Programming Project

Due : 11:59 PM, November 24th, 2024

Program Description:

In this project, you will write complete python programs to support the server and client in a client/server model using Linux stream sockets to develop a chat application. The program will consist of a server that will implement a service (like a chat room) using a simple text protocol that clients can use via client program to message other clients on the service.

Server Implementation:

- The server will set up as a chat server and establish a socket connection using a port number passed in as an argument to the program. For example, the server code may be executed as follows:
  python3 server.py <svr_port>
- The server will print out a usage statement and terminate the program if the user does not enter the command with the port as the command-line argument.
- The server will support up to and including 10 "registered" client connections. Although more than 10 clients can connect to the server, no more than 10 clients may be registered in the database at the same time. In the case of more than 10 clients attempting to register, the server will print out an error message and simply close the connection. Note that clients may come and go as they please, so if, for example, the 10th client "leaves" the service, space is now made available for another client to register.
- The server will set up a network socket (i.e., AF_INET) over TCP (i.e., SOCK_STREAM) using the port number passed in as an argument to the program.
    - The server will bind and listen to the port specified as a command-line argument. When a client connects, a new thread will be spawned to handle the socket using POSIX threads (i.e., pthreads) in Linux.
    - The server will support the following case-sensitive commands which are all sent as plain ASCII text:
        - JOIN username
            - When a client wants to join (i.e., register for) the service, it first connects to the server command with the hostname of the server CSE machine and the port number, and then sends a JOIN request with the username. Usernames will only consist of alphanumeric characters and will not contain spaces or other "special" or control characters. You may assume that the user follows this requirement for alphanumeric characters, so no validation is needed. If the database is "full" (i.e., 10 clients have registered for the service), then the server will print out a status message and send a "Too Many Users" message to the client. Once a client has already registered with a JOIN request, any subsequent JOIN requests from the same registered client will be discarded with a status message sent back to the client.

        - LIST
            - If a registered client wants to know who is currently subscribed to the service, the client will issue a LIST request. Upon receipt of the LIST request, the server will send a list of all registered clients on individual lines and return this

list (newlines and all) to the client. Note that the client must be registered for the service to receive any "services", such as this one, provided by the server. In this program, the registration is equivalent to joining the server using username.

- MESG <username> <some_message_text>
  - If a registered client wants to send an individual message to another registered client, the client will issue the MESG request with the username of a registered client followed by whatever message he/she wants to send to the other registered client. The server will then act as a relay and forward this message to the registered user. Note that the client must be registered for the service to receive any "services", such as this one, provided by the server. If the client who is not registered for this service sends a MESG request, the server will print out a status message and send an "Unregistered User" message to the client with the JOIN request instructions. If a registered client sends a MESG request to an unregistered client, the server will print out a status message and send an "Unknown Recipient" message to the client.

- BCST <some_message_text>
  - If a registered client wants to broadcast a message to all other registered clients, the client will issue the BCST request followed by whatever message he/she wants to send to the other registered clients. The server will then act as a relay and forward this message to the registered users (but not the sender). Note that the client must be registered for the service to receive any "services", such as this one, provided by the server.

- QUIT
  - When a connected client wants to leave the service, the client will issue a QUIT request, at which time the server will disconnect the client from the service. The database entry for registered clients should be removed after the client has been disconnected. Note that an unregistered client will still be disconnected from the service (i.e., their connection closed) with a status message at the server, though no data needs to be removed form the database since there is none for that client.

- Unrecognized Messages
  - If a registered client sends an unrecognizable request (i.e., one not supported by this protocol), the server will print out a status message and send an "Unknown Message" message to the client.

- The server will provide important status updates of messages sent and received, such as connection events and received requests, identifying the client using their socket file descriptor.
- The server will support error checking across all relevant system calls and other potential issues.

Client Implementation

You will client program client.py to connect to the server, specifying the hostname and port number of the server to connect to.

- When the client program starts up, the client can issue any request (even unrecognizable ones) to the server, but is expected to issue the JOIN request immediately with the client's username using alphanumeric characters so as to be able to use the services provided by the server.
- When desired, the client will send a LIST request to the server to see who is currently subscribed to the service, but only the registered client will receive the listing (i.e., others will receive an error message with JOIN request instructions).
- When desired, the client will send a MESG or BCST request to the server who will then relay that message to an individual registered client or all registered clients, respectively.
- The client may voluntarily leave the service at any time by issuing the QUIT request.

Your program (i.e., a server program) should run on the INET domain using SOCK_STREAM (i.e., TCP) sockets so that the server and client execute on different ECS machines at the same time. The server will accept the port number to communicate on as an argument to the server program. Your code will handle errors appropriately, such as printing an error message, disconnecting the client, or termination of the program.

Group Formation Instruction:

You will form a group of up to 3 people for this project and sign up on Canvas for the same. You must finalize your group membership by 24th October. If you wish to join another section (someone in section 3 want to join section 4 group) to form a group, you must inform me by the above date. If you face any problem with forming a group, you must contact me. Anyone, who could not find a group by the above date will work individually and submit an individual project assignment and report.

Please fill out the project sign up sheet provided here CSCCPE138-01 GROUP INFORMATION. Additionally, you must add yourself to the intended group on Canvas.

SAMPLE OUTPUT:

Note: The ECS server may not allow you to connect from ecs-coding1.csus.edu server to ecs-coding2.csus.edu server due to security policies. You can test your chat server in the same machine by opening multiple terminals and emulating the server and client. The below example is shown for client/server chat application in ecs-coding1.csus.edu machine.

==> SERVER on ECS-CODING1

$ python3 server.py

usage: python3 server.py <svr_port>

$ python3 server.py 8001

==> Client on ECS-CODING1

$python3 client.py ecs-coding1.csus.edu 8001

Enter JOIN followed by your username: Mike

Server output after 5 users joined the chatroom

The Chat Server Started
Connected with ('127.0.0.1', 50692)
Alice Joined the Chatroom
Connected with ('127.0.0.1', 51131)
Bob Joined the Chatroom
Connected with ('127.0.0.1', 51395)
Alex Joined the Chatroom
Connected with ('127.0.0.1', 51785)
Mike Joined the Chatroom
Connected with ('127.0.0.1', 52183)
Trudy Joined the Chatroom

Test Case 1 : Alice wants to send a broadcast message using "BCST Hello" to all users. After sending the broadcast, the window looks like below.

Alice's Window

Enter JOIN followed by your username: JOIN Alice
Alice joined!Connected to server!
Bob joined!
Alex joined!
Mike joined!
Trudy joined!
BCST Hello
Alice is sending a broadcast
Alice: Hello

Bob's Window

Enter JOIN followed by your username: JOIN Bob
Bob joined!Connected to server!
Alex joined!
Mike joined!
Trudy joined!
Alice: Hello

Alex's Window

Enter JOIN followed by your username: JOIN Alex
Alex joined!Connected to server!
Mike joined!
Trudy joined!
Alice: Hello

Test Case 2: Mike wants to send a private message to Bob using "MESG Bob How are you"

Mike's Window

Mike joined!Connected to server!
Trudy joined!
Alice: BCST Hello
MESG Bob How are you?

Bob's Window

Mike joined!
Trudy joined!
Alice: BCST Hello

Mike: How are you?

Alice's Window

BCST Hello
Alice is sending a broadcast
Alice: Hello

Test Case 3: Now, Trudy wants to see who all have joined the chat server with LIST command.

Trudy's window

Trudy joined!Connected to server!
Alice: BCST Hello
LIST
Alice,Bob,Alex,Mike,Trudy

Test Case 4: Alex wants to quit the chat room with QUIT command.

Alex's Window

Enter JOIN followed by your username: JOIN Alex
Alex joined!Connected to server!
Mike joined!
Trudy joined!
Alice: BCST Hello
QUIT
Alex is quitting the chat server

Alice's Window

Enter JOIN followed by your username: JOIN Alice
Alice joined!Connected to server!
Bob joined!
Alex joined!
Mike joined!
Trudy joined!
BCST Hello
Alice is sending a broadcast
Alice: BCST Hello
Alex left

Program and Submission Guidelines:

This assignment must be submitted via Canvas with the following elements:

- Your code should be well documented in terms of comments. For example, good comments in general consist of a header (with your name, course section, date, and brief description), comments for each variable, and commented blocks of code.
- Your python program file should be named "server.py" and "client.py", without the quotes, for the server code.
- Your program will be graded based largely on whether it works correctly on the ECS machines (e.g., ecs-coding1, ecs-coding2, ecs-coding3), so you should make sure that your program compiles and runs on a ECS machine.
- Please pay attention to the SAMPLE OUTPUT for how this program is expected to work. If you have any questions about this, please contact your instructor.
- If you have used external resources (paper/digital) to complete your project, you must mention it in the report.
- You will prepare a report describing the methodology/functionality of the code, attaching the code output/screenshots and test cases for validating your results.
- Your report must contain your name, section number and email ID.
- You will electronically submit your programming codes and report files by the due date on Canvas.

GRADING RUBRIC:

- Working codes with all the required functionality: 70 Points.
    - JOIN Functionality: 15 Points
    - LIST Functionality: 15 Points
    - MESG Functionality: 15 Points
    - BCST Functionality: 15 Points
    - QUIT Functionality: 10 Points
- Report submission with description, outputs: 30 Points.
- Comments for the code: 10 Points.
- Maximum points will not exceed 100.