**Final Project Report**

**Group members:**

Ayush Srinivas

Saisarath Kanamarlapudi

Suhaas Chandra Chowdary Achanta

Sreekaar Konchada

Vidith Reddy Salla

**Project Synopsis:**

For our project we decided to work on a client-server chat application. The server routes messages between clients. Clients can send messages to a public room, or send private messages to other clients.

**Running the Project (User Manual):**

Compiling the Server:

gcc final\_project\_server.c -lpthread -o server

./server 7777 20

NOTE: 7777 is the port number

NOTE: 20 is the maximum number of clients allowed to connect to

the server at any given time (maximum number of simultaneous connections). This can be changed by changing the value of the command-line parameter.

Compiling the Client:

gcc final\_project\_client.c -lpthread -o client

./client client1 localhost 7777

OR

./client client1 127.0.0.1 7777

NOTE: client1 is the ID of the client that will be registered with the server

NOTE: The MAXIMUM size of a client ID is 20 characters (if you have an ID

with more than 20 characters, the client ID will only take the first 20 characters).

The server **must** be run before any clients are run because the server is the program accepting connections. If the server program is not running, the terminal will indicate that the client's connection was refused.

The server and client programs should be compiled and run in separate tabs.

Once connected to the server, clients can send messages to specific clients or the entire room.

**Sending Messages from a Client:**

To send a message to the room, input the following command into any client’s terminal window:

>>> send "room" "hello everyone!"

**NOTE:** You must use double quotes.

To send a message to a specific client, input the following command into any client’s terminal window:

>>> send "client\_id" "this is a private message"

**NOTE:** You must use double quotes.

Replace “client\_id” with the ID of the client you want to send the private message to (this is the ID you enter when you first create a client in the terminal). A client **cannot** send messages to themselves.

**Closing the Application:**

1. You can close clients by typing Ctrl+C in the client’s terminal. If all clients are closed, the server will also automatically exit.
2. The server can also be closed by typing Ctrl+C. This will automatically close all the clients. **However,** it will take approximately 2 minutes for the port you ran the server on to re-open. In this time, you cannot re-run the server, and you must wait, because the port is busy. This is because when you close a connection from the server’s end, the network connection enters a TIME\_WAIT state and ties up the port for some time. This TIME\_WAIT state is always avoided if you close the connection from the client end. You can read more about this unfixable issue here:

<https://hea-www.harvard.edu/~fine/Tech/addrinuse.html>

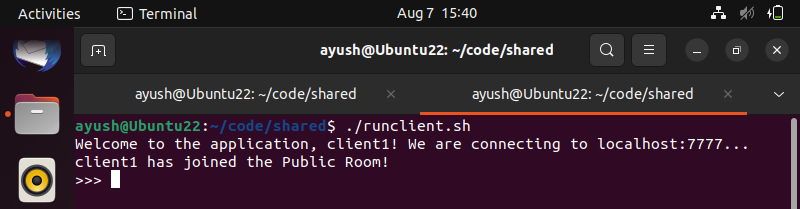
**Scope**

**Multi-Threading:**

The server spawns a new thread to handle each client. Each thread receiving “send” commands from them and uses these commands to send messages between clients

The client spawns 3 new threads: one to handle messages coming from the server, one to continuously prompt the user for new commands, and one to continuously look for input coming from stdin.

**Mutex and Condition Variables:**

In the client, we use mutex with a condition variable to tell the program when to print the message prompter. The message prompter is the 3 arrows “>>>” that prompt the user to enter a “send” command.

**TCP socket programming**

We use TCP connections in both the client and the server to send and receive messages over the network.

**Signals**

The SIGTERM and SIGINT signals have been used to gracefully close both the client and server. The signal handler closes all connections and file descriptors, and exits afterward.

**File Descriptors, Read, Write**

We use file descriptors when writing and reading messages in both the client and the server.

**Activity Log**

Ayush Srinivas – Wrote the server-side code (final\_project\_server.c). Helped coordinate group calls.

Saisarath Kanamarlapudi – Wrote half of the final report (Project Synopsis and Usage) and performed testing.

Suhaas Chandra Chowdary Achanta – Wrote the other half of the final report (Scope and Activity Log). Helped coordinate group calls.

Sreekaar Konchada – Wrote half of the client-side code (final\_project\_client.c) and helped in planning the chat application architecture.

Vidith Reddy Salla - Wrote half of the client-side code (final\_project\_client.c) and helped in planning the chat application architecture.