COMP3331 Assignment 1 Report

Python version - 2.7.10

Program Design Approach and Working of the System:

I started by implementing a single server and client system.
To do this, on the server side, I opened a welcoming server socket with an IP of 0.0.0.0 (to allow both private and remote clients to connect) and listened for a client connection.
On the client side, I provided the server's IP with a port and sent a connection request which when accepted by the server creates a new socket b/w that client and server.

I then extended this approach to allow for multiple clients to connect to a server by using threading. Each client starts its own new thread as its connection is accepted by the server.
Each thread maintains its client's state (loggedIn, username etc). The server also maintains a list of client threads which are currently active and this list is updated if a user logs in or out.

I then implemented various functionality for the clients:

1) Authenticating user : implements login and a blocking feature if the user enters an incorrect password more than three times
2) Disconnect user : sends a 'Bye" message to client and disconnects gracefully
3) Broadcast : broadcasts a message to all active clients
4) Block user: Since a user may not necessarily have to be active to be blocked from another user, I decided to maintain a blockedFrom dictionary with key as username and value as a list of usernames that they are blocked from.
5) Unblock : is similar to blocked from and manipulates the 'blocked from' dictionary
6) Whoelsesince <time> : calculates the elapsed time since the user logged out and accordingly displays the users who were active within given time frame
7) Whoelse: prints a list of currently active users
8) Sending offline message : I decided to maintain a dictionary with key as username and value as a list of any message they receive when they are offline. When a user logs in, I check if this dictionary has any message for them and print accordingly
9) Message user : works by sending a message to the client thread associated with the given username

After implementing the above functionality, I added threading in the client side to allow for peer to peer messaging.
The general logic I used for this is that if a client A initiates peer to peer messaging with client B, the server sends this connection request to the specific client B who then opens a server socket and send a message back (containing port and ip) to the server asking to client A to open a connection on the same socket.

The server then sends the port, and the ip to client A who then accepts the connection by creating a socket with the given port on its side.
Now a connection is established b/w client A and B directly.

Application layer message format:

The application layer message format I'm using is a UTF encoded string. I encode the string before sending and decode before receiving it.
Both the client and the server send and receive this message and have multiple if(..) statements to handle the actions taken for each message received.
Since this is a basic version of an instant messaging simulator, using strings sufficed.

Design tradeoffs and Possible improvements:

This instant messaging application is limited in scope and scalability. One possible reason is for this is that I use several dictionaries to maintain state for user names. (Eg: a dictionary to store username and passwords, or to track user status if they are blocked or unblocked etc). Currently, since the number of username is small, this program runs fast. However, in a real world scenario where there could be millions of different usernames and passwords, tracking all of them by using a dictionary such as this would be inefficient and defeats the purpose of "instant" messaging.
A possible improvement or an extension to this would be to store this user data in a database such sql or mongo and fetch user data from there.

Secondly, on the subject of testing this program for multiple remote clients, I tested this code on my machine by providing my own private IP address to the as an argument to the Client.py program. However, if run from a different machine, it is possible that we may run into issues of routability or a firewall.
Hence this was difficult to test with a remote client running on a different machine.
A possible improvement would be to carry out more thorough testing of this app with remote clients on different systems.

Overall, this project served it's purpose in that it taught me a lot of the intricacies of reliable socket programming and provides relatively clean and reliable application for instant messaging.