Creating a Client-Server Communication

Melchisedek M Mbamba, Hanzala A Razzak, Maciej Kampka, Marc Henry May and Luis M Figueiroa

COMP1549: Advanced Programming
University of Greenwich
Old Royal Naval College
United Kingdom

Abstract. Client-server is a system that can execute the roles of both the client and server in order to promote the sharing of information between them. It enables many users to use the same database at the same time, and the database will hold a large amount of information. We were required to implement a Graphical User Interface (GUI) or Command-Line Interface (CLI) based networked distributed system for a group-based client-server communication, which conforms with the following requirements. This report will provide information about the development and design of the client-server model that we have created as well as talk about benefits and issues we have met during the developing process.

Keywords: Command-Line Interface (CLI), Python, IP Adress, Server

1 Introduction

With the advancement in technology, the Web is becoming much more important in our daily lives, in which virtually every- thing we do nowadays involve the use of the web. More so, the application of the Web is not limited to computers but it is opened to different kinds of intelligent digital devices, for example the mobile ones. Also, the architecture of the Web is the Client-Server model, in which as a result the communication between the client and server is the first thing we should be concerned about. Client/server systems have increasingly minimized application development time by dividing functions of sharing information into both the client and server. The client is the requester while the server is the provider of service. In most client-server environments, the data processing is handled by the server, and the results are returned to the clients, which is made to speed up the rate of performance. For example, in a workstation, a printer can be attached to a computer (representing the clients) while other computers sharing from it are the server.

In the computing world today, client-server systems have become so popular because they are being used virtually every day for different applications. Some of the standardized protocols that clients and servers use to communicate with themselves include: File Transfer Protocol (FTP), Simple Mail Transfer Protocol (SMTP) and Hypertext Transfer Protocol (HTTP). Thus, the Client-server system can be defined as a software architecture made up of both the client and server, whereby the clients always send requests while the server responds to the requests sent. Client-server provides inter-process communication because it involves the exchange of data from both the client and server whereby each of them performs different functions.

This main focus of the coursework is to implement a graphical user interface (GUI) or command line interface (CLI) based on the requirements given. In order to have solid communication 2-3 times per week we would have a group call which would last approximately 1-4 hours. Within that time, we worked on the project and communicated our next steps. This project will require people to join a server (No current user limit) in which they will need to connect with the user's name, id and server port (Would be given). The first user which will join the server will automatically become a coordinator (Max 1 at a time) and have special tasks such as moderating the server and inform other members to update their list of existing members so everyone can be up to date. As it's a server everyone should be able to direct message every other member through the server. Once when the current coordinator will quiet or not be active (Specific time frame) a new coordinator will be automatically approved and carry the role on.

2 Design/implementation

In this section we will talk about the design and the implementation of the code we have created which will complete the following:

- Everyone should be able to send messages to every other member through the server
- People can join a server without user limit
- The coordinator maintains state of group members by checking periodically how many of them live and informs active members about it so that they can update their state of members

The first technical requirement that we needed to tackle was creating the core program to allow users to communicate over a server. To begin with we started with creating the server. The programming language we chose to use was python. The reason for this was due to the fact that we all had a good understanding of this platform. To start with we referred to 'Socket Programming in Python Guide' (Jennings 2020) to use as an example that helps us create a foundation and produce the socket which will allow us to construct the communication system. Socket is an endpoint of a communication link of two programs that receives the data. Using socket, we created it to use ipv4 and made it use TCP. We then went on to make it so that the socket will bind to an IP and a port. Our program will get the local host and use the port 5555. The program then listens for clients. When a client connects it will take their address as well as

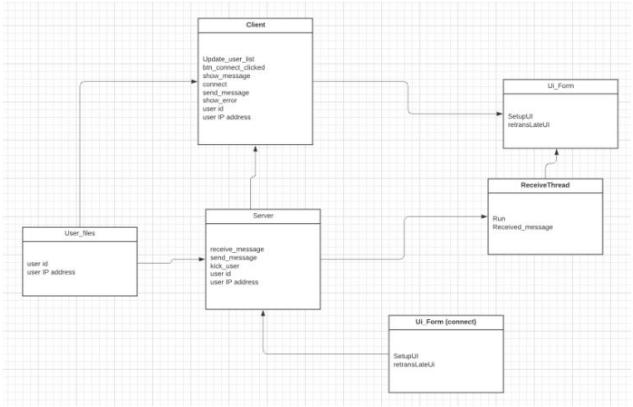
their name which the user will give when connecting to the server. The server will then store the user into a text file which will be used later to display users online for the GUI. Next, we worked on messages being received. To address one of the requirements of group state maintenance, we then continued to construct it.

For our program we decided to use a Graphical User Interface instead of a Command line interface as we thought it would be easier and more user-friendly. It's easier to display statistics for example how many people are online which would be displayed in a box, timestamp of when a message is sent by a user and who has left from the current server. The GUI allows the user to interact easier if they are not familiar with programming, each step to login is displayed in a box with text to inform the user of the current step.

As you can see from the output of the code the first thing that shows us is a GUI of a login page. In the login page it shows the host's name, the port and the name. What happens in that login is that you enter them and then you click on the connect button which will then allow you to connect to the client-server system.

This is what the client-server system looks like once you connect to it. We have structured the GUI in a sense that is user-friendly so that the users are able to understand how the GUI is used. So, first of all, once you connect to the client-server system, in the left box it shows how many users are online as well as who has joined the server. This is because users may be able to identify who is in the chat for it could be a stranger or someone they know. On the box located in the bottom right, you can be able to type your messages which helps communicating with users. The button below is the send button which will help you send the messages to other users. Finally, the box located above the message box shows the messages you and other users have sent as well as the time stamp on when the message has been sent.

Moreover, we have created a code that performs different actions if you are an admin. So, once the user logs in as an admin (mind you we coded it in a way that only one admin can login), The admin can be able to kick users out of the client-server. We decided to choose this function because there may be some users that may act inappropriately for example spamming or talking about subjects which are unrelated to the topic so that issue can be controlled by that user getting kicked out of the server.



The image above illustrates the UML class diagram and the basic hierarchical structure for the chat server program.

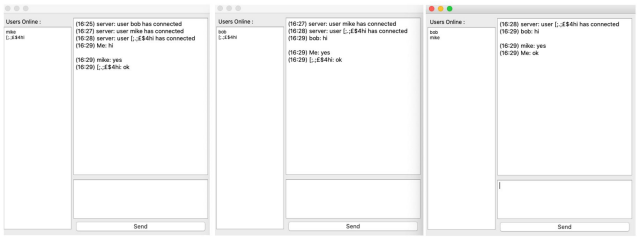
3 Analysis and Critical Discussion

If the situation occurs that the coordinator gets abnormally disconnected then we have designed the program so that it will select a replacement coordinator, which will then gain the permission of that role. Within the chat box of the GUI it will state that the coordinator has disconnected and will state who has now become the new coordinator. This is a step to address the requirement for fault tolerance.

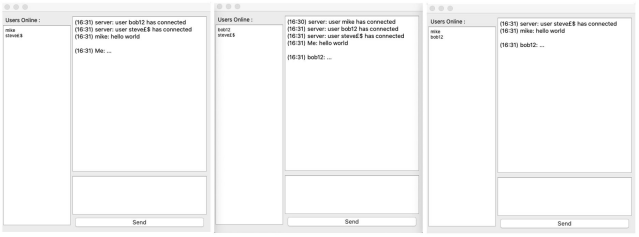
After we have designed the code, we have decided to test the functionality of the design to make sure that the code is be able to function well and if we do have any errors or crashes, we will talk about if these errors can be fixed or not.

In this test we tried consisted of 4 different members to join a live server and send a different message each in chat, this shows that the server would be correctly running also that other current server members would be able to see everyone's messages including the admin/coordinator.

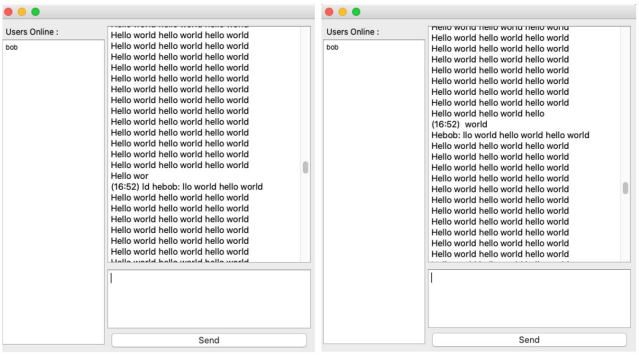
The next we did was to test the kicking ability of the current admin/coordinator, we had 1 member of the current server to be kicked which was successful as after the actions they were not in the server anymore.



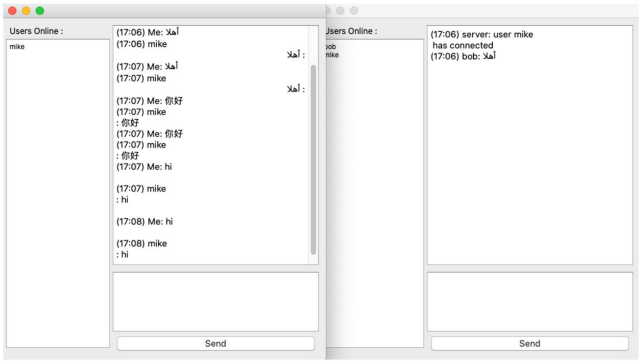
Special Characters: In this test we wanted to check you can be able to send messages with special signs. So what we did is that we sent messages with special characters and the result for that was successful because the messages was able to be read by other users.



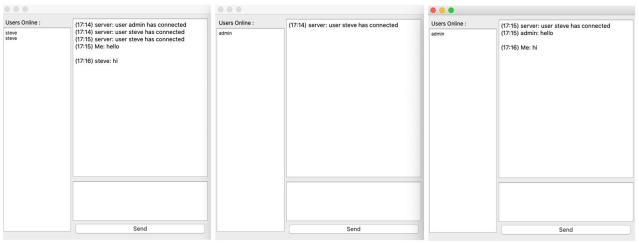
User ID with Special Characters: In this test we wanted to find out was typing in different characters and special signs into the ID of users, the test came out successful and allowed users to have different signs/characters without any issues. We also tested them typing to see if that might cause a crash however it didn't.



Typing lots of words: In this test we ran was that we wanted to know what would happen if typing lots of words in the program would crash or do something unusual, surprisingly there was no problem with the program and could handle a high number count of words.



Communicating in a Different Language: In this test what we ran is that we were interested to know what would happen if users communicated together using different languages. So, what we did was that we sent a message in Chinese. This resulted in GUI being broken to the point that you can't you can't send messages even though you can receive them.



Using the Same Name for ID: Finally we wanted to test is if two different users connected with the same name, the result was that the latest person with the same name joined only they would be able to receive any messages and if the person with the same name who joined first would not get any messages in addition you can only see one user with the same name online in the current server.

4 Conclusions

Overall, we believe our client-server system satisfies most of the requirements in the coursework specification in the circumstance of its technical features and functions. However, the main concerns with this client server communication system is the small bugs and minor issues we discovered as talked about in the analysis and critical discussion, due to the nature of these bugs it does not interfere with using the system regularly as it is still usable. These minor occurrences can be resolved although they may be very time consuming to find the source of these minor incidents.

It is important to recognize that the introduction of technology is an innovation in the IT environment as well as the corporate world because job processes will be difficult to implement and could take longer without the implementation of this technology.

ACKNOWLEDGEMENTS

We would like to thank our colleagues (which is basically each other) that has helped and supported us with the research that was made for developing the code as well as identifying and fixing the mistakes that has been made on the code.

REFERENCES

[1] Jennings, N., 2020. Socket Programming in Python (Guide). [online] Available at: https://realpython.com/python-sockets/

[2] Sharma, A., 2020. Unit Testing in Python (Guide). [online] DataCamp Community. Available at: https://www.datacamp.com/community/tutorials/unit-testing-python[Accessed 31 Mar.2021]