**SE3313 Project**
**Group 40**

## Introduction

This application utilizes the concepts of threads, semaphores and parallel processing to create a multi-user multi-transaction server and chat client. A user can run this C#-based chat application on their device, communicating with other users over 9 different chat rooms and their desired username. The server is run on the Google Cloud Platform with an designated external IP address allowing for clients to connect from different computers as long as they can run the C# application.

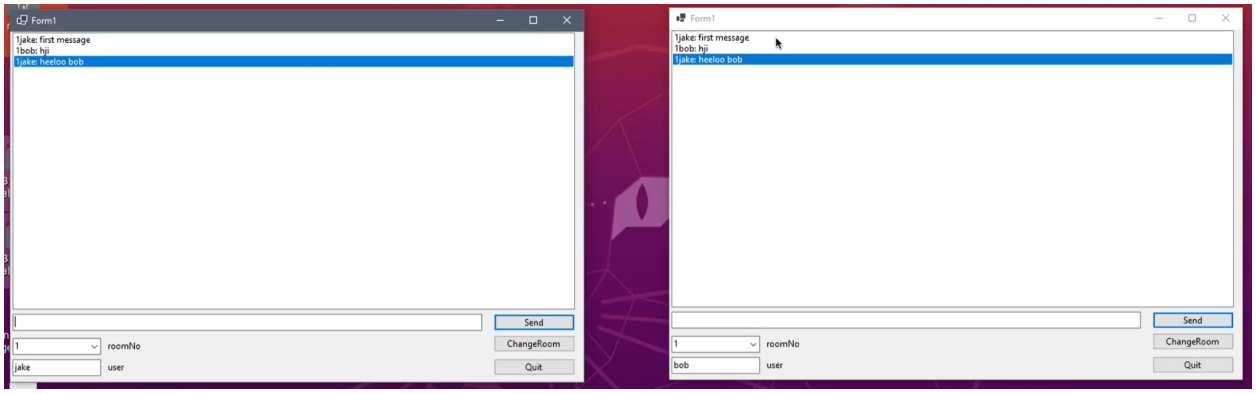
## Server

The Server uses the class ServerThread to handle multiple clients. For each new client, the ServerThread class will create a new SocketThread and store the respective reference in a vector. New SocketThreads are given the total number of chatrooms. An instance of a SocketThread will connect to a specific chat room, and then listen for data sent from the client. Certain commands will do certain things, '/' for instance, will cause the specific SocketThread to change the chat room to a new one specified in the received string(after conversion). After data is read from the client application, the SocketThread instance will enter a semaphore blocking call to wait. The wait will prevent interruption to clients' messages. The SocketThread then will look for client sockets with the same chat room number, and write the user data to those clients.

## Client

Clients are in the form of C# windows form application that connects to the server. The # of rooms available in the chat room are sent to each client from the Server. Each respective client will display a combobox button to the user which displays all of these values. Clients will initially be put into room one, but must specify a username along with a message. When a client is to press 'send', the data(username,currentroom, and password) are all written to the Server, which will handle this data. Additionally, the client will continuously check for messages in the specific socket, and if a message is received, the data is shown in the GUI to the user. The client informs the Server which room they are in by including the room number at the beginning of each message sent. If a user desires to change their current room, they can do so by selecting a value from the provided combobox(with all possible room #'s) and clicking 'ChangeRoom'. This will cause a message to be sent to the server with a '/' along with the specified combobox selection. The Server handles this data accordingly. C#'s TCP socket modules connect

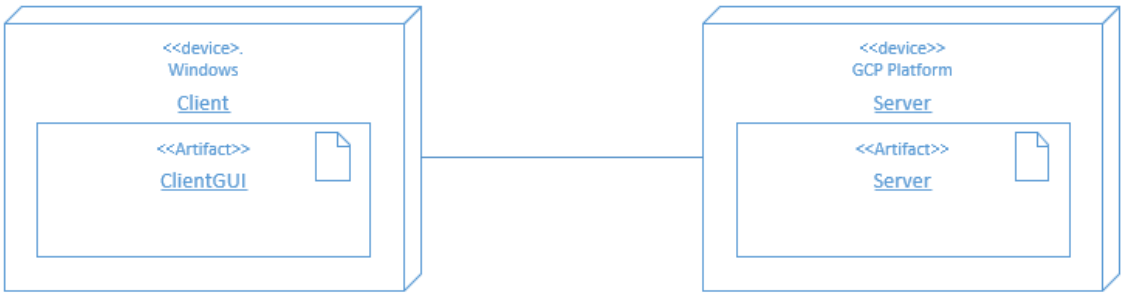
with C++ server fairly well to ensure each client can communicate fluently with the server. Below you can see two clients communicating in chatroom #1.



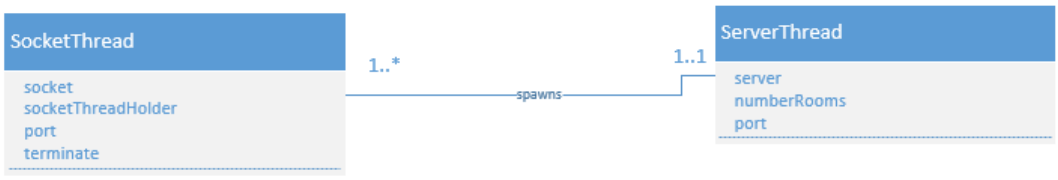

**Graceful Termination**

For clients, when the “Quit” button is clicked, the connection with the server is ended gracefully by first sending a message that the client is leaving and then closing the TCP connection before exiting the application.

On the Server side, if the server was stopped or quit, the method in ServerThread runs a for loop for every reference of client threads connected, closing this and removing them from the vector object.




**We see a UML deployment diagram for our application. Clients run the application on a Windows host system, and this communicates with a Server that runs the server application.**




**Here is a simple class diagram for the Server operation. ServerThread will spawn multiple SocketThread instances.**