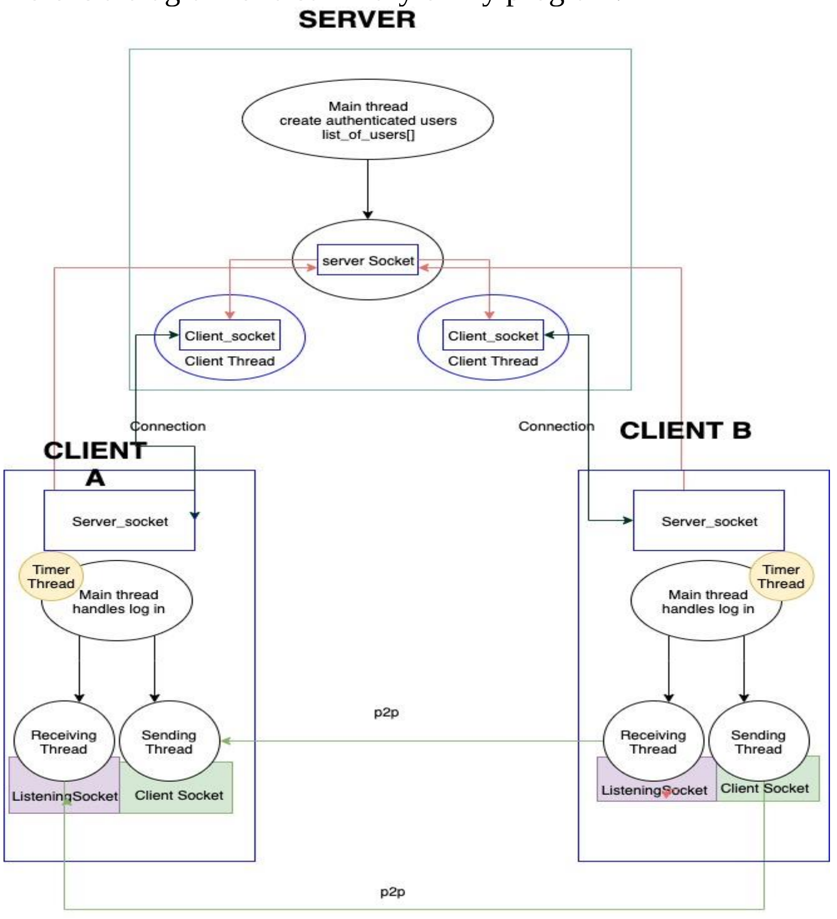
Report

I use python 3 for this program. I have implemented up to Part 3.3 Commands supported by the client and some of part 3.4 P2P Messaging.

Program Design

Here is a diagram of a summary of my program:



- Each user is a user object which stores values such as username, password, log in history, etc.
- Server contains a list of user objects to keep track whether it is online
- Server creates a thread to each user
- Each online user has a thread for receiving/listening and sending
- I user timer threading to log out user who are inactive
- Overall I have 6 files: Each a main file for server and client, a file to store structure of user class, afterlogin.py handles commands used after log in and command_functions has all the commands it needs, lastly threads_client to handle listening and receiving, this include both p2p and server-client connections.

  Normal Messaging:
  - Delayed (offline) messages goes into a list within the user object, which is appended based on the time it is sent.
  - Blocking and unblocking is tracked using a block[] list in the user object

- Time out of inactive users: Set a timer for each time a user is last active and cancels the timer the next time user prompts a command

P2P Messaging:

- Client side has a class called ThreadManager which stores all sockets/server it is connected to
- using select.select to filter out non-used sockets
- Server triggers the initator_client to start its port to listen
- then Server sends the initiator's listening port and address to the receiver so that it can "knock" on initiator's port

Message Format:

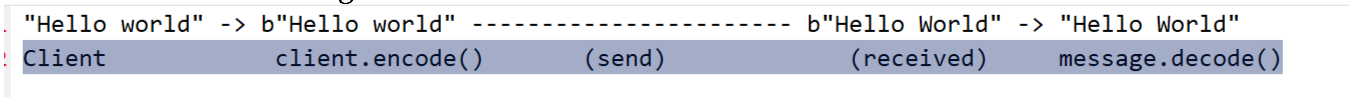
- All messages are in string format, encoded to send and decoded to receives

Class and Object:

- 2 classes I have is User class in user.py and ThreadManager class in threads_client.py

Application Layer Message Format

Messages are string format, which is passed by its bits using decode() in between server and client. It looks like the diagram as below:



```
"Hello world" -> b"Hello world" ----------------------- b"Hello World" -> "Hello World"
Client             client.encode()       (send)              (received)      message.decode()
```

To receive messages, I use message = socket.recv(2048).decode()
and then I split the message if needed, to separate the commands and informations required

To send messages, I use socket.send(message.encode()) where message is in String form, groups of characters separated by space.

How my system works

Server:

1. When server runs, it starts off by loading all user details from credential.txt. Each possible user is stored in a user class, containing its other information such as last log in time and whether it is online.
2. Server allows two clients to talk to each other or initiate p2p connection
3.

Client:

1. Client program prompts username and input to be inputted by the user
2. If username is not stated in credential.txt, the program will keep on prompting user
3. client has two threads, one for sending and one for receiving

Trade Offs

In order to be more flexible with the type of string I want to pass in, and the length of information I want to transfer between sockets, I use the function decode and encode for the majority of my assignment to transfer information between client and server, this will have security issues such as it is prone to injection attacks, adding more information into the string and modification in between the transport process. But since this is TCP, I'm assuming that it has a safe layer. I also set a limit of receive as 2048 bytes, which is a lot of bytes for simple strings that I am using. This also allows huge flexibility and easy implementation in terms of coding. There is wide range of strings I can add and use as a command.

Improvements

- Secure formatting - use a dump function to format strings as JSON instead of purely strings
- Having many strings and white spaces, I realise stripping each string before sending them to another socket would be good for in case there is trailing white space or next line
- If I have considered timer better, I would make a signal and handler to implement timeout to give better and reliable retransmission (but this is not accounted in spec)
- My P2P program sometimes does not work, but it always does a valid check if the client the initiator (client) is hoping to connect to another client (checking is working correctly)
- Other than that, I think my program works well for the commands in Part 3.3