PROJECT REPORT

ON

Chat Server using Socket Programing

REPORT SUBMITTED

TO

VISHWAKARMA INSTITUTE OF INFORMATION TECHNOLOGY, PUNE

FOR THE PBL OF PYTHON FOR ENGINEERS

IN

ENGINEERING AND APPLIED SCIENCE DEPARTMENT

BY:

1	**Vaibhav Dadarao Kokate**	**9016**
2	**Yash Romil Shah**	**9020**
3	**Khushal Anand Malu**	**9021**
4	**Shrish Mohan Dollin**	**9022**
5	**Yash Jagannath Sant**	**9023**

Class: First Year **Division: I** **Batch: I1**

Batch Teacher:
Kalpana Kumbhar

INDEX

ABSTRACT:

Socket Programming! It was a new field and had not been explored lately by our colleagues! This was the inspiration to choose the topic and work on it!

We have used many functions and created a server on Linux and a client side for windows as well as Linux. We have also tried our platform by joining to the server individually and chatting there and guess what it was fun! The joy of chatting on our self-created chat server! So basically, the server needs to be connected! Then the client can enter the IP Port and his preferred username and easily access the server to chat with as many as people currently active!

So basically, this is just a prototype as the user needs the code to be available on this device and compile the code by himself! Moving forward apart from the PBL, this can we developed as an extensive chat room as "Messenger"!

Overall, it was a great learning process!

INTRODUCTION AND THEORY:

TCP (Transmission control protocol)

A TCP (transmission control protocol) is a connection-oriented communication. It is an intermediate layer of the application layer and internet protocol layer in the OSI model. TCP is designed to send the data packets over the network. It ensures that data is delivered to the correct destination.

TCP creates a connection between the source and destination node before transmitting the data and keeps the connection alive until the communication is active.

In TCP before sending the data it breaks the large data into smaller packets and cares the integrity of the data at the time of reassembling at the destination node. Major Internet applications such as the World Wide Web, email, remote administration, and file transfer rely on TCP.

TCP also offers the facility of retransmission, when a TCP client sends data to the server, it requires an acknowledgment in return. If an acknowledgment is not received, after a certain amount of time transmitted data will be loss and TCP automatically retransmits the data.

The communication over the network in TCP/IP model takes place in form of a client-server architecture. ie, the client begins the communication and establishes a connection with a server.

the general procedure of establishing a TCP binding connection is:

1. The server process issues an OPEN command to a TCP device.
2. The server process issues a USE command, followed by a READ command, awaiting input from the client process. The server must be listening before a client can establish a connection. The initial READ command completes when the client has opened the connection and sent some data. You can include the “A” mode parameter in the OPEN command to make the initial READ complete as soon as the server accepts the connection.
3. The client process issues an OPEN command that specifies the TCP device to which it is connecting.

4. The client process issues a USE command followed by a WRITE command to complete the connection. Inter Systems IRIS copies all characters in the WRITE command(s) to a buffer.
5. After the server has read the characters that the client sent in its first WRITE command, both sides can continue to issue READ and WRITE commands. There is no further restriction on the order of these commands to the same port.
6. Either side can initiate the closing of a connection with the CLOSE or HALT command. Closing the client side first is preferable. If the server needs to disconnect so that it can accept a connection from another client process

A static IP address is simply an address that doesn't change. Once your device is assigned a static IP address, that number typically stays the same until the device is decommissioned or your network architecture changes. Static IP addresses generally are used by servers or other important equipment.
Static IP addresses are assigned by Internet Service Providers (ISPs). Your ISP may or may not allocate you a static IP address depending on the nature of your service agreement. We describe your options a little later, but for now assume that a static IP address adds to the cost of your ISP contract.

Dynamic IP addresses are subject to change, sometimes at a moment's notice. Dynamic addresses are assigned, as needed, by Dynamic Host Configuration Protocol (DHCP) servers.
We use dynamic addresses because IPv4 doesn't provide enough static IP addresses to go around. So, for example, a hotel probably has a static IP address, but each individual device within its rooms would have a dynamic IP address.

A *multi-treaded server* is one that uses a separate thread of execution for each client.

The primary advantage of using multiple threads is that the code for each thread need only worry about one client. If a blocking system call is needed to serve the client, no problem is created because it only affects one client. The code for threaded servers tends to be fairly easy to follow.

The most obvious complexity of the multi-threaded approach comes from dealing with data that is shared between multiple threads. This data must be protected with locks and associated synchronization algorithms. These synchronization algorithms can be more complex than one would hope, but there are known solutions to most any synchronization problem. The most difficult chore for the programmer is just to determine which of the generic problems with known solutions can be used to solve the task at hand. In general, once the synchronization issues are resolved, multi-threaded techniques have better potential than asynchronous attempts to scale to larger, more complex problems.

Another challenge of multi-threaded network programming stems from the fact that, as was previously mentioned, the default behavior of the socket.recv() function is to block (not return) until either data or a disconnect is received on the socket. For applications, such as web servers where the clients do not interact with each other, this is not a problem at all. When a connection is received, a child thread is started and it waits on the socket.recv() call until data is received and then it collects the response needed, perhaps via a data base query, and returns the results to the client. However, in a multi-party chat server, the thread must not only watch the socket for new data, but must also pay watch for new messages from other users that need to be sent to the client. Thus, the socket.recv() calls can not block indefinitely.

In a pure threads based solution, the only way to watch a socket and other data at more or less the same time is to set a timeout on the socket such that after a short time with no data received, a socket.timeout event will be raised. The main disadvantage of this approach is that users are delayed waiting for the timeout before being sent a new message. However, if the timeout value, is kept fairly small, this delay will not generally be noticed by users. However, the performance is perhaps not as good as we would like because of repeated checking for messages after each timeout.

Socket timeouts can also be used with client applications. In this case, the application must be ready to both receive data from the server and to send data, which the local user entered. Since the client only manages one socket, the use of timeouts to facilitate the multi-tasking is a good solution.

Since the objective of this activity was to develop a multi-threaded chat server, the approach of using timeouts was also adopted for the initial implementation of the server. I decided that a pretty clean approach would be to hold a global message queue that all threads can write to and read from per the synchronization mechanisms of the readers and writers problem. So grabbing an operating systems text book, [nutt] which I knew had a solution to readers and writers, I had a synchronization solution. When I first opened an operating systems text book and saw all of the locks, (actually, they used semaphores, but in this case, those semaphores can be switched out with Python’s simple Lock object) I was a little overwhelmed for a minute and wondered if I really needed so many locks. Then I remembered that readers and writers has been worked on by lots of really smart people and that it has proven itself to be safe and free of deadlocks.

ALGORITHM:

//SERVER

STEP 1: START

STEP 2: ENTER IP ADRESS & PORT NUMBER FOR ALL INCOMING CONNECTIONS

STEP 3: LOAD THE MACHINE AND CONNECT THE CLIENT

STEP 4: WE CAN CONNECT AS MANY AS NUMBER OF CLIENTS WITH THE HELP OF THREADING

STEP 5: START LISTENING THE CLIENT IF CONNECTION IS NOT ESTABLISHED TRY TO CONNECT AGAIN

STEP 6: WAIT FOR THE MESSAGE THE CLIENT SENDS

STEP 7: PARSE THE MESSAGE WHICH CLIENT SENDS

STEP 8: SERVER GETS THE MEMBERS WHO JOINES WITH IP ADDRESS

STEP 9: ALL THREADS ARE JOINED GETS NOTIFIED

STEP 10: CLIENTS CAN CHAT FROM ANY LOCATION TILL SERVER IS ON

STEP 11: SERVER AS GETS DISCONNECTED ALL CHATS ARE STOPPED AND FURTHER CLIENTS CAN'T CHAT

STEP 12: STOP

//CLIENT

STEP 1: START

STEP 2: SOCKET CONNECTION

STEP 3: CONNECT THE SOCKET

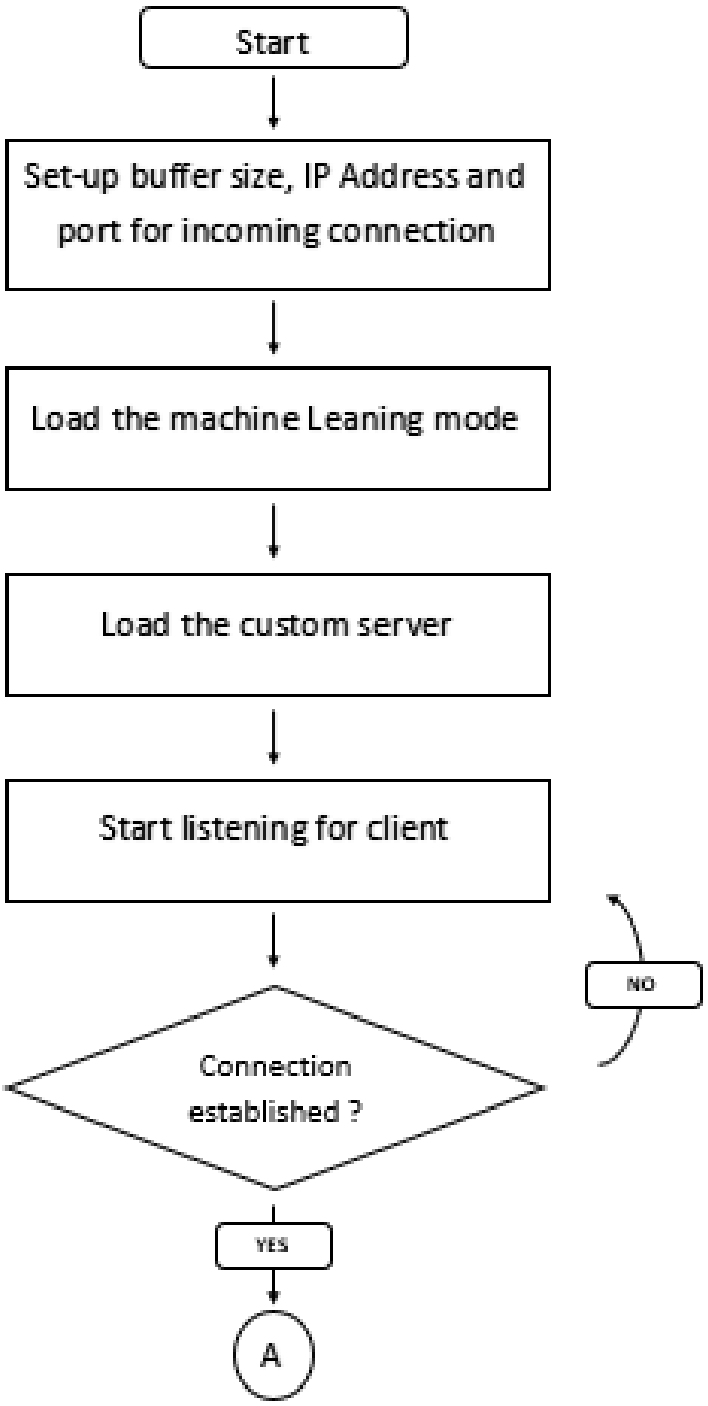
STEP 4: SEND & RECEIVE MESSAGES ON THREADING

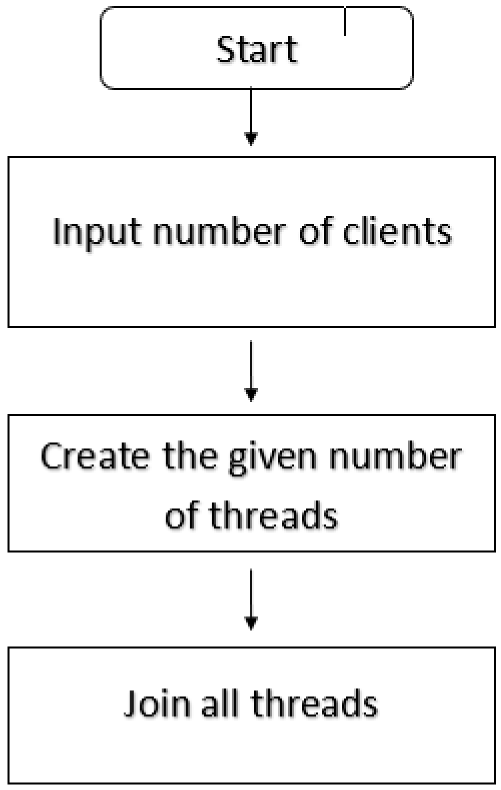
STEP 5: SHUTDOWN THE SOCKET

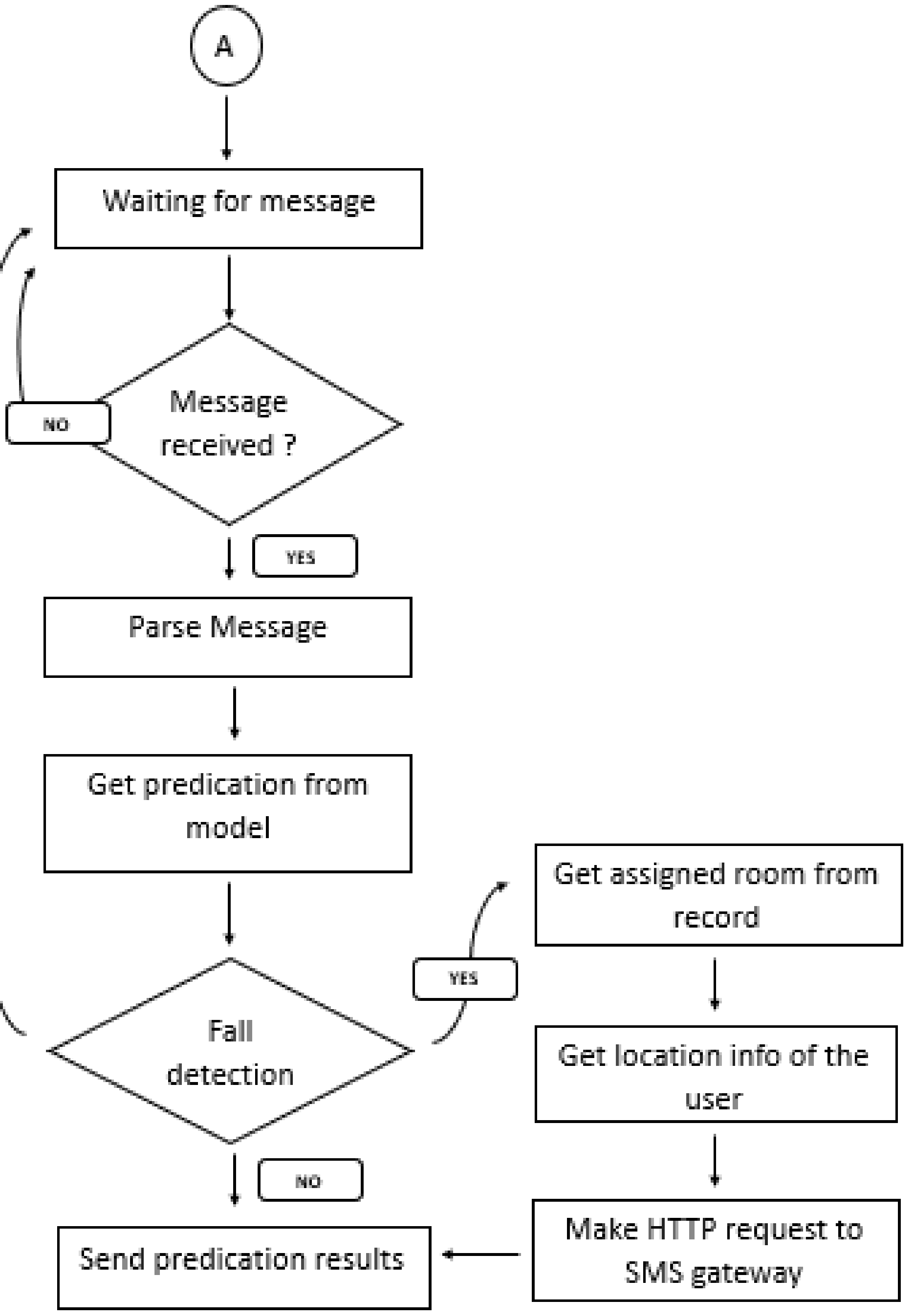
STEP 6: STOP

FLOWCHART:

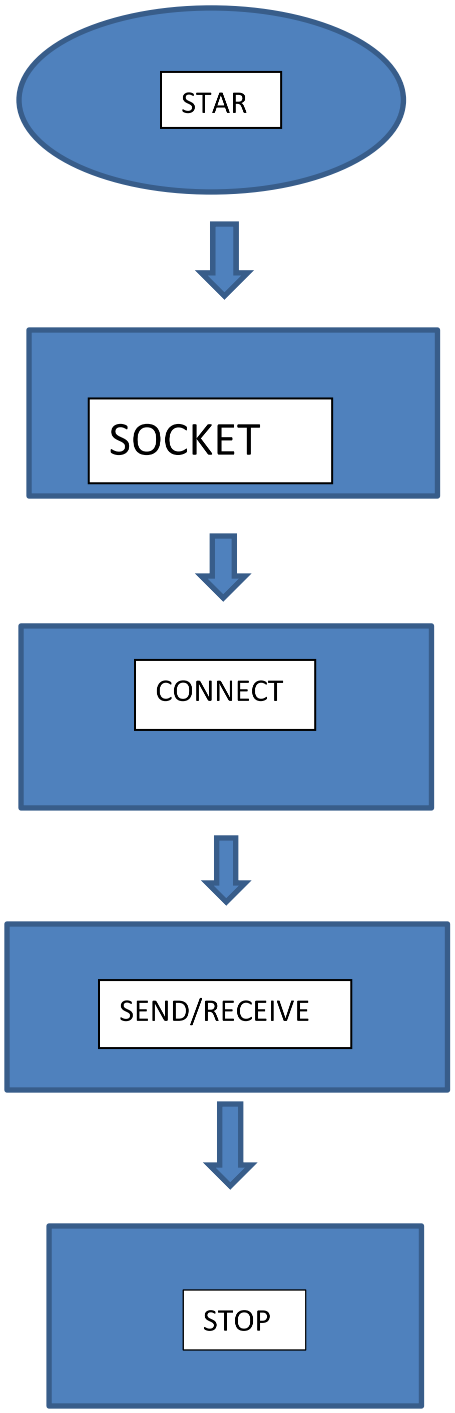
// SERVER


Start
Set-up buffer size, IP Address and port for incoming connection
Load the machine Leaning mode
Load the custom server
Start listening for client
Connection established ?
NO
YES
A


Start
Input number of clients
Create the given number of threads
Join all threads


A
Waiting for message
Message received ?
NO
YES
Parse Message
Get predication from model
Fall detection
YES
NO
Get assigned room from record
Get location info of the user
Make HTTP request to SMS gateway
Send predication results

// CLIENT



PROGRAM:

// SERVER

```
// Includes

#include <iostream>
#include <sys/types.h> //https://pubs.opengroup.org/onlinepubs/009695399/b
asedefs/sys/types.h.html
#include <sys/socket.h> //https://pubs.opengroup.org/onlinepubs/7908799/xn
s/syssocket.h.html
#include <arpa/inet.h> //https://pubs.opengroup.org/onlinepubs/7908799/xns
/arpainet.h.html
#include <netdb.h> //https://pubs.opengroup.org/onlinepubs/7908799/xns/net
db.h.html
#include <unistd.h> //https://pubs.opengroup.org/onlinepubs/7908799/xsh/un
istd.h.html
#include <string>
#include <cstring> //https://www.cplusplus.com/reference/cstring/
#include <netinet/in.h> //https://pubs.opengroup.org/onlinepubs/009695399/
basedefs/netinet/in.h.html
#include <vector> //https://www.cplusplus.com/reference/vector/vector/vect
or/
#include <pthread.h> //https://pubs.opengroup.org/onlinepubs/7908799/xsh/p
thread.h.html
#include <set> //https://www.cplusplus.com/reference/set/set/set/

#define BACKLOG 127

using namespace std;

// Declareing Variables

int k;

// structing arguments for clients
struct args {
    int *k;
    int *clients;
    string *clientNames;
    int src;
};

// const char
```

```
char *serverName;
char *port;

// creating a set for clients
set<int> excludedList;

// Function to Send message to client

void sendToClients (int *k, int *clients, string *clientNames, int src, co
nst char *msg, int flag = 0) {
    if (excludedList.count(src))
    return;

    for (int i = 0; i < *k; ++i) {
        if (excludedList.count(i))
        continue;
        if (i == src)
        continue;

        // Sending messages to all clients
        string txtToSend = string(msg, 0, strlen(msg));
        if (flag == 0)
        txtToSend = clientNames[src] + ": " + txtToSend;

        send(clients[i], txtToSend.c_str(), txtToSend.length() + 1, 0); //
https://pubs.opengroup.org/onlinepubs/7908799/xns/send.html
    }

    //Greetings when joined
    if (flag == 1) {
        string greet = "Welcome to Server \"" + string(serverName, 0, strl
en(serverName)) + "\", " + clientNames[src];
        send(clients[src], greet.c_str(), greet.length() + 1, 0);
    }
}

// Fucntion to Read message from server

void *ReadFromClient(void *p) {
    args *myArgs = ((args *) p);
    int *k = myArgs->k;
    int *clients = myArgs->clients;
    string *clientNames = myArgs->clientNames;
    int src = myArgs->src;
    if (excludedList.count(src) == 0) {
        char buff[256]{0};

        while (1) {
            memset(buff, 0, sizeof buff);
            int rbSize = recv(clients[src], buff, sizeof buff, 0);

            // if any client got disconneted
            if (rbSize <= 0) {
                string prompt = "\"" + clientNames[src] + "\" disconnected
 from the server";
                cout << prompt << "\n";
                sendToClients(k, clients, clientNames, src, prompt.c_str()
, 2);
                excludedList.insert(src);
                close(clients[src]);
                break;
            }
            sendToClients(k, clients, clientNames, src, buff);
        }
    }
    pthread_exit(0);
}

// Main

int main(int argc, char **argv) {

    // const arg
    serverName = argv[1];
    port = argv[2];

    // creating addrinfo (values)
    addrinfo hints, *res;

    memset(&hints, 0, sizeof hints); //memset() is used to fill a block of
 memory with a particular value.
    hints.ai_family = AF_INET;
    hints.ai_socktype = SOCK_STREAM;
    hints.ai_flags = AI_PASSIVE;

    // protocol-
independent translation from an ANSI host name to an address.
    getaddrinfo(NULL, port, &hints, &res); //https://linux.die.net/man/3/g
etaddrinfo
```

```
    // Creating Socket
    int sockfd = socket(res->ai_family, res->ai_socktype, res-
>ai_protocol); //https://pubs.opengroup.org/onlinepubs/7908799/xns/socket.
html
    if (sockfd == -1)
        cout << "Error creating socket." << endl;

    // Binding Socket
    int bindsock = bind(sockfd, res->ai_addr, res-
>ai_addrlen); //https://pubs.opengroup.org/onlinepubs/7908799/xns/bind.htm
l
    if (bindsock == 0)
        cout << "Socket successfully bindined." << endl;
    else if (bindsock == 1)
        cout << "Error binding socket." << endl;

    // Listening
    listen(sockfd, BACKLOG); // https://pubs.opengroup.org/onlinepubs/7908
799/xns/listen.html

    int newfds[BACKLOG]{0};
    sockaddr_in their_addresses[BACKLOG];
    socklen_t their_sizes[BACKLOG];

    // creating threads
    pthread_t tids_recv[BACKLOG];
    pthread_attr_t attrs_recv[BACKLOG];

    string clientNames[BACKLOG];

    for (int i = 0; i < BACKLOG; ++i) {
        pthread_attr_init(&attrs_recv[i]);
    }

    for (int i = 0; i < BACKLOG; ++i) {
        their_sizes[i] = sizeof their_addresses[i];

        // accepting connection from client
        newfds[i] = accept(sockfd, (sockaddr *) &their_addresses[i], &thei
r_sizes[i]); //https://pubs.opengroup.org/onlinepubs/7908799/xns/accept.ht
ml

        // first request
        char tBuff[256]{0};
        // recving from client
        int rbSizeF = recv(newfds[i], tBuff, sizeof tBuff, 0); //https://p
ubs.opengroup.org/onlinepubs/7908799/xns/recv.html
        clientNames[i] = string(tBuff, 0, rbSizeF);

        // getting IP of client
        char theirIpBuff[INET_ADDRSTRLEN]{0};
        string clientIp;
        //return a pointer to the buffer containing the text string
        inet_ntop(AF_INET, &(their_addresses[i].sin_addr.s_addr), theirIpB
uff, sizeof theirIpBuff); // https://pubs.opengroup.org/onlinepubs/0096044
99/functions/inet_ntop.html
        clientIp = string(theirIpBuff, 0, strlen(theirIpBuff));

        // port number of client
        int theirPort = ntohs(their_addresses[i].sin_port); //convert valu
es between host and network byte order

        // message to server
        string prompt = "\"" + clientNames[i] + "\" connected to the serve
r from IP " + clientIp + ":" + to_string(theirPort);

        // message to other clients
        string toclientgreet = "\"" + clientNames[i] + "\" joined the chat
 room";
        cout << prompt << "\n";
        k++;


        sendToClients(&k, newfds, clientNames, i, toclientgreet.c_str(), 1
);
        args myArgs;
        myArgs.clientNames = clientNames;
        myArgs.clients = newfds;
        myArgs.k = &k;
        myArgs.src = i;

         // creating thread object https://pubs.opengroup.org/onlinepubs/7
908799/xsh/pthread_create.html
        pthread_create(&tids_recv[i], &attrs_recv[i], ReadFromClient, &myA
rgs);
    }

    // closing socket
    close(sockfd);

    // joining thread for all clients
    for (int i = 0; i < k; ++i) {
        pthread_join(tids_recv[i], NULL);
    }

    // closing connection when client leaves
    for (int i = 0; i < k; ++i) {
        close(newfds[i]);
    }

    return 0;
}
```

// WINDOWS CLIENT

```
// Includes

#include <iostream>
#include <sys/types.h> //https://pubs.opengroup.org/onlinepubs/009695399/b
asedefs/sys/types.h.html
#include <WS2tcpip.h> //https://docs.microsoft.com/en-
us/windows/win32/winsock/creating-a-basic-winsock-application
#include <string>
#include <cstring> //https://www.cplusplus.com/reference/cstring/
#include <vector> //https://www.cplusplus.com/reference/vector/vector/vect
or/
#include <thread> // https://www.cplusplus.com/reference/thread/thread/

#pragma comment(lib, "ws2_32.lib") // To include ws2_32 lib

using namespace std;

// Declaring variables
int k;

string name;
string serveradd;
string portnum;

bool stop_flag = false;

// Function to read from Server
```

```
void ReadFromServer(void* p) {
    int sockfd = *((int*)p);

    char buff[256]{ 0 };
    while (1) {
        memset(buff, 0, sizeof buff);   //memset() is used to fill a block
 of memory with a particular value.

        // reciving from server
        int rbSize = recv(sockfd, buff, sizeof buff, 0); // https://docs.m
icrosoft.com/en-us/windows/win32/api/winsock/nf-winsock-recv

        // if server is disconnected
        if (rbSize <= 0) {
            cout << "The Server Disconnected\n";
            stop_flag = true;
            break;
        }
        cout << buff << "\n";
    }
}

// Function to Send message to Server

void SendToServer(void* p) {
    int sockfd = *((int*)p);

    //if server disconneted
    while (1) {
        if (stop_flag)
        break;

        // user input
        string str;
        getline(cin, str);

        // sending input to server
        send(sockfd, str.c_str(), str.length() + 1, 0); //https://docs.mic
rosoft.com/en-us/windows/win32/api/winsock2/nf-winsock2-send
    }
}

// Main

int main() {
    WSAData wsData; //The WSADATA structure contains information about the
 Windows Sockets implementation.
    WSAStartup(MAKEWORD(2, 2), &wsData);  //The WSAStartup function initia
tes use of the Winsock DLL by a process.

    cout<<"\n\t\t\t\t************************";
    cout<<"\n\t\t\t\t* WELCOME TO CHAT ROOM *";
    cout<<"\n\t\t\t\t************************" << endl;

    // Asking for input and converting string to const char

    cout << "Enter your username" << "\n";
    cin >> name;
    const char *clientName = name.c_str();

    cout << "Enter IP address of server" << "\n";
    cin >> serveradd;
    const char *serverIP = serveradd.c_str();

    cout << "Enter port" << "\n";
    cin >> portnum;
    const char *port = portnum.c_str();

    // creating hints (valus for socket)

    addrinfo hints, * res;
    memset(&hints, 0, sizeof hints); //memset() is used to fill a block of
 memory with a particular value.
    hints.ai_family = AF_INET;
    hints.ai_socktype = SOCK_STREAM;

    // https://docs.microsoft.com/en-us/windows/win32/api/ws2tcpip/nf-
ws2tcpip-getaddrinfo
    getaddrinfo(serverIP, port, &hints, &res);

    // creating sockets
    int sockfd = socket(res->ai_family, res->ai_socktype, res-
>ai_protocol);

    // conneting to server
    connect(sockfd, res->ai_addr, res->ai_addrlen);

    // sending to server for first time
    send(sockfd, clientName, strlen(clientName) + 1, 0);
    // creating thread https://www.cplusplus.com/reference/thread/thread/t
hread/

    thread thread1(ReadFromServer, &sockfd);
    thread thread2(SendToServer, &sockfd);

    // joining threads  https://www.cplusplus.com/reference/thread/thread/
join/
    thread1.join();
    thread2.join();

    // socket close and cleanups
    closesocket(sockfd);
    WSACleanup();
    return 0;
}
```

// LINUX CLIENT

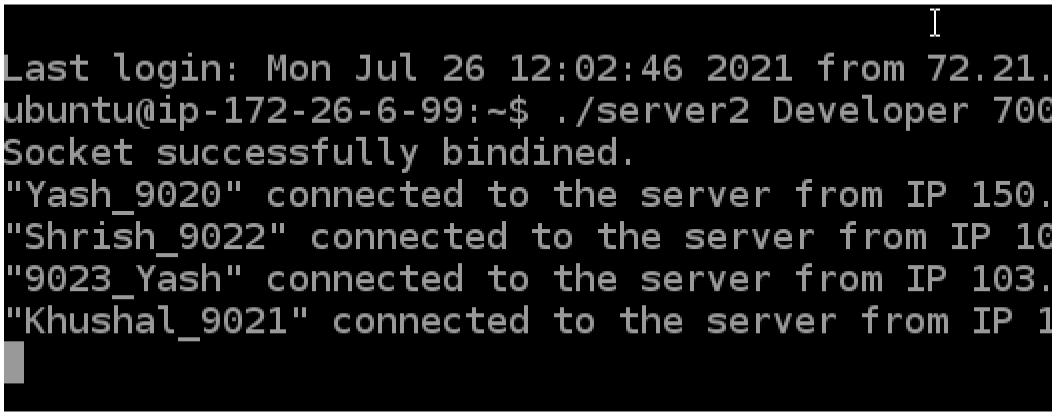
```
// Includes

#include <iostream>
#include <sys/types.h> //https://pubs.opengroup.org/onlinepubs/009695399/b
asedefs/sys/types.h.html
#include <sys/socket.h> //https://pubs.opengroup.org/onlinepubs/7908799/xn
s/syssocket.h.html
#include <arpa/inet.h> //https://pubs.opengroup.org/onlinepubs/7908799/xns
/arpainet.h.html
#include <netdb.h> //https://pubs.opengroup.org/onlinepubs/7908799/xns/net
db.h.html
#include <unistd.h> //https://pubs.opengroup.org/onlinepubs/7908799/xsh/un
istd.h.htm
#include <string>
#include <cstring> //https://www.cplusplus.com/reference/cstring/
#include <netinet/in.h> //https://pubs.opengroup.org/onlinepubs/009695399/
basedefs/netinet/in.h.html
#include <pthread.h> //https://pubs.opengroup.org/onlinepubs/7908799/xsh/p
thread.h.html

using namespace std;

// Declareing variables
int k;
```

```
string name;
string serveradd;
string portnum;

bool stop_flag = false;

// Function to Read From Server

void *ReadFromServer(void *p) {
    int sockfd = *((int *) p);
    // buff messages

    char buff[256]{0};
    while (1) {
        memset(buff, 0, sizeof buff); //memset() is used to fill a block o
f memory with a particular value.

        // reciving from server
        int rbSize = recv(sockfd, buff, sizeof buff, 0); // https://pubs.o
pengroup.org/onlinepubs/7908799/xns/recv.html

        // if server disconneted
        if (rbSize <= 0) {
            cout << "The Server Disconnected\n";
            stop_flag = true;
            break;
        }
        // print out message
        cout << buff << "\n";
    }
    pthread_exit(0);
}

// Function to Send message to Server

void *SendToServer(void *p) {
    int sockfd = *((int *) p);

    // stop sending when server is disconnected
    while (1) {
        if (stop_flag) break;

        // user input
        string str;
        getline(cin, str);

        //sending input to server
        send(sockfd, str.c_str(), str.length() + 1, 0); //https://pubs.ope
ngroup.org/onlinepubs/7908799/xns/send.html
    }
    pthread_exit(0);
}

// Main Program

int main() {

    cout << "\n\t\t\t\t************************";
    cout << "\n\t\t\t\t\t* WELCOME TO CHAT ROOM *";
    cout << "\n\t\t\t\t************************" << endl;

    // Asking for input and converting string to const char

    cout << "Enter your username" << "\n";
    cin >> name;
    const char *clientName = name.c_str();

    cout << "Enter your serverip" << "\n";
    cin >> serveradd;
    const char *serverIp = serveradd.c_str();

    cout << "Enter your port" << "\n";
    cin >> portnum;
    const char *port = portnum.c_str();

    // creating hints (valus for socket)
    addrinfo hints, *res;
    memset(&hints, 0, sizeof hints); //memset() is used to fill a block of
 memory with a particular value.
    hints.ai_family = AF_INET;
    hints.ai_socktype = SOCK_STREAM;

    // protocol-
independent translation from an ANSI host name to an address.
    getaddrinfo(serverIp, port, &hints, &res); //https://man7.org/linux/ma
n-pages/man3/getaddrinfo.3.html

    // createing socket
```

```c
    int sockfd = socket(res->ai_family, res->ai_socktype, res-
>ai_protocol);

    // connecting socket to server
    connect(sockfd, res->ai_addr, res-
>ai_addrlen); //https://pubs.opengroup.org/onlinepubs/7908799/xns/connect.
html

    // sendind detils to server for first time
    send(sockfd, clientName, strlen(clientName) + 1, 0);

    // constructing thread

    pthread_t tid_recv, tid_send;
    pthread_attr_t attr_recv, attr_send;

    // initialising threads attribute object https://pubs.opengroup.org/on
linepubs/7908799/xsh/pthread_attr_init.html

    pthread_attr_init(&attr_recv);
    pthread_attr_init(&attr_send);

    // creating thread object https://pubs.opengroup.org/onlinepubs/790879
9/xsh/pthread_create.html

    pthread_create(&tid_recv, &attr_recv, ReadFromServer, &sockfd);
    pthread_create(&tid_send, &attr_send, SendToServer, &sockfd);

    // joining thread object https://pubs.opengroup.org/onlinepubs/7908799
/xsh/pthread_join.html

    pthread_join(tid_recv, NULL);
    pthread_join(tid_send, NULL);

    close(sockfd);

    return 0;
}
```

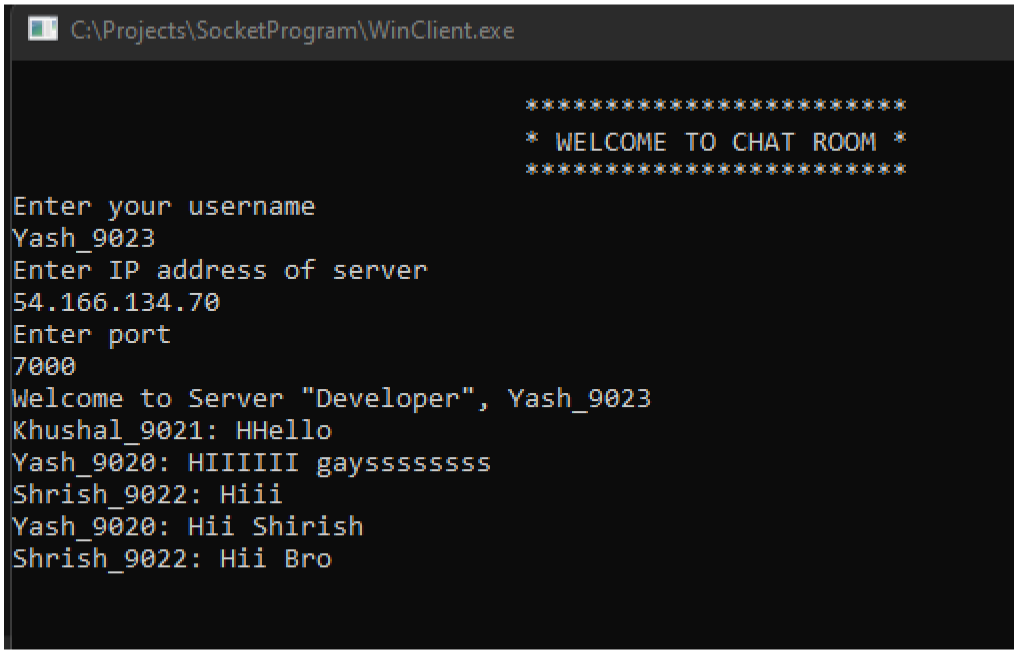
RESULTS:

// SERVER OUTPUT



```
Last login: Mon Jul 26 12:02:46 2021 from 72.21.
ubuntu@ip-172-26-6-99:~$ ./server2 Developer 700
Socket successfully bindined.
"Yash_9020" connected to the server from IP 150.
"Shrish_9022" connected to the server from IP 10
"9023_Yash" connected to the server from IP 103.
"Khushal_9021" connected to the server from IP 1
```

// CLIENT OUTPUT



```
C:\Projects\SocketProgram\WinClient.exe

                                **************************
                                * WELCOME TO CHAT ROOM *
                                **************************
Enter your username
Yash_9023
Enter IP address of server
54.166.134.70
Enter port
7000
Welcome to Server "Developer", Yash_9023
Khushal_9021: HHello
Yash_9020: HIIIIII gayssssssss
Shrish_9022: Hiii
Yash_9020: Hii Shirish
Shrish_9022: Hii Bro
```

CONCLUSION:

So, we have successfully learnt the logic to make a server and facilitate the joining of multiple members at a basic level using multiple threading! Although using multiple clients on one thread was an option but multiple threading provided us with the depth in knowledge.

Now all our group members have tested the chat server by connecting to it and chatting on it!

REFERENCES:

https://pubs.opengroup.org/onlinepubs/009695399/basedefs/sys/types.h.html
https://pubs.opengroup.org/onlinepubs/7908799/xns/syssocket.h.html
https://pubs.opengroup.org/onlinepubs/7908799/xns/arpainet.h.html
https://pubs.opengroup.org/onlinepubs/7908799/xns/netdb.h.html
https://pubs.opengroup.org/onlinepubs/7908799/xsh/unistd.h.html
https://www.cplusplus.com/reference/cstring/
https://pubs.opengroup.org/onlinepubs/009695399/basedefs/netinet/in.h.html
https://www.cplusplus.com/reference/vector/vector/vector/
https://pubs.opengroup.org/onlinepubs/7908799/xsh/pthread.h.html
https://www.cplusplus.com/reference/set/set/set/

https://pubs.opengroup.org/onlinepubs/009695399/basedefs/sys/types.h.html

https://docs.microsoft.com/en-us/windows/win32/winsock/creating-a-basic-winsock-application

https://www.cplusplus.com/reference/cstring/

https://www.cplusplus.com/reference/vector/vector/vector/

https://www.cplusplus.com/reference/thread/thread/