CLIENT-SERVER CHATROOM

OPERATING SYSTEMS LAB (CSC211) PROJECT

TEAM

TA : Shivam Sharma

Members :

1. Aditya Prakash (20JE0059)
2. Chandan Kumar (20JE0285)
3. Mukund Kumar Verma (20JE0589)
4. Siddharth Dhiman (20JE0948)

Objective

To create a chatroom server program in C using Socket Programming, Threads and Process synchronization using Mutex locks.

Methodology

The first step was to research how a client-server program works. We then looked into how such client-server programs could be implemented using socket programming.

Our next step was to look into all the header files that would later be used in the program such as **“sys/socket.h”** , **“pthread.h”** , **“netinnet/in.h”**, **“apra/inet.h”**, etc. Additionally, we studied the structures and functions associated with those header files. Then we planned out how to use socket programming and threads to make our program.

Using Socket Programming, we connect two nodes on a network so that they 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. A server program typically provides resources to a network of client programs. Client programs send requests to the server program, and the server program responds to the request.

We use threads to handle clients efficiently. As the server starts, the main thread is created that listens on the given port number. As soon as there is any connection request from a client, a connection is established. For every connection to a client, a new thread is created to start that communication between server and client, and continue listening for requests from other clients.

If one client logs into the server, I want to lock that connection request completion process with mutex, so that if any other client tries to join the server they would have to wait for the current client’s connection to be established and then proceed to join the server. This prevents a deadlock situation.

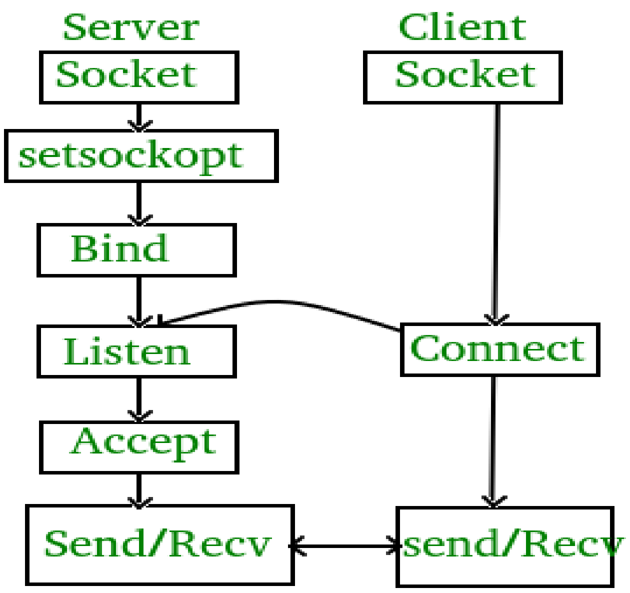
After that, we moved on to the coding phase.

We started to construct structures and implement different functions in order to simulate a single client-server program where the client would just connect to the server.

Then we created a structure called **“sockaddr_in”** for both client and server and initialized their member elements to the desired values.

Then we created a socket by calling the **‘socket()’** and stored its address in a file descriptor **‘sockfd’** .

After that we used **‘bind()’** which binds the server socket to a specific address (IP Address and port and **‘listen()’** which indicates a readiness to accept client connection requests.



Basic implementation of a general Client-Server Program

From the client side we used **'connect()'** to send connection requests to the server through the specified port.

From the server side we accept the connection request from the clients using **'accept()'** and assign them to the structure that stores client information.

The next hurdle was to connect and manage multiple clients. For that we created an array of clients that can store an appropriate number of clients to avoid system overload

Now we reached the part where we developed simultaneous message passing functions for both client and server. For the client we created a thread (**send_msg_thread**) and assigned **send_msg()** to it. Similarly we created another thread (**recv_msg_thread**) and assigned **recv_msg()** to it. For the server part we created a single thread (tid) for every client and assigned **'handle_client()'** to it.

In **'handle_client()'**, we receive messages from the client using '**recv()'** and send messages to the other clients using **'send()'**. All of this can be done simultaneously for different clients without entering the deadlock using **mutex_lock( ).**

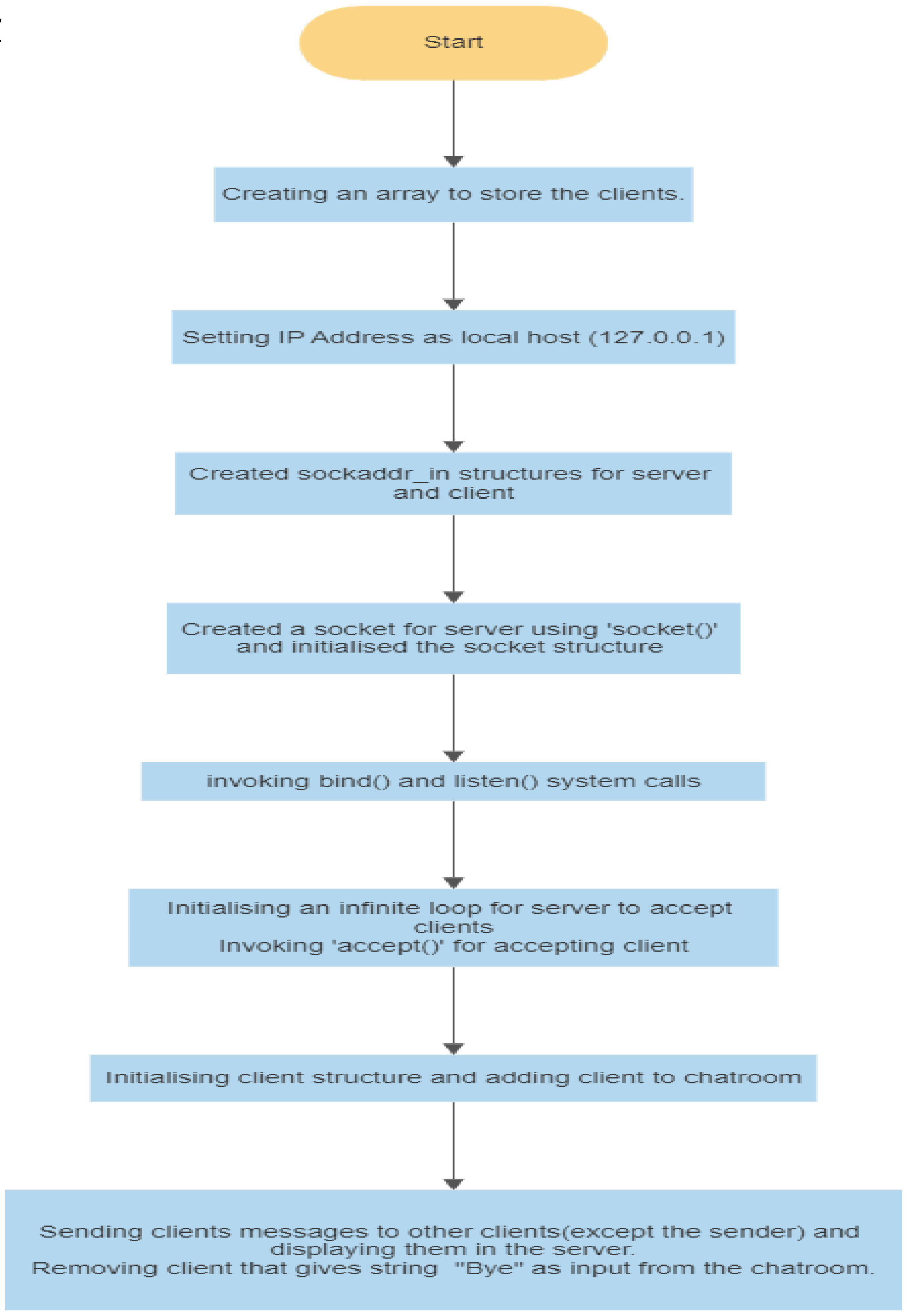
The client can exit the chatroom by typing the string " **Bye** ". This will remove the client from the array in the server and closes the socket of the client.

Our final effort was aimed towards enabling the client to join the chat room server from another device connected on the same network as the server with the help of Transmission Control Protocol (TCP) but we were unable to achieve our goal.

Flowchart

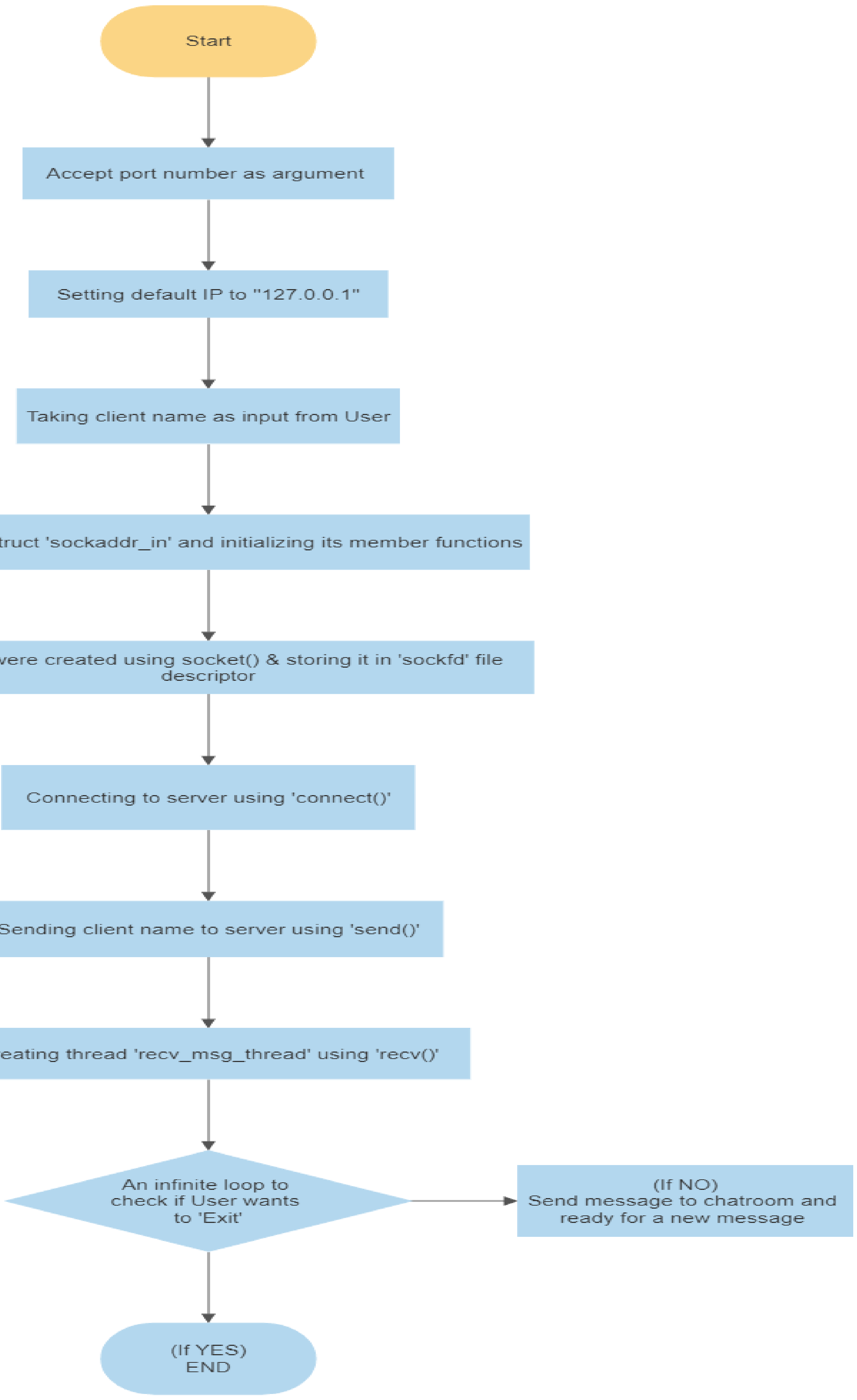
Flowchart

for Server

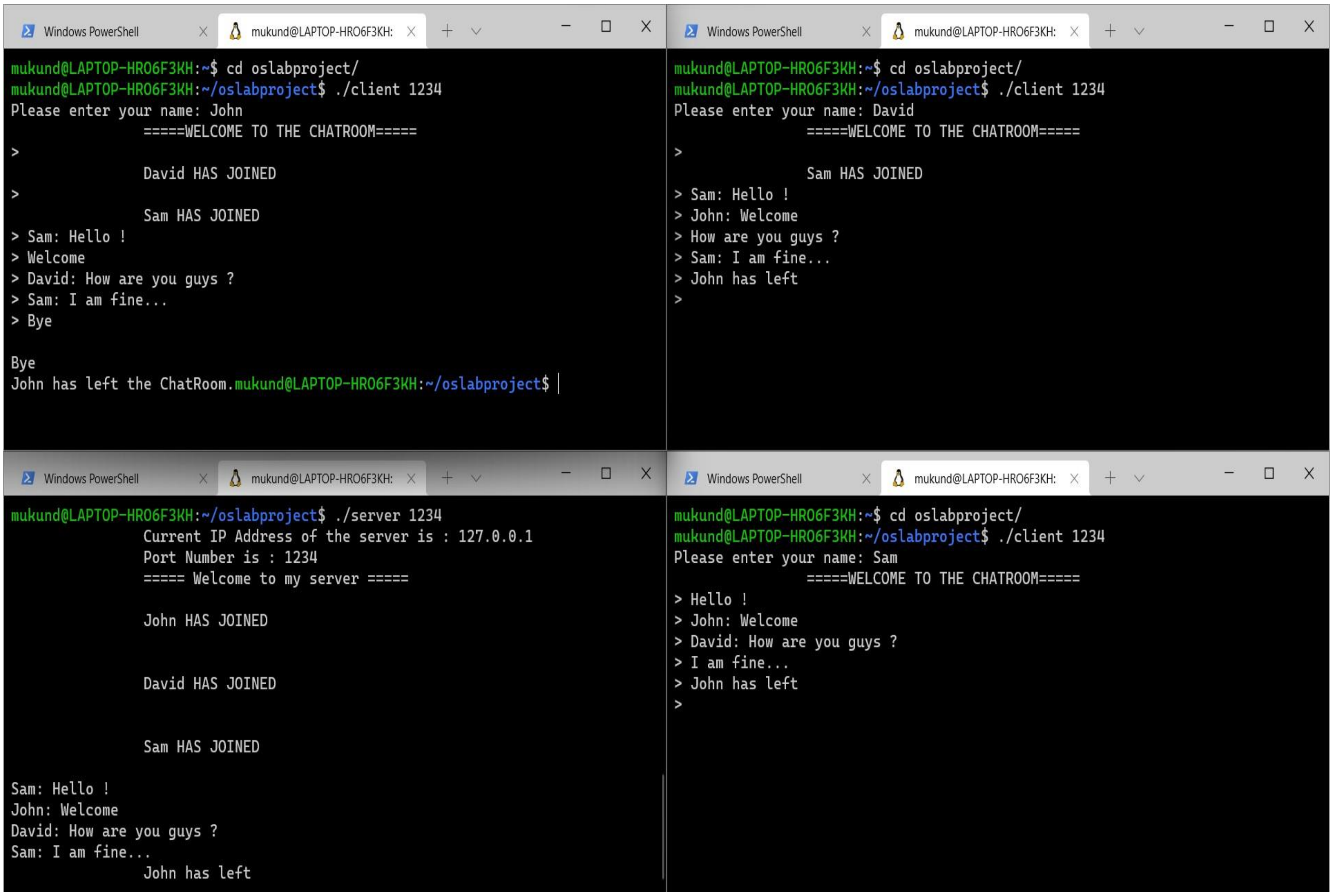

Start
Creating an array to store the clients.
Setting IP Address as local host (127.0.0.1)
Created sockaddr_in structures for server and client
Created a socket for server using 'socket()' and initialised the socket structure
invoking bind() and listen() system calls
Initialising an infinite loop for server to accept clients
Invoking 'accept()' for accepting client
Initialising client structure and adding client to chatroom
Sending clients messages to other clients(except the sender) and displaying them in the server.
Removing client that gives string "Bye" as input from the chatroom.

Flowchart

for Client


Start
Accept port number as argument
Setting default IP to "127.0.0.1"
Taking client name as input from User
Creating Struct 'sockaddr_in' and initializing its member functions
Sockets were created using socket() & storing it in 'sockfd' file descriptor
Connecting to server using 'connect()'
Sending client name to server using 'send()'
Creating thread 'recv_msg_thread' using 'recv()'
An infinite loop to check if User wants to 'Exit'
(If NO) Send message to chatroom and ready for a new message
(If YES) END

Output



```
mukund@LAPTOP-HRO6F3KH:~$ cd oslabproject/
mukund@LAPTOP-HRO6F3KH:~/oslabproject$ ./client 1234
Please enter your name: John
                =====WELCOME TO THE CHATROOM=====
>
                David HAS JOINED
>
                Sam HAS JOINED
> Sam: Hello !
> Welcome
> David: How are you guys ?
> Sam: I am fine...
> Bye

Bye
John has left the ChatRoom.mukund@LAPTOP-HRO6F3KH:~/oslabproject$
```

```
mukund@LAPTOP-HRO6F3KH:~$ cd oslabproject/
mukund@LAPTOP-HRO6F3KH:~/oslabproject$ ./client 1234
Please enter your name: David
                =====WELCOME TO THE CHATROOM=====
>
                Sam HAS JOINED
> Sam: Hello !
> John: Welcome
> How are you guys ?
> Sam: I am fine...
> John has left
>
```

```
mukund@LAPTOP-HRO6F3KH:~/oslabproject$ ./server 1234
                Current IP Address of the server is : 127.0.0.1
                Port Number is : 1234
                ===== Welcome to my server =====

                John HAS JOINED


                David HAS JOINED


                Sam HAS JOINED

Sam: Hello !
John: Welcome
David: How are you guys ?
Sam: I am fine...
                John has left
```

```
mukund@LAPTOP-HRO6F3KH:~$ cd oslabproject/
mukund@LAPTOP-HRO6F3KH:~/oslabproject$ ./client 1234
Please enter your name: Sam
                =====WELCOME TO THE CHATROOM=====
> Hello !
> John: Welcome
> David: How are you guys ?
> I am fine...
> John has left
>
```