Assingment-5

Group Details:

1. Adarsh Dhakar → 22CS01040
2. Avik Sarkar → 22CS01060
3. Debargha Nath → 22CS01070
4. Soham Chakraborty → 22CS02002

Github Repository Link:
https://github.com/adarshdhakar/cn_lab_sheet5/

Images Directory Link:
https://github.com/adarshdhakar/cn_lab_sheet5/images

Report.pdf Link:
https://github.com/adarshdhakar/cn_lab_sheet5/Report.pdf

Demo Video Link:
https://github.com/adarshdhakar/cn_lab_sheet5/Demo.mp4

1.

Why C++ over C ?

- Use of **string** makes it easier to handle text messages instead of relying on character arrays where we would have used strcpy() and strcat().
- Use of **set<int>** to maintain list of active clients efficiently.
- Use of **map<string,int>** makes is easier to maintain mapping of active clients to their respective socketfd. This allows for quick lookups, insertions, and deletions.
- Use of **map<string, set<int>>** to store the **socketfd** of the members in a **room**.
- Use of **map<int, string>** to store the **rooms name** where a particular **socketfd** belongs to.

What all libraries used ? Funtionalities provided by these.

#include <bits/stdc++.h>
Functions used: string, set, iostream

#include <netinet/in.h>
Provides sturctures for internet addresses **(sockaddr_in)**

#include <netdb.h> (only in Client)
Contains **gethostbyname()** to resolve hostnames to IP addresses.

#include <pthread.h>
Used for multi-threading:

- Creating threads for handling multiple clients on server.
- Creating separate threads for reading and writing for the client.

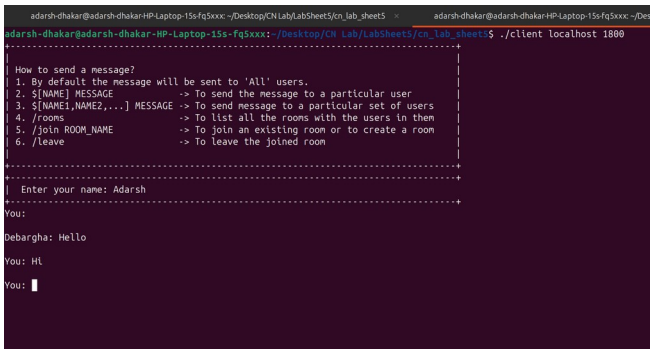
#include <unistd.h>
Provides access to the POSIX operating system API and contains system calls related to process control, file handling and I/O operations.
Functions used: **read, write, close, shutdown, sleep**

Features and Functionality:

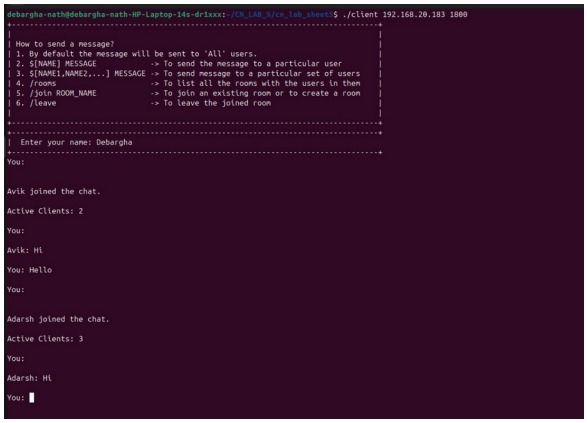
Broadcast Messages

‘hi’ sent by Adarsh is sent broadcasted everyone.

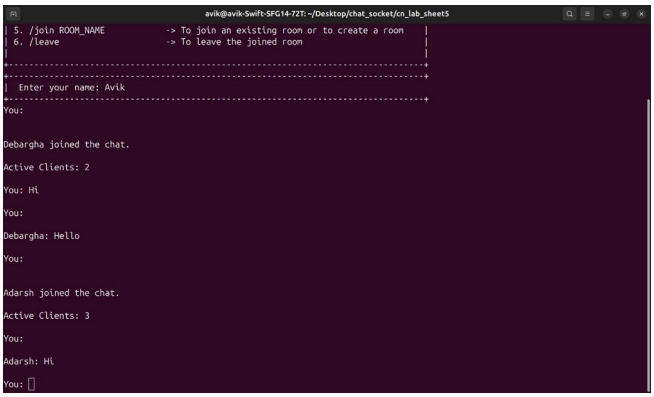
Adarsh



Debargha



Avik

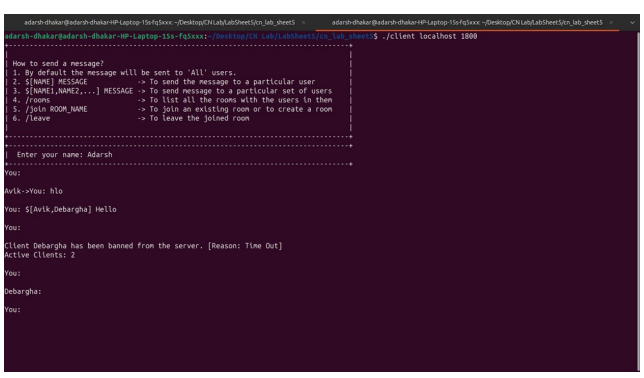


Private Messaging Direct Messaging

Messages sent using ‘$’ commands are sent to the specified people only.

Messages sent by Adarsh using $[Avik,Debargha] are sent to only Avik and Debargha

Adarsh



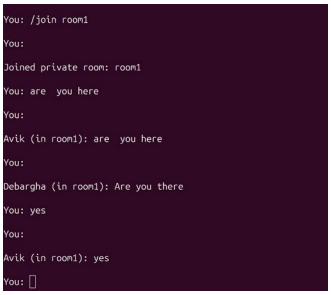
Avik

Debargha

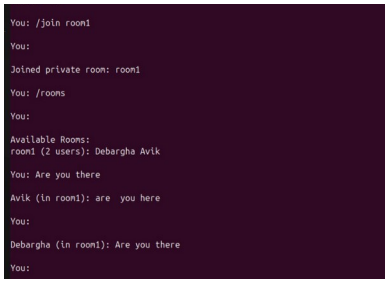
Join a Room **Room Messaging** **Leave a Room**

Messages sent after joining room are only sent to the members in the room.

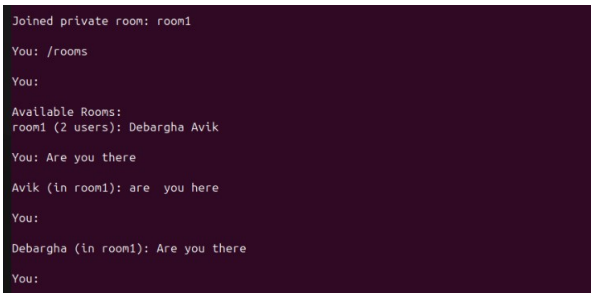
Avik



Debargha

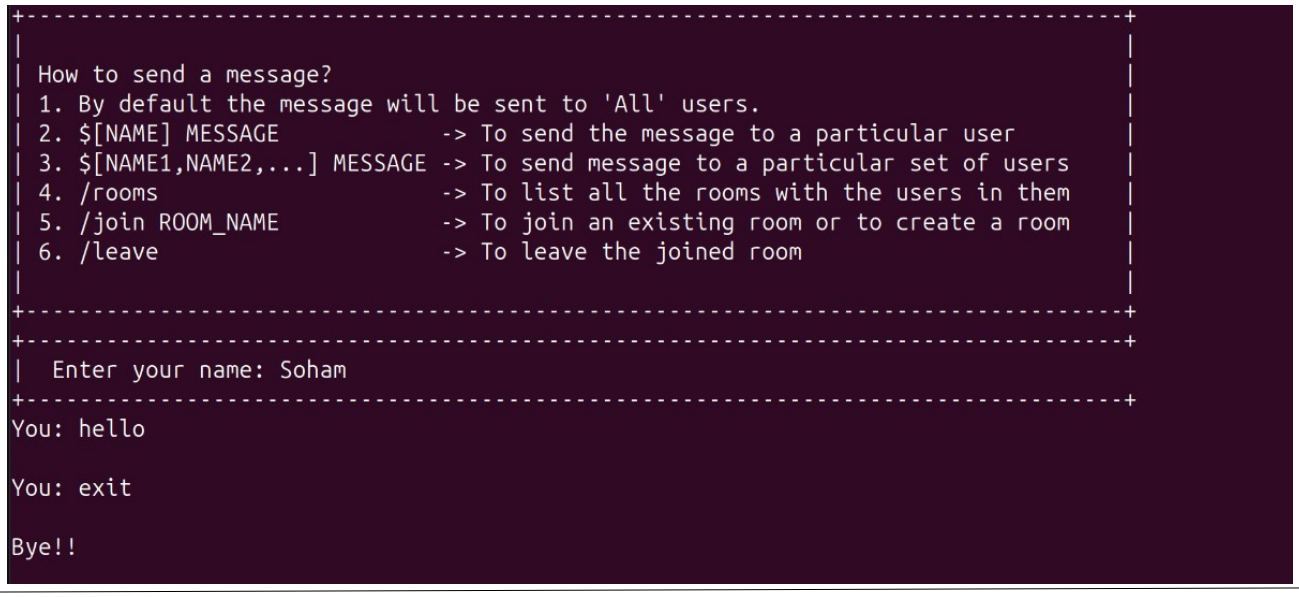


Check available rooms and members



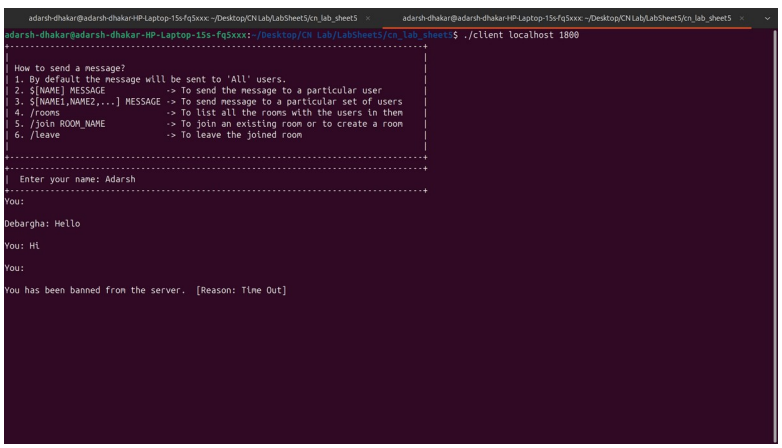
Exit Functionality

User **Soham** exited using the exit command.



Timeout (120 sec)

After 2 minutes the user **Adarsh** is banned due to time out, and he can’t send any more messages now.



Explanation of Flow and Working:

Server (threaded):

1. Main Function / Initialization:

The server initiates by creating a **server socket as any typical socket**, Upon establishing this socket, the server will make a **thread** to handle that client's communication. This ensures the server can handle multiple clients simultaneously without blocking or waiting on individual connections

2. Client Handler Function:

The Client function is invoked within a thread to manage the client's communication. Initial variable are set as required. Connection acceptance and client identification is done.

3. Timeout Thread:

Client spawns a separate timeout monitoring thread for each client. This thread is responsible for periodically checking the client's inactivity time and ensuring that the client does not exceed the allowed timeout limit.

4. Message Processing:

Messages are processed and parsed based on the following commands.

a) **By default** the message will be sent to **'All'** users.
b) **$[name]message** → To send the message to a particular user
c) **$[name1,name2,..]message** → To send message to a particular set of users
d) **/rooms** → To list all the rooms with the users in them
e) **/join ROOM_NAME** → To join an existing room or to create a room
f) **/leave** → To leave the joined room
g) **exit** → To exit the chat

Note: joining another room automatically takes you out of your current room.

Client:

1. Main Function / Initializations:

The Client is responsible for handling the **bidirectional communication** between the client and server. To efficiently manage the flow of messages, the client utilizes **two distinct threads**—

- one for reading messages from the server
- and another for writing messages to the server.

2. Read Thread:
It constantly reads data from the server's socket and processes or displays it accordingly.

3. Write Thread:
It waits for the user to input messages and sends them through the socket connection (using write()).
The write thread is responsible for gathering and formatting user input or system-generated messages and dispatching them to the server.

Both threads operate independently, ensuring that sending or receiving messages doesn't block the other operation.

2.
i) Develop a Chat Server program using threads which can:

- **Handle multiple clients at the same time.**

Mulitple client's read and write are handled at the same time, using threads.

- **A client can join/disconnect from the chat.**

Exit functionality is implemented for the same.

- **Two clients can chat via server.**

Private messaging is implemented.

- **A client can choose to broadcast the message to all clients alive.**

Messages broadcasting is handled as a default case.

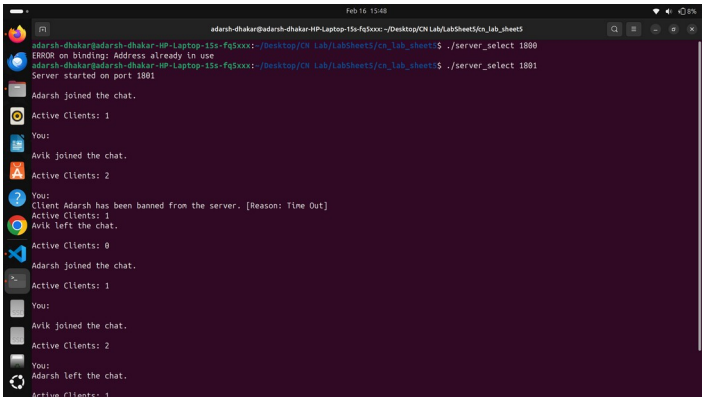
- **Add more functionalities**

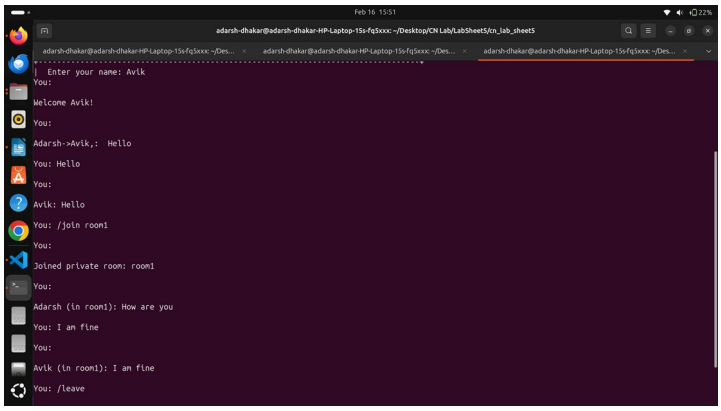
Functionalities like :

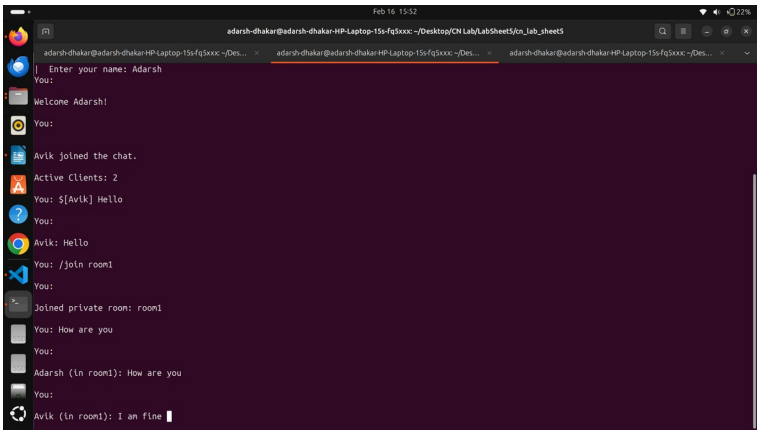
a) timeout
b) create and join a group
c) message and leave a group are also implemented

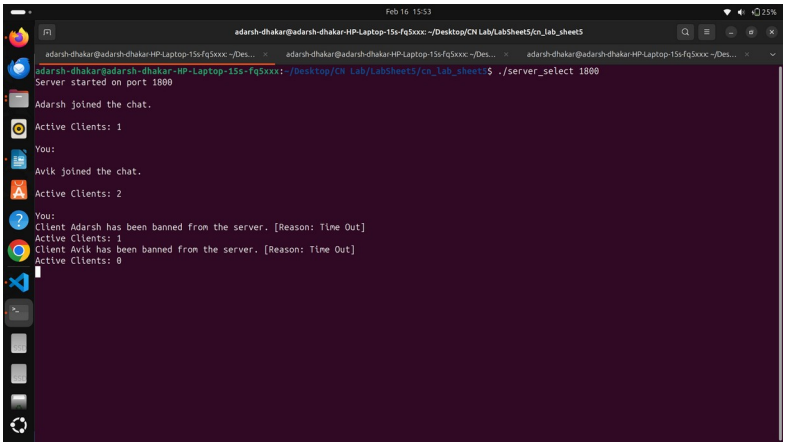
ii) Design the Chat Server program (in place of threads) use select() system call to connect multiple clients.

Chat Server Program is in **server_select.cpp**









iii) Design the Client program for the chat-server.

Client Progam is in **client.cpp**