Let's Chat

Objective:

In this project, you will build a client and a server application where client-to-client communication, excluding a server, is possible. The server application will be used to share contact information about active clients and broadcast messages on behalf of a client. Clients can connect to the server, authenticate using a password, and see the active client list. It can choose any active client to connect with, and the remote client will be asked to approve the new connection. Once a connection is successful, clients can send messages between themselves without the server. Besides that, a client can request the server to broadcast a message to every client who is currently active. All clients who are active within 30 minutes should receive the broadcast message.

Description:

There will be two different applications: a server and a client.
The server app will listen to port 4400 and simultaneously allow at least 50 connections. The server tasks are listed below:

- Create socket:
  - It will create a TCP and UDP socket. The TCP socket is used to communicate
  - with clients, and the UDP socket is used to broadcast messages.
  - It will use port 4400 and at least allow 50 connections at a time, using either multiprocessing/multithreading.
  - If there are more than 50 connections at some time, TCP will ignore the new connections, and clients will retry (automatically) at least 5 more times in the next 10 minutes.
- A database:
  - The server will create a database (MySQL) in the backend once it is running. The database will be used to store and retrieve client authentication.
  - The database will have a table with the client username and password (in hashed).
- Client connection to the server:
  - Once the client connects to the server, the server will respond with a question with login/signup. It can start with the client saying HELLO to the server at first.

- If the client chooses signup, the server will reply back asking new username and password (asked twice, including confirming the password).
- The server will check the database, and if found the entry is non-conflicting (username), it will store the username and password (in hashed).
- The server then replies to clients to log in and asks for the username and password.
- If the client chooses login, then the server will ask for the username and password.
    - If the authentication fails, the server will allow the client two more tries and then close the connection.
    - If the authentication passes, the server will allow the client with two options: list of active clients or broadcast a message.
- List of active clients:
    - The server will show all active clients (only client username and IP address). All P2P connections can use the same port number.
    - Every active client will already have the receiver TCP socket enabled.
- The client can request the server to let connect to one of the clients by typing their IP address (and port).
    - If that is a valid request, a peer-to-peer connection will be open between clients.
    - Otherwise, the server will penalize the client by terminating their connection.
    - If a P2P connection is complete, the server connection can be closed (for both peers).
- Broadcast a message:
    - If a client chooses to broadcast a message, the server will use its UDP socket to broadcast the message.
    - The server asks the client to type the message and then broadcast it on behalf of the client.
    - The broadcast message, by default, will include the sender's information (client username) and date/time (broadcast).
    - The server will keep broadcast messages for up to 30 mins. After that, the broadcast will be dropped.

The client app will connect to the server (fixed IP address and port 4400). The client app will have the following applications:

- Create Socket:
  - The client will start with creating a client-side TCP socket and connect it to the server.
  - The client will also create a new TCP socket (as the receiver for the P2P). A P2P connection completion will include a sender TCP socket to be active.
  - Once the P2P connection is complete, all other connections can be closed (TCP server and UDP broadcast).
- P2P messaging:
  - The client will use a new TCP (persistent) connection to send messages.
  - The message would be regular English text.
  - The client can disconnect the P2P connection using a special command (ctrl+c).
  - The remote client will receive a connection request and will explicitly be asked (sender client information will be shown) to approve by its client-side app.
- Receive broadcast message:
  - The client will receive the broadcast message from the server.
  - However, if it receives the same message (same client and timestamp) multiple times, it will ignore the laters to notify the user.
- The client will also have a UDP socket (to receive the broadcast message) open with the server once it is authenticated by the server.
- Connection status:
  - Since authentication, clients will always maintain their TCP and UDP connection with the server. It could use multiprocessing/multithreading for this purpose.
  - When there is no P2P connection active, the client can request the server to close the connection using a special command (ctrl+x).
  - The server will automatically drop an idle client (no P2P activity in 1 hour). A timeout message will be sent to the client.

Note:

- *You need to submit your code written in C/C++ and for Linux distribution. No other language is acceptable for this assignment.*
- *Either application should not have any segmentation fault. Such cases will be penalized.*

- *You should also include a Dockerfile and docker-compose.yml for your project.*
    - *There should be 3 clients and a server container connected to the same virtual network.*
- *A Readme.md file should be submitted together with the code to guide:*
    - *How to build (use Makefiles) your project.*
    - *How to use the applications.*
    - *If any libraries have been used (list them).*
- *Your code needs to be properly formatted and expected to have proper comments.*
- *You are free to use the resources provided by the instructor.*
    - *If you are using any external resource, you will have to take responsibility of that if there is a crash found or feature mismatch.*
- *This is an individual assignment; no teamwork is allowed. If found any evidence, academic actions will be taken.*
- *If you have any confusion, ask the TA/Instructor. Don't discuss with your classmates about solutions (this is how misinformation spreads).*

LetsChat_Student_Name.zip

- src
  - server
    - server.c
    - ...
  - client
    - client.c
    - ...
  - Makefile
  - Dockerfile
- Readme.md
- docker-compose.yml