Christopher Luong (z5309196)

Comp3331 Assignment Report

Program Design

The program was designed by using 2 main files (client.py and server.py) which both contain various functions within them to perform the tasks that are initiated inside the main while loop that runs until the server/client is closed. Multithreading is implemented in the server through the ClientThread class which connects with new clients on different ports on the same IP address, allowing multiple users to communicate with the server at once. This means the server can also communicate with multiple active clients.

Application Layer Message Format

The server sends messages based on the state/status of the user (e.g. inactive/active) and also in response to what messages the client sends to the server.
The client is constantly listening to the server for messages prompting their input or notifying the state of the client in the system through a non-blocking socket. A function was created to receive messages from the server as a stream since TCP is a stream-based protocol as opposed to message-based. The data stream is then separated into messages which are loaded into a message queue (acting as a buffer). One message is then retrieved from the queue and removed once it has been used by the client. If there is no data received, the receiver for the client will pause for 1 second and then attempt to retrieve data from the server. Input from the client is sent to the server in a message-based format since clients send less messages than the server so there is no need for a buffer on the server's side.

How the System Works

The user is required to log in through the client before being able to use the functions (commands). If the user inputs an invalid username, they are prompted to input a valid username indefinitely. Once they input a valid username, they are allowed a set number of password attempts for that username; and are blocked for 10 seconds if they fail to login within the allowed number of attempts. Sudden/Forced logouts are implemented when the user inputs a "Keyboard interrupt" (e.g. CTRL + C).
Once users are logged in, their status is changed to active and while they are active, they can use the commands offered by the server. Invalid arguments provided with the commands are

mostly handled on the client side, with some arguments that require checking the ‘database’ from the server being handled on the server side.
Databases include a dictionary on the server side containing key information on the users and a dictionary containing the existing separate chat rooms.
Responses and status/changes in the state of the system are displayed on the client’s and server’s terminal respectively.
Users can use 6 commands (BCM, ATU, SRB, SRM, RDM, and OUT) once they have logged in. UPD was not implemented.

Design Tradeoffs

Most of the functions and code were placed in the client and server files which could make it difficult to understand the functions since the files could get very large.
Although functions were used to extract repeated code to improve the readability of the code, there were still some functions that were similar to each other (e.g. functions concerned with broadcasting messages and sending separate room messages).
The method for receiving messages on the client side led to various cases that were dealt with individually with if-else statements which reduced the readability of the code.

Possible Improvements and Extensions (and how to realise them)

The removeUser function could be implemented more efficiently by only modifying the userlog.txt file after the logged-out user, as opposed to overwriting the whole file.
More files could have been created to shorten the length of the main client and server files, and the use of classes could have been beneficial since the client and server could be easily represented by classes with their own methods (e.g. commands get represented by methods in the server class).

Circumstances where the program may not work

If user input is sent to the server in extremely quick succession (i.e. spamming the username field with multiple inputs per second), the server may not send the response messages to the client fast enough.

Christopher Luong (z5309196)

Borrowed code

Multithreading in the server and starter code for the server and client were adapted from WebCMS in the following links:

https://webcms3.cse.unsw.edu.au/COMP3331/22T2/resources/75949

https://webcms3.cse.unsw.edu.au/COMP3331/22T2/resources/75946

Reading the last line in a file was adapted from

https://www.codingem.com/how-to-read-the-last-line-of-a-file-in-python/