REPORT for Messenger

Program Design

The whole program should only contain:

- Client.py: the client mostly contains of just printing all the messages sent by the server, while also having some functions and logic such as
    - Authentication = for checking if the username and password is correct
    - Threading = it uses threading to support multiple clients being connected to the server simultaneously.
    - Peer-to-peer = send video file to another active user directly via UDP socket
- Server.py: the server contains all the logic possible, and send all the messages back to the clients to be output.
    - Authentication = has all the logic for checking if the user and password matches.
    - Command = this command will handle all text message commands (such as message to, create group and more).
- Overall, it has a nice and efficient design as it uses functions to further simplify things.

Note that I am using python3 to run all the commands.

The application layer message format

Messages are sent as both JSON format and a normal string.

- The JSON format will be used to breaking up the functions for authentication or login, for commands, and for displaying other text which doesn't match any of the options. This JSON format will be sent before the clients send the real message for what they want to do.
- The normal string will be mainly used for the commands and checking the authentication as USERNAME and PASSWORD.

For example, at the start of the clients, the client program will automatically send all the message as JSON format with the type of: `{'type': "command"}`. The server will then be notified and run the functions for authentication logic. The clients can then start typing their input as string format message, and this will be sent back to the server to be checked. The logic goes the same way for the commands function.

How my system works

To start the servers: { python3 client.py server_IP server_port }

It should look like:

```
z5394328@vx18:~/COMP3331/ass1$ python3 server.py 5000 5

=== Server is running ===
=== Waiting for connection request from clients...===
```

For each client: { python3 client.py server_IP server_port client_udp_server_port }

As soon as the server starts, it creates threads and listen to the client's response to support multiple clients.

On the other hand, the client will also create threads to support the private message and other communication between clients. It will also have a receiver threading to support the UDP socket for peer-to-peer communication without the involve of server.

The client will then start to implement the authentication, followed by commands and the logic for the peer-to-peer communication.

The server will response to the output from the user and respond to the client as a message.

Here are some of my simplified logic for commands:

- /msgto: send the message directly to the receiver if they exist
- /activeuser: Using the "clients" dictionary to check if they are active. I then use the userlog.txt file to get their details.
- /creategroup: created 2 dictionaries, "group" for the joined member and "inviteGroups" for the invited users.
- /joingroup: simply add the user to the "groups" dictionary
- /groupmsg: Similar logic to /msgto command, however, I need to loop through all the members inside the "inviteGroups" dictionary to send all the messages.
- /logout: I simply erased the user from the active clients dictionary, and remove the user from the userlog.txt file.
- /p2pvideo: it is implemented in the clients. However, if the server get the message, then do not response anything. This involves modifying the /activeuser command to give all the username and their udp port number to the client (to check if the user exist). I then created UDP receiver threading that listens to both servers and clients simultaneously.

Design trade-offs considered and made

- For the messages, I used to implement it all as strings. However, there are many bugs occurred due to the server receiving multiple messages of the same type at the same time, causing an error. Hence, my tutor suggested me to send it as a JSON format so that the server can differentiate them.
- When calling the /activeuser, my program should also send all the clients and their UDP port number. Creating a dictionary which contain this and sending them to the client would be efficient than the try and except method I'm doing at the `receive_thread` function to get all the clients and their UDP port number. However, this creates problem, as the client will ask the server for another message after they receive the dictionary. Hence, I use this try and except method to get all the UDP port number.

Possible improvements and extensions to your program

- When a command is types, and the server need to both printing and send a message back to the client, this logic can be implemented as a separate function. I noticed that it has a lot of repetition.
- All the text message commands can be break up into multiple different functions again, to even make it more readable

Program does not work under any circumstances

The program works at most circumstances. However, if the server is unable to run, try using different port numbers.

Segments of code that you have borrowed from the Web or other books.

The client.py and server.py is used from the sample in the COMP3331 assignment websites.