COMP3331 Assignment Report

Program Design

In this assignment, a basic instant messaging application is created and written in Python 3.7. Source code from the course website by Wei Song is used for implementing multi-threading in Server.py. Some source code from another video tutorial called Simple TCP Chat Room in Python (https://www.youtube.com/watch?v=3UOyky9sEQY) is used in Client.py.

Server.py

The server is started by a user entering the command line arguments 'python3 Server.py port_number block_duration timeout_duration'. The user can close the server by pressing 'ctrl + C' any time and the program will terminate.
The server handles user command interpretation and also stores all user states.
Multi-threading is used so whenever a new client connects with the server, a new socket will be created for this certain client. If they log out, the old socket will be discarded. If they log in again, a new socket will be assigned to them.

Client.py

Multiple clients are supported to communicate with the server at the same time. A client can start a connection with the server (at the same port number) by entering the command line arguments 'python3 Client.py port_number'. A client can kill the connection by pressing 'ctrl + C' but an error will occur and be shown in the client's terminal. Whenever a client connects or disconnects from the server, the server will always be notified.
Two separate threads are used at the client side to receive the messages sent from the server and to send messages (requests) to the server. These two threads are started together and run simultaneously when a client connects to the server.
When I tried to implement disconnection from the server in the cases except logout (official disconnection), I noticed that it is hard to close two threads at the same time and I didn't find a way to close the receive thread and then notify the write thread. Instead, I closed the client socket whenever the receive thread receives messages that require client disconnection, and then the write thread can detect error and break the loop (but this might need the client to press 'enter'). I added a condition in the loop inside the write thread so whenever 'logout' is sent, the loop will break, but this does not solve the errors with disconnection from server under certain cases.

Authentication

Users need to login so that they can type in the other commands and send requests to the server. Sometimes the server lags and the user needs to press 'enter' twice after typing in 'login' in the terminal, and then a prompt asking for username/password will show up. The reason why this happens is yet to be determined.

After entering the password wrong for 3 consecutive times, the client will be notified that they have been blocked. However, this is not fully implemented due to some unknown bugs. The client will be disconnected from the server in this case.
If the username client entered is not stored in ‘credentials.txt’, the server will ask the client to enter a new password and the credentials of this new client will be appended to the end of ‘credentials.txt’.

Login and Logout

Every time a user successfully logs in or logs out, all other online users will receive a notification. This is achieved by broadcasting the login/logout message to all online users. A user cannot log into an account that has already been logged in, or has been blocked. In the former case, the user can still type in other commands such as ‘login’ again. In the later case, the user will be disconnected from the server.

Messaging and Broadcasting

A user that is logged in can message any other online users. The message will show in the recipient’s terminal in the format of ‘sender: message’. A user can also broadcast the message to all online users. A user cannot send a message to themselves, an error message will show in their terminal if they do so.
Blocking is not fully implemented, but if a user tries to block themselves, an error message will be sent to them.

Whoelse and Whoelsesince

This is not fully implemented but some structure code was written.

Timeout

Unfortunately, this is not fully implemented due to unexpected client disconnection from the server..

P2P Messaging

This is not implemented.

Data Structure Design

A class called ‘User’ is created to store all individual user data including username, password, block_duration, timeout, online, blocked, blocked_since, last_active. ‘Blocked’ stores a bool indicating whether the user is blocked by the server (if blocked == True, user cannot login). ‘Blocked_since’ keeps track of when the block duration started. Similarly, ‘online’ indicates whether the user is currently online (client socket is connected to server), and ‘last_active’ should store the time when the server last received a command from the user (which is when the user last sent a command to the server).

There are a couple of ‘global’ variables that are meant to be accessible across the whole script of Server.py as data storage, and they are initialised when the server is started. ‘Valid_users’ contains a list of users (User classes) that are created from the list ‘valid_usernames’. ‘Valid_usernames’ stores all the usernames that are written in ‘credentials.txt’ and any new username that is created. ‘Client_sockets’ stores a list of client sockets. ‘Blacklist’ should store a list of users that are blocked by a user for all valid users. A user should have their unique index based on their index in ‘valid_usernames’ (because of their credential position in ‘credentials.txt’), which means this index should give details about this user in any of the above lists mentioned. When a new user is created, its credentials/client socket etc. will be appended to the corresponding list, so the index should be unique for all users and adding new users will not affect the existing order of all other users.