```
- Multi-User Chat Application in Python
Project by:
- Guruvansh Singh Bhatia
- Gaurav Madkaikar

## Files and directories:
`` client.py
`` server.py
`` database/
`` --> userbase.json
`` --> backup/
`` --> --> chat_backup.txt
```

Function Specification

- ## Client Side

  client_login(): Take user credentials and send them to the server for authentication.

  client_signup(): Take user credentials and send them to the server for registration.

  launch_terminal(): Handle any incoming commands through the command line once the GUI window is launched. Specifically used for exitting the client program.

  send_data(): Handles sending of messages from the client to the server. Messages are send in packets of size 1024 bytes each.

  Message Format

  Header(4 bytes)|Preamble(7 bytes)|Data(1013 bytes)

receive_data(): Receives messages from the server and displays it on the GUI interface.

system_instruction(): Function to display system instructions(for e.g., when a user exits) on each client GUI window.

close_GUI_window(): This function is to be called just before the GUI interface is closed. It notifies the server to send a system instruction indicating the end of that client.

quit_GUI(): Quit the GUI interface.

- **Server Side**

accept_client(): Assigns a new thread to each incoming client

handle_client(): For a particular client 3 choices are provided:-

1. **Login: Username and password are checked in the authentication() function. Wide range of exception handling is implemented.**
2. **Signup : signup() function is called**
3. **Quit : Client instance is removed from user_client[]. Client socket is closed.**

authentication(): Checks if the passed username and password matches with any of the records stored in *userbase.json*.

signup(): An instance of client username and password is stored in *userbase.json*.

broadcast_global(): Sends a message to each of the active client sockets.

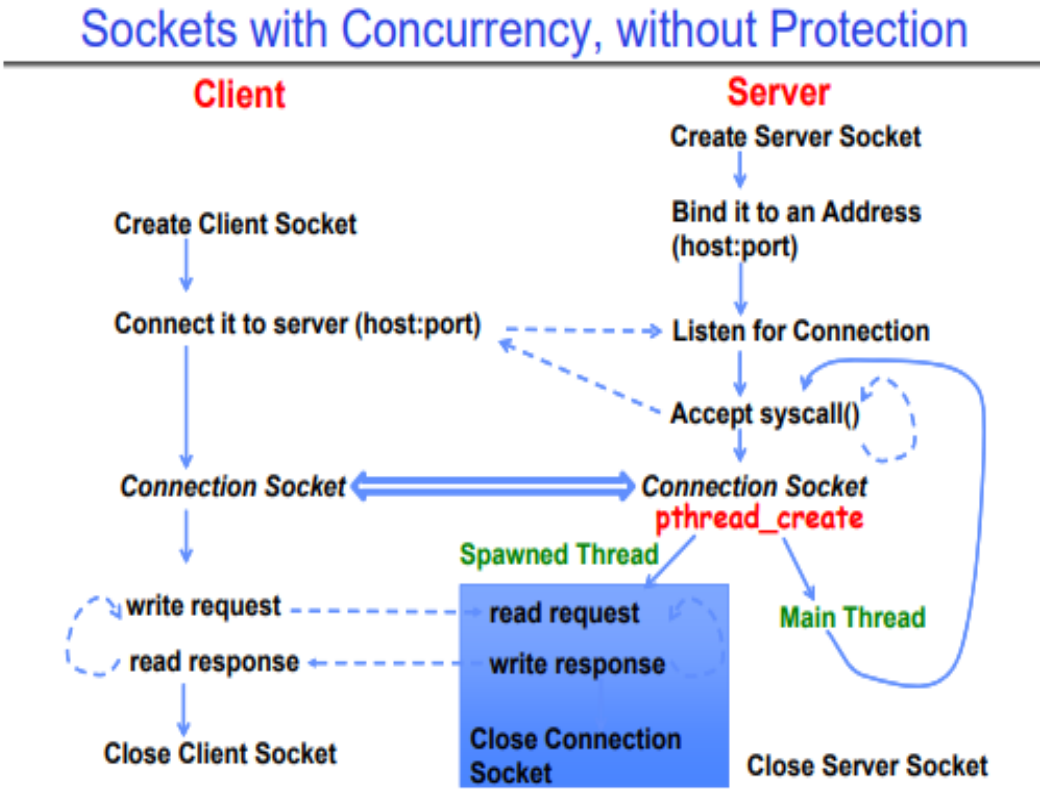
broadcast_selective():Sends a message to particular client socket. Send the status of completion of functions like signup() and client_login().
A raw message is passed into broadcast_selective() function which is 'Y' or 'N' signifying Yes and No. Now, broadcast_selective() adds "<SYSTEM>" to the raw message and sends the bytes along with message length as header to the client socket.

update_upon_login(): Checks for the chat backup if available and displays it.

logging(): Log the current message in the corresponding log file. Log files are ordered by dates.

chat_backup(): Open chat_backup.txt for storing any new messages.

Operational Flow



Once a TCP connection is established between the client and the server, the flow of control corresponding to both the client and the server files is described as follows:

Given three options of 1. Chatroom, 2. SignUp and 3. Quit, the client is free to navigate through any of these. The client then sends this valid choice to the server. Depending on the chosen choice the client/server will perform the following functions:

1) Chatroom

Client: Asks for user credentials and sends them to the server

Server: Authenticates the user credentials received from the client with the details stored in the user database and sends back the login status (using broadcast_selective())

Client: If the client is successfully authenticated, opens a GUI interface for further message sharing, consequently launch two threads and wait until these threads are executed. The two threads are described as follows:

1) receive_thread(): Receive messages from the server that are to be displayed on the GUI screen.
2) interaction_thread(): A separate thread to handle commands like EXIT from the terminal while the user is interacting through the GUI.

Client (GUI): Send messages to the server using the **Send** button (in packets of 1024 bytes)

Server: Receive messages from the client and broadcast it to all other clients. If the clients wishes to EXIT, broadcast an exit message selectively.

2) SignUp

Client: Asks for user credentials and sends them to the server

Server: Receives the credentials from the client, stores them in the user database returns the status.

3) Quit

Client: Sends a <EXIT> message to the server indicating the end of the client program.

Server: Removes the client descriptor from the current list of active clients and sends a global broadcast to other clients.

Client: Displays the global broadcast and closes the socket connection.

We have tried keeping buffer size in accordance to the message length to minimize delay. We always attach header to a message which is length of the message and pass it as buffer length.

For each message/command, log files will be updated to store the sequence of operations. The client can exit the program anytime by using the <EXIT> command via the GUI or the command line. Similarly, an user can interact with the server program through the command line by using the following commands:

1) **<EXIT>:** Aborts the server program.
2) **<ACTIVE USERS>:** Displays a list of the currently active users.
3) **<DELETE CHAT BACKUP>:** Delete the chat_backup.txt file storing history of all past messages.

Steps to run the project

Run both the server as well as client concurrently. Ensure that the server must be running before the client since the type of connection is TCP.

- python client.py
- python server.py <**hostname**> <**port**> (Eg. python server.py 127.0.0.1 21000)

If server and client run on the same computer, use hostname as localhost (127.0.0.1)