COMP3331 Computer Networks

Assignment Report

Written by Yutong, Ji.

zid: z5191511

Program Design

This assignment is using python3.7, which can be run via command `python3 server.py [server_port] [block_duration] [timeout]` for server and `python3 client.py localhost [server_port]` for client.

For this project, there are four python files and a txt file, namely `server.py`, `client.py`, `user.py` and `help_functions.py` and `credientials.txt`.

The `server.py` implements most of server functionality and it is using python OOP together with multithreading to handle multiple connections with clients and realise the communication between clients and server. Generally, the threads are allocated accrodingly. The server itself uses two threads without connecting to any client, one for the timer and another one to listen for connections from clients. Whenever a client is accepted from the server, a new thread is created to deal with the commnuication with this client, if this client leaves the chat, this thread associated with the client will be closed.

The `client.py` implements the client side application, which supports both Client-server and P2P communications. Mulit-threading is also applied to establish P2P connections. The client is easy to set up, after connecting to server, it can start messging to server through command line interface. All the messaged typed in will be processed and sent to server. However, the client will only do minor message process which are related to P2P communications, other commands related to server will be processed at server side.

The `user.py` is a help file for server. It only has a user class which stores all relevant information of a client object and some methods to manipulate the data. It makes easier for the server to store, get and handle requests from clients.

The `help_functions.py` only has few functions help to transform data format or user finding.

Message format

Message format

client to server:

`<command> <name(if require)> <other information(if necessary)>`

Sever to client:

- error
  - `Error: <error detail>`
- infomation
  - `[info] [message]`
- others
  - `System: <other information>`

Description

All the messages sending between client and server are a single string, different parts are separated by space. To be more specific, the first word of the string always be an indicator. For instance, if a message `message Yoda hi` is sent to server, the first word `message` tells the server that it is asking for a messaging service. Similarly, if it starts with `broadcast`, it is a broadcast message. However, if the first word is not recognised by the server, an error message will be sent back to the client. To inform the client there is an error message, all error messages will begin with `Error`.

Warnings and some other messages sent from server will start with `System:` except messages generated from `broadcast` (not the login, logout message) and `message` commands.

System description

Server:

After the server successfully setup, it is listening on a known port, waiting for clients to connect. After establishing connection with a client, it sends `Username:` to require clients to login. The server do client's username validation first, if the given user is currently online or an unknown user, corresponding error messages will be sent and displayed at client's window. Then it sends `Password:`. If the password received does not match given username, the client will be asked to enter BOTH username and password again. After three failure attempts of a particular user, the account is blocked for `block_time` duration. If the client tries to login at this time, server will send a message to inform the user that he has been blocked.

Once a client successfully logged in, he can send command to server using the command format mentioned above. Server will assume messaged received by clients are in valid format. The server maintains a list of all registered users and a list of online users and they will be kept updating whenever the client's status changes. Every second, the server will check all online clients to logout inactive clients. A clients is marked as inactive if the gap between last command sending time and current time is greater than `time_out`. The server logout an user through closing connection.

Clients:

Clients only store his connecting peers and his username. Clients will not do much command validation or message processing, it only process information messages from server. That is to say, the first word of the whole message is a known special word.

After a client's successfully login, he will set up a port for listening and sending the port number to server.

User:

User is a class to help store clients information at server side, it has `username, password, connection socket, login time, last active time, block, black list, address, port number, offline messages, number of login attemps` and getters, setters as well as some other check functions.

Design issues and future improvements

Design issues:

Due to the unpredicted behavior of python multithreading, all the threads may not be fully synchronised, which causes an issue of P2P communication. Since I used a thread to listen from server, a thread to listen from stand input and each peer uses a thread. If all the threads are synchronised, it works well. But sometimes it has an asynchronised peer which may results in other threads waiting for it to run.

Communications between server and clients are using strings which are apparently not scalable.

Future Improvements:

Using C or Java to deal with multi-threading issue.

Using pickles instead of strings can make communication more clear and extendable.