CS 6543 – Fall 2024

Assign02 – Due Date: check Canvas -

Total 100 points.

Using the submission link in Canvas, you must submit your solutions as a single .zip file abc123_assing02.zip (please remove executable file and include include a makefile or instructions to compile and create executables). Make sure your program works under our FOX xx machines. Also include comments and extra printfs so we can easily follow your programs execution/logic when grading! If you work with a partner, name the files as abc123A_abc123B_assing2.zip, where abc123 is your own ids, one submission is enough.

You are allowed to use Gen AI to generate code for little components that you can mearge to solve the whole problem. But you need to document how you use Gen AI, and make sure you understand everything in your code and double check the correctness.

!!! No late Assignment will be accepted, also No email submission !!!

In this assignment, you will directly use both TCP and UDP sockets in C/Java/Python (or any other language) to implement a simplified publisher-subscriber system, where we will have a master (server program) and several students (running the same client program many times). The Master (server) manages the system's state (users, topics, and their subscriptions), while the students (clients) register with the server, publish/subscribe to topics, using TCP and send direct messages using UDP protocol.

server_host > master 5555 *# run once*

client-host > student username server_host 5555 *# run multiple times with different username*

Master Node (Server):

- The main thread opens a TCP welcome socket to accept connection requests from clients (students).
- When a connection request is accepted, the main thread creates another thread to serve that client. Then the main thread goes back and waits for another client.
- The service thread gets each client's username, client_host, and the UDP port it has registered for direct messaging, and stores these in a user registry (array/linked list)

- The service thread then waits for other requests from the client such as list topics/users, subscribe to, and publish to topics, get UDP port number etc. Accordingly, the service thread responds to these requests.
    - Maintain a topic registry: list of topics and track which users are subscribed to which topics.
    - TCP Communication: All initial communications like registering a user, listing topics, and creating topics will happen over TCP.
    - UDP Information Exchange: Once a client (A) wants to send a direct message to another client (B), A gets the host name of B and its UDP port information from the server again using TCP connection (but clients then can use UDP to send messages to each other) (So, Master/service thread acts like a DNS server in that case)

Student Node(s) (Client):

- Open a UDP port for receiving direct messages from other clients and bind it to an available port (no need for user to enter that number).
- Have at least two threads:
    - One listens to the UDP port for direct messages, and print them on the screen
    - The other establishes a TCP connection with the server and first registers a username, UDP port. Then interacts with the user (student) and sends his/her commands to the master such as list topics/users, subscribe to, and publish to topics. Also, when the user wants to send a direct message to another user, first get his/her registered UDP port from the server, then send the direct messages (DMs) to that user.

Each group is free to use any language and data structure internally. But when developing your application protocol, please use the following message formats (each line ends with '\0' null character" like string in C:

- **REGISTER**: REGISTER <username> <client-host> <UDP port>
- **LIST TOPICS**: LIST TOPICS
- **LIST USERS**: LIST USERS
- **CREATE TOPIC**: CREATE TOPIC <topic_name>
- **SUBSCRIBE**: SUB <topic_name>
- **PUBLISH**: PUB <topic_name> <message>
- **GET DM INFO**: GET-DM-INFO <username> *# client program will send this to the server when the next command is processed*
- **DM USER**: DM <username> <message>

---------- **Example Flow:** ----------

```
server-host > master 5555

client-host > student turgay server-host 5555
```

1. Register a User:

- Client A creates a UDP socket and binds it to an available port (6001) and creates TCP socket and connects it to the master
- Using TCP connection sends: REGISTER turgay client-host 6001 (where 6001 is the UDP port for direct messages, client can use an available UDP port number, no need for user to enter that when running student program).
- Server stores the username turgay, client-host name and port 6001.

2. Create and Subscribe to a Topic:

- Client A sends: CREATE TOPIC tech_news
- Server creates the tech_news topic.
- Client B sends: LIST TOPICS
- Client B sends: SUB tech_news
- Server adds Client A to the list of subscribers for tech_news.

3. Publish a Message:

- Client C subscribes to tech_news and
- Client C sends: PUB tech_news "5G updates are out!".
    - (only clients who are SUBscribed to a topic can PUBlish on that topic)
- Server forwards the message to all subscribers (Client A and B).

4. Direct Messaging:

- Client A can get list of users LIST USERS
- Client A wants to DM Client B.
- Client A sends: DM B "Hello Bob!". (client A gets UDP port number and B's host name from the server use GET-DM-INFO B, this will be sent by the client program, so user does not need to use that command))
- Client A Server looks up Bob's UDP port, returns the info, and Alice uses UDP to send the message directly