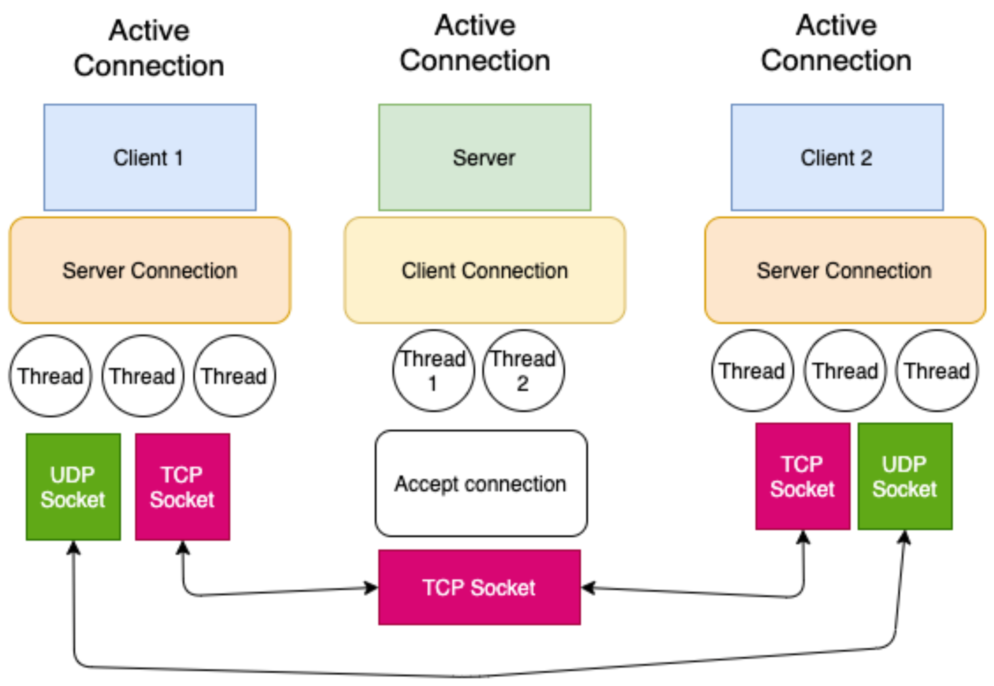
Program Design

Language: Python 3.7.3 CSE Machine



Program Flow

Server

1. Create a TCP socket
2. Listen for socket connections requests from client/s
3. Run server by setting an active socket and switch the flag is_running = True, create an empty clients_connection, active_users and lockedout_users list.
4. If there is a connection request from a client, accept that new connection and add it to the clients connection list.
5. Create a new thread for each client.
6. In each of the new thread, keep the connection open and receive the message from a client.
7. Authenticate a user before entering Toom. User's have x times login attempts until their accounts get locked out.
8. In the case of successfully authentication, the client will perform the seven available commands to communicate with the server.

Client

1. Create a TCP socket.
2. Request a socket connection to the server.
3. When the client is active, the client will keep listening and receive responses back from the server.
4. Once a client has passed the authentication step, a UDP socket will be created and 3 threads will run concurrently. The first thread is for receiving the command, second thread is to detect shut down status from the server and the third is the open of UDP socket for P2P interaction.
5. The client enters the command and send and receive messages with the server via the socket.
6. If the client receives a message from the server "[DISCONNECTED] Server disconnected…" the program will exit immediately all for clients.

The application layer message format

Command	Server	Client
0. Authentication		
Case 0.1 Login Success	> [NEW CONNECTION] {username} connected. > [LOG] {username} logged to userlog.txt	***************** Hello, {username}! Welcome to Toom *****************
Case 0.2 Login with user who already has an active session	> [STATUS] The username {username} is currently active. Please enter a different username.	> [STATUS] {username} already has an active session.
Case 0.3 Login with user with an invalid username	> [STATUS] {username} is an invalid username	> [STATUS] The username {username} is invalid. Please enter a different name.
Case 0.4 Login with an incorrect password	> [STATUS] {username} login attempts failed.	> Invalid Password. Please try again.
Case 0.5 Login with an incorrect password (exceeded the number of consecutive failed attempts)	> [STATUS] {username} account has been blocked for 10 seconds.	> Invalid Password. Your account has been blocked. Please try again later.
Case 0.6 Login attempt when account is blocked for 10 seconds	[STATUS] {username} is initiating a connection...	> Your account is blocked due to multiple login failures. Please try again later.
Case 0.7 Login attempt (with correct credentials) after account is blocked for 10 seconds	> [NEW CONNECTION] {username} connected. > [LOG] {username} logged to userlog.txt	***************** Hello, {username}! Welcome to Toom *****************
1. MSG *MSG <message>*	> {username} issued MSG command.	
Case 1.1 Successfully posted a message	> {username} posted MSG #{message_number} "{message} at {timestamp}	> Message #{message_no} posted at {timestamp}
Case 1.2 When a user tries to enter an empty body message		> A MSG body cannot be empty. Please try again.
2. DLT *DLT <message_no> <timestamp>*	> {username} issued DLT command.	
Case 2.1 Successfully deleted a message	> {username} deleted MSG #{message_no} "{message}" at {timestamp}	> MSG #{message_no} deleted at {timestamp}
Case 2.2 User enters message number that does not exist in the messagelog file	>{username}: Message number #{message_number} does not exist on the messagelog.txt file.	> Message number #{message_number} does not exist on the messagelog.txt file.
Case 2.3 User tries deleting message that does not belong to them	> {username} attempts to delete MSG #{message_number} at {timestamp}. Authorisation fails.	> Unauthorised to delete message #{message_number}.
Case 2.4 User tries deleting a message when the messagelog file does not exist	> {username}: No message to delete.	> No message to delete.
3. EDT *EDT <message_no> <timestamp> <new_message>*	> {username} issued EDT command.	
Case 3.1 Successfully edits a message	> {username} edited MSG #{message_no} "{new_message}" at {timestamp}.	> MSG #{message_no} edited at {timestamp}.
Case 3.2 User enters a message number that does not exist in the messagelog file	> {username}: Message number #{message_no} does not exist on the messagelog.txt file.	> Message number #{message_no} does not exist on the messagelog.txt file.

Case 3.3 User tries editing message that does not belong to them	> {username} attempts to edit MSG #{message_no} at {timestamp}. Authorisation fails.	> Unauthorised to edit message #{message_no}.
4. RDM *RDM <timestamp>*	> {username} issued RDM command.	
Case 4.1 Successfully reads messages	> Return messages #{message_no} {username}: "{message}" {posted/edited} at {timestamp}. ….	> #{message_no}; {user}: "{message}" {posted/edited} at {timestamp}. ….
Case 4.2 User enters the timestamp where the messages are not available	> {username}: No new message	> No new message
5. ATU *ATU*	> {username} issued ATU command.	
Case 5.1 Successfully logs other active users who are currently in the session.	> Return active user list: {username};{ip_address};{udp_port}; active since {timestamp}. ….	> {username}, {ip_address}, {udp_port}, active since {timestamp}. ….
Case 5.2 Logs when no other active users currently online	> No other active user.	> No other active user.
6. OUT *OUT*	> {username} logout	> Bye, {username}!

Command	Client1 (Presenter Client)	Client2 (Audience client)
7. UPD		
Case 7.1 Successful sends file to another client	> {filename}.mp4 has been uploaded	> Received {filename}.mp4 from {presenter_client_name}
Case 7.2 Attempts to send a file when the audience client is offline	> {audience_client_name}is currently offline. {filename}.mp4 not sent.	
Case 7.3 Attempts to send a non-existing file to an online audience client	> File Not Found! Please ensure that {filename}.mp4 exists in your directory.	

Design trade-offs

1. **The shutdown detection function.** This new thread on the client side is used to check whether or not the server is still alive and active. As a trade-off for this that it will definitely increase traffic in the network but at the same time the idle clients who are online could benefit the detection of shutdown. Once the server closes its socket, all of the connected clients are terminated, and the program will exit.
2. **Video file sharing via UDP.** There will be packet loss and there is no guaranteed that the audience client will receive the file in full i.e., corrupted file.

Possible improvements and extensions

In the current design the video file sharing is done via UDP so that the students can practice implementing both the UDP and TCP socket. For the extension of this program, I'd like to refactor the file sending in TCP instead so that I am guaranteed that the audience client receives the file as intended as it is more reliable than the UDP implementation.