Report for multi-client server

Jingjie Weng, Yuzhao Cao and Yuan Gao

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

***Abstract*—This report aims to discuss how we implement a Graphical User Interface (GUI) based network distributed system for group-based client-server communication. This report will explain how we use the IDLE Python 3.8 environment to create and implement the group-based client-server.**

Key words: *(group-based client-server, GUI, socket, thread)*

I. Introduction

The aim of a group-based client-server is to provide accessibility from one or more remote clients. We used Python-based programming language to implement this group-based client-server. We mainly use the GUI (Graphical User Interface) to create the main window for the group-based client-server, socket, and thread to handle the network communication and execute tasks. GUI is an interactive component that enables the user to interact with icons such as labels, windows, buttons, menus, and more to carry out the commands and convey information and actions.

Socket library is to create servers and clients for the group-based client-server. It is a way of connecting two nodes on a network to communicate with each other. One socket(node) listens on a particular port at an IP, while the other socket reaches out to the other to form a connection. The server forms the listener socket while the client reaches out to the server. (GeeksforGeeks, 2020)

II. Design/implementation

We used the IDLE Python 3.8 environment to start to create and implement the tasks. IDLE is Python's Integrated Development and Learning Environment, and it is pure Python code, cross-platform, debugger, configuration, etc.

First of all, we started with the Graphical User Interface (GUI) component layout to design the main window. As we mentioned above, GUI allows the user to interact with the system. To create a GUI for the server, we placed a group of widgets in specific areas in the main window. The server window has placed one widget at the top of the window frame, which is "Connect." The "Connect" button allows the user to set up a connection with the server. Below the "Connect" button, we have placed two labels for displaying the host and IP address. Below the host and IP address, we have created a space for "Client List," the client list displays the user information, such as username, user ID, host, and IP address.

When we finished designing the GUI for the server application, we started to import the socket and thread, which is essential for handling network communication and executing multiple tasks.

We bind the server to the host address and port number, start listening for the user's connection. We want to accept new user connection requests while sending and receiving the message from an already connected user. We store and display the user information in the client list to keep track of all connected users. Therefore, we use the "thread" function to keep accepting new user connection requests, listen for messages to the user, print out the message, and store the user information such as user ID to the list. This function monitors the information of all connected users and removes users no longer connected to the server.

Here is a detail of how we use "thread" to achieve the above functions. When a new user connects to the server, it will send the user ID, port, and IP address. We use the "socket. recv" method to receives the user's information and use "socket. send" method to send the message to the server. Then we used a "while loop" to keep receiving/sending messages and update the user's status. However, if the user is no longer connected or executed the "Ctrl-C" command, the server will remove disconnected or quitted users. Otherwise, all users on the server can send and receive messages to every other member through the server.

After we completed the server application, we started to implement the client application. The user needs to connect to the server application first, and then they can receiving and sending messages to all other users on the server.

When the user clicks the "Connect," the client application will connect to the server and ask them to enter their details. Furthermore, we have created a client socket (this is similar to the server application). The client socket will initiate a connection request to the server through the IP address and port number. We also use "socket. send", "socket. recv," and "while loop" to allow the user to keep sending and receiving the message from the server. Once all the functions ran successfully, our users can start

sending and receiving messages to and from the server.

III. Analysis and Critical Discussion

After we finish create and implement both client and server code together. We start to run the code for the group-based client-server, the result shows that we have successfully created the group-based client-server.

When we run the server application, the small window with the window title “Server” will pop up. There is one button at the top of the server application window, which is “Start.” The host address and port number are listed below the “Start” button. When users press the “Start” button, the “Start” button will change to a “Connected” state, which means that the user is connected to the server.

In the server window, we will see a client list; when the new user joins the server, it will display the username with the user ID, port, and IP address. The server will keep displaying the username, user ID, port, and IP address as long as the new user continuously joining the server. Furthermore, if a user leaves the server or does not respond within 2 minutes, the server will remove the left and disconnected the user automatically and update the list. This provides the username, user ID, port, and IP address of existing users to the new users and the number of users on the server. However, if the member does not respond, it will inform the server about this, and the server will inform another user to update the list to the existing user. If the coordinator did not respond to the server or left, a random member will become the new coordinator.

To achieve the above server features, in our code, we created a class called “class serverInfo” inside the class, we define a “def receive” function, use “while loop”, to allow the server to receive request, store and broadcast the user details. Use the “if loop” to print the coordinator’s detail. To display in the current user and users’ detail by defining a “current_online” function. Furthermore, we define a “handle” function to broadcasting message, remove the left and disconnected the user, and appoint a new coordinator.

When we run the client application, there will also pop up a small window with the window title “Client.” There are three labels at the top of the client application, namely, “Name,” “Port,” and “Host,” each with a message box, and these message boxes will allow the user to input their username and port and IP address. Next to these labels, we placed two buttons, which are “Connect” and “Check Status,” the “Connect” button allows the user to connect to the server; the “Check Status” button will be only available when the user is a coordinator. We will see an instruction on how to exit the client window below these labels, and this provides information to the user that how they can exit the server. At the bottom of the client window, there is a message box with the “Send” button, which will allow the user to send messages to all other members in the server.

To run the client function, the user needs to enter the correct required details to the message box and press connect button to connect to the server. Then the user will be able to send and receive messages from other users on the server. However, if the user enters the wrong or leaves the port and the IP address is empties, the error warning will pop up.

Furthermore, when the first user joins the server, the client will inform the user and make the first joined user as coordinator, and the “Check Status” button will be available at the same time. The coordinator can check group members' status by clicking the “Check Status” button, and then the coordinator will able to see all the “current online users” information, and who is the current coordinator display in the client window. Also this information is available and update to all the members in the server. Therefore every member on the server can see the “current online user”.

To achieve the above client features, in our code, we created a class called “class clientCode” inside the class; we define a “def connect” function to allow the user to connect to the server, and if the user did not run the server application, the error message would pop up, this will remind the user to make sure to turn on the server first. We also define a function “def receive”, “def write”, and “def status” to send the welcome message to the coordinator; users can write and send a message to the server and allow the coordinator to check the currently online users. If the user did not enter the correct information, such as the wrong host and IP address, then the error message will pop up to remind the user to correct everything. Use thread to execute “def send” and “def check_status” function. “def check_coordinator” function is to make the “Check Status” button only available when the user is a coordinator, and the “def handler” function allows the user to use “Ctrl-C” command to exit the server.

We implement all features listed above by using GUI and unit test under the IDLE Python 3.8 environment. The testing goal is to ensure that all the features we created all run successfully, such as member can connect to the server by entering the correct port and IP address, the coordinator can check members’ status and other features.

We ran the unit test for both client and server applications. To carry the unit test, we need to import the “unittest” module into the IDLE Python 3.8 environment and run both client and server applications in the background. When the unit test runs successfully, in the server window, we will see a test user with an IP address and an unique ID connected to the server. The unit test result in Python showed ‘ok’, which means we have successfully passed the test.

We have also included fault tolerance in our group-based client-server. The fault tolerance prevents interruptions to the whole server due to a single failure in code. We manually created an error during the test stage, making the system unable to appoint a new coordinator randomly. Besides, if the user did not enter the correct port and IP address in the client window, the system will notify the user to check if the port and the IP address are being entered correctly or if the user is not connected to the server. Furthermore, if the IP and the port are already used, the user attempts to use it again, the server will notify the user with an error message. Under these circumstances, the users can still communicate with each other without having an error message.

When we run the configurations with the same IP address, we found a weakness in the code. In the client windows, any user connected to the server will be able to change the information displayed in the message box. This problem is not acceptable in this situation; if the user has changed the information in the message box, this will make the information on sent messages is unreliable. However, the results of the whole code will not be affected.

IV. Conclusions

This report proposes a “group-based client-server” application which is the motivation for this work. The report also outlines and clarifies how we designed, implemented, and tested the group-based client-server under the IDLE Python 3.8 environment. We have found some bugs in the code during the test stage, but we fixed all the bugs perfectly. However, some features and designs could be developed further in the future, such as a better GUI layout, which will give the user a better experience. We gathered pieces of knowledge and gained an unique understanding of creating a group-based client-server; this is a meaningful milestone for us to develop similar applications in the future.

References

Effiong, C., 2020. Learn Python by Building a Multi-user Group Chat GUI Application.
[online] Medium. Available at:
<https://levelup.gitconnected.com/learn-python-by-building-a-multi-user-group-chat-gui-application-af3fa1017689> [Accessed 20 March 2021].

GeeksforGeeks. 2020. Socket Programming in Python - GeeksforGeeks.
[online] Available at:
<https://www.geeksforgeeks.org/socket-programming-python/> [Accessed 11 March 2021]