Comp3331 Assignment Report - Bofei Wang

Program Design

Structure:
- client *folder where client resides* - client.py *client entry file* - server *folder where server resides* - credentials.txt *credentials to feed to server* - server.py *server entry file* - UserManager.py *auxiliary file to help server delegating tasks*

Application Layer Message Format

Messages are communicated as json format.

Generally, an `action` is a must in a message to indicate the intention of a message. Depends on intentions of a message, other fields in the json are different.

For example in when `action` is set to `login`, server's reply message will include a `status` field to tell the client if it is authenticated or not.

Another example, when server handles a `action: broadcast` requested by a user, it sends a `action: receive_message` json to some other users, whereas when it handles a `action: receive_broadcast`, it sends a `action: receive_message` to the user who should receive the message.

How Your System Works

Quick command reference:

Terminal 1: server `shell script cd server python3.7 server.py 12346 10 60`

Terminal 2: client `shell script cd client python3.7 client.py localhost 12346`

On server starts, it starts to listen to every new connections and create a thread for it, and the new thread will listen to every incoming data requested by client. Generally all the messages will be checked by UserManager and reply message will be send back to client immediately. Only when a user is not online, the messages are temporarily pushed to a pending list and another thread will continuously check the pending list and process these message when the user is online.

On client starts, it establish a TCP connection to the server and creates two threads - one for displaying server incoming messages and another for handling user inputs and send the messages to the server.

P2P is implemented by client retrieving address and port number of another client and establish a TCP connection directly with another user.

Design trade offs

One of the trade off is to use multi-processing or multi-threading. Multi-threading is chosen because it is easier to manager the communications between threads.

Another trade off is the format of the messages. json is chosen because it is light weight and is best fit for a small scaled assignment.

One more trade off is the number of manager classes to create. Only a UserManager is finally implemented and it turns out that it is sufficient. Initially I was thinking to create classes like CredentialManager, MessageManager, TimeoutManager etc., but because of the functionality of the messaging application are quite limited, one UserManager is sufficient for all the functions.

Possible Improvements

Efficiencies of the messaging application can be improved. Currently server checks input in every client thread in a while loop without pausing at all and updates users every 0.1 seconds. It is very recourse-wasteful. A better implementation is to use event driven programming where it only updates or runs when there is a new events, instead of looping infinitely.

Extensions And How You Could Realise Them

P2P messaging is implemented.

Does Not Work Under Any Particular Circumstances

Yay - it works under most of common circumstances on CSE machine!

If you are unable to run the server in a particular port, try a different one - that port may be busy.

Segments Of Code That You Have Borrowed

- `server.py` and `client.py` is developed based on the multi-threading codes provided by COMP3331.
- https://stackoverflow.com/a/4653306/12208789 at client.py to prevent having the prompt overwritten by other threads in Python