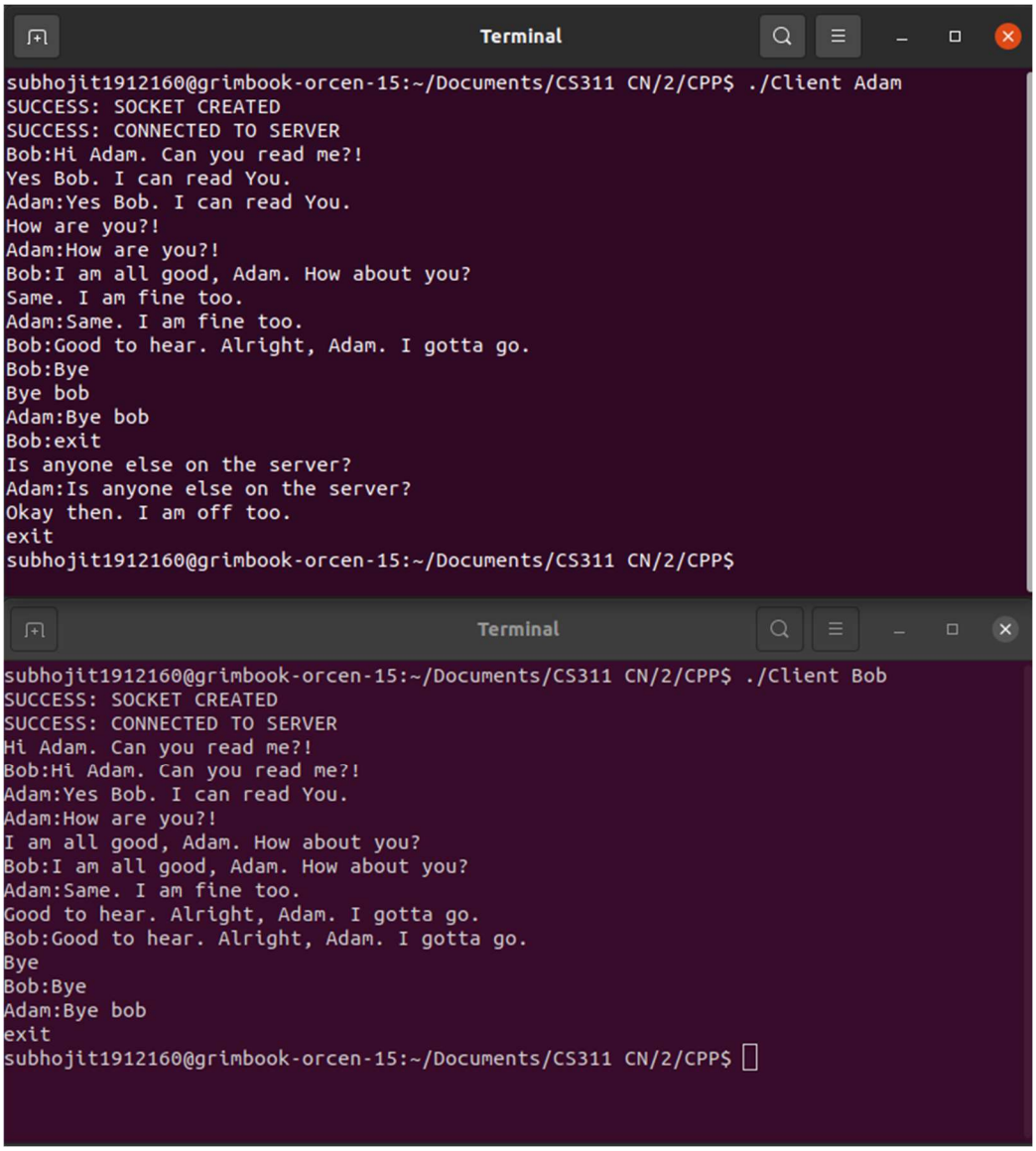
```
// Create client thread always ready for message receive
pthread_t recvt;
pthread_create (&recvt, NULL, &recvMsg, &sockfd);

// READY TO RECEIVE MESSAGE FROM CONSOLE
int len;
char sendMsg[1000], clientName[100];
strcpy (clientName, argv[1]);
while (fgets (msg, 500, stdin) > 0) {
        strcpy (sendMsg, clientName);
        strcat (sendMsg, ":");
        strcat (sendMsg, msg);
        len = write (sockfd, sendMsg, strlen(sendMsg));
        if (len < 0)
                cout << "FAIL: MESSAGE SEND\n";
        if (!strncmp ("exit", msg, 4))
                break;
}

// pthread_join (recvt, NULL);
close (sockfd);
return 0;
}
```

OUTPUT:

// TCP CHAT SERVER



// TCP CHAT CLIENT



OUTPUT EXPLANATION:

Firstly, the TCP Server Program is compiled and run. After the server is up and running, any or all clients created will be able to connect to the server. In TCP based Chat server, the server cannot, or rather, does not communicate with the clients. The clients communicate with one another. The server just acts as a secure medium to transmit the messages from one client to another. So, once the server is created, the TCP Client Program is now compiled and run. As many clients (or as many as the server allows) can run the compiled file, or the application file, and connect to the server and send messages to all the online clients , as well as receive the messages sent by other clients on the same server. After sometime, when the server overloads, it exits automatically, disconnecting everyone.