COMP1549 ADVANCED PROGRAMMING COURSEWORK

Lakshman Thillainathan 001064670
COMP1549: Advanced Programming
University of Greenwich
United Kingdom

Abstract- I have created a network distribution system for group-based client-server communication. The final code was created with clear design patterns in mind and component based development.

Key Words:

Threading - a thread is a single flow of control within a program. Threads exist within processes. Explained further later..

Socket - The endpoint of a two way communication link. It is a combination of both an IP address and a Port number

IP address - A unique string of characters that is used to identify a computer over a network.

Port - A number used to uniquely identify the service being used for the connection. A well known port is 80, used for the web server.

I. Introduction

Communication is essential in this world, especially during a global pandemic where meeting in person is prohibited. In order to connect with people nowadays we used apps such as WhatsApp, Discord, Teams. These apps are used daily. But how do they work?
Well obviously with apps such as whatsapp, our mobile phone numbers are used, but for programs like Discord and Skype. In order understand these apps, this task allowed us to replicate a messaging system. My project began with the above keywords. Most devices, whether it be a computer or a phone, have a unique string of characters that can be used within a network to identify the device.

In order to use the IP address to send a message, we need to also provide a port number, this basically describes the service that is being used. for our case any number above 1000 should be fine as they should have less traffic. When paired with the IP address we can create a socket. The task required you create a client-server network. I decided to create my chatting program using the programming language python.

II. Design/Implementation

To begin my project, I needed to know the modules I would need. I knew I had to use sockets as the fundamentals. I began my program with 2 separate files. One containing all the server functions and all the server handing, the other file containing the client functions. When I first implemented this version of the code, I wanted to create a command line interface to keep it simple and get a better understanding of socket programming. My first version of the chat would take in the clients message and simply relay it, while also printing the IP and port number used by the client to connect, and was able to take in multiple clients. I then decided to add identity to each client by asking them for a name to enter before entering the server, this was then used alongside each message the client sent. The task required me to take in 3 inputs for the ID (Partially implemented) a port and IP to listen to and the port and IP address of the server you are trying to connect to. I decided, that it would be better suited to have a GUI as the login screen. So I used QtDesigner to create a PyQt ui file of the login window. As shown in figure 1.

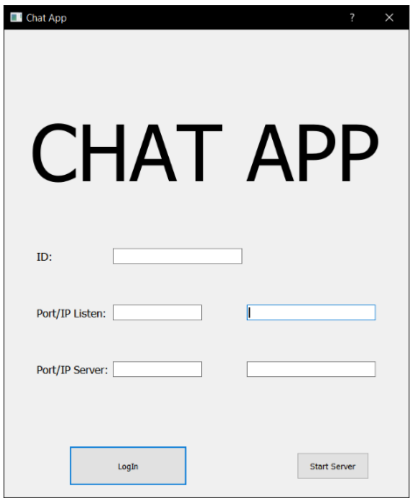


Figure 1. Log In screen Graphical User Interface

After creating the GUI and providing appropriate variable names, I was able to access these through my main ChatApp.py file by importing it in. Upon reaching this step of my program I wanted to have it possible to run both the server and client at once. This is where threading became an essential concept to learn. In my years of programming, I'd have to say this is the most interesting and fun concept I have learned. Threading allows for you to create a program and have 2 processes work simultaneously. In this instance I needed to have the server ,client and GUI run simultaneously. I made it possible to create a server or client provided that the user entered valid inputs for the text input boxes. The program essentially ran on a try statement, so the client would try run and if it couldn't then it would try start as the server, this was essentially the co-ordinator role as the server. I then changed this to fulfil the GUI and provide separate functions to start a server or log in, this was to allow the GUI to eventually close when a client is created. This was to avoid multiple clients on one interface as it led to stacked messages and a delay in the server. I tried to get another Chat Window graphical interface working (figure 2). However, when trying to open both ui's at once, and possibly due to the amount of threads running, the program kept crashing. I then decided that it would be nice to mix both graphical and command line interface, and ultimately decided to have the main chat element provided in the command line interface. I found that other python interpreter's were fine.

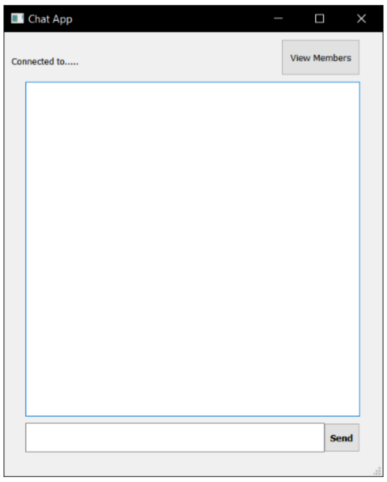
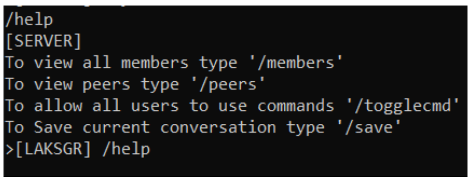


Figure 2. "ChatScreen.ui" Graphical User Interface

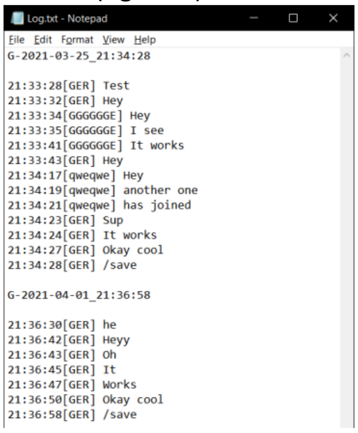
With the main chatting aspect of the program complete, I decided to create co-ordinator exclusive commands, as it makes sense for them to have a little privilege. Also keeping in mind that I need to be able to save the chat log. So I made a couple of commands(figure 3).



```
/help
[SERVER]
To view all members type '/members'
To view peers type '/peers'
To allow all users to use commands '/togglecmd'
To Save current conversation type '/save'
>[LAKSGR] /help
```

Figure 3. Co-ordinator Commands

To create each command, I read the words after the "/" character and followed the command with if statements. Using the re module to split the square brackets from the username. When viewing the members of the group it will create a string of all current members then send it out to everybody. To view the current peers, it does the same as the members. To toggle the commands for all users, I used "or" in an if statement. To save the conversation the user can enter "/save" and using the inbuilt python function I append to the "log.txt" file (figure 4).



```
G-2021-03-25_21:34:28

21:33:28[GER] Test
21:33:32[GER] Hey
21:33:34[GGGGGGE] Hey
21:33:35[GGGGGGE] I see
21:33:41[GGGGGGE] It works
21:33:43[GER] Hey
21:34:17[qweqwe] Hey
21:34:19[qweqwe] another one
21:34:21[qweqwe] has joined
21:34:23[GER] Sup
21:34:24[GER] It works
21:34:27[GER] Okay cool
21:34:28[GER] /save

G-2021-04-01_21:36:58

21:36:30[GER] he
21:36:42[GER] Heyy
21:36:43[GER] Oh
21:36:45[GER] It
21:36:47[GER] Works
21:36:50[GER] Okay cool
21:36:58[GER] /save
```

Figure 4. "Log.txt" text file

The texts are saved using the first letter of the current co-ordinator followed by the date and time of save. This is used as the heading. Then all texts from the start of the server to when the command was entered is saved with timestamps for each message.

III. Analysis and Critical Discussion

The project could have had minor additional features to make the program better. One of the biggest things I wish I could have implemented, is the secondary graphical interface, I believe this would have been simple if the program didn't continuously crash. On hindsight I could have taken all the contents of the ui file and placed it into the main file. Another feature that I did not have time to properly implement is when the server disconnects the co-ordinator should handle it with a new co-ordinator. This was a feature that due to the end result of my program, was more difficult to implement at the end. On hindsight I think I could have created a try except statement to try start a new server if the server crashes and the clients can no longer communicate. Rather than keeping the username in the client side and using it as a variable to pass to the server, I could have easily done this by sending the username along with the port information when connecting to the server. This would have been useful if I wanted to create a dictionary containing the usernames with the IP and Port information stored together. Furthermore, I could have stored all this information in an excel file, I could have separated name and ID so each person has a name in the server and is more secure, could have made certain peers unable to become the server etc.

IV. Conclusions

This project was quite fun to work on, having learned such an interesting and useful concept, threading. This project challenged me to understand networking more especially as it is probably one of my weaker points. As mentioned in the analysis section I could have made several minute changes to my program to make it more user-friendly. For future I'd also like to make the program work through TCP connections too, and connect to computers on other networks. I am most definitely going to be using threading in my future projects.

References

Python Socket Programming Tutorial, added by Tech With Tim
[Online]https://www.youtube.com/watch?v=3QiPPX-KeSc&t=2422s&ab_channel=TechWithTim
[Accessed 27th February]

Socket Programming HOWTO, by Gordon McMillan
https://docs.python.org/3/howto/sockets.html

threading library -
https://docs.python.org/3/library/threading.html

Python Threading Tutorial: Run Code Concurrently Using the Threading Module, added by Corey Schafer
[Online]https://www.youtube.com/watch?v=IEEhzQoKtQU&ab_channel=CoreySchafer
[Accessed 13th March]