Computer Engineering 4DN4
Laboratory 4
Online Group Chatting Application

The objective of this lab is to develop a client/server network application that implements multi-group online chatting. The code will be written in Python 3 using the Berkeley/POSIX socket API discussed in class. The server code operates as a directory manager for online "chat rooms" that can be queried by the clients (i.e., the chat room users). The clients can dynamically create, delete and join chat rooms. After joining a chat room, the client software exchanges messages with other clients using IP multicast communications. The working system has to be demonstrated by each team in one of the laboratory sign-up sessions.

1 Description

The software to be developed consists of separate server and client applications written using Python 3. The required functionality is discussed as follows. Note that in the Marking Scheme section below, there is a description of output that the software must generate when doing the demonstration.

Server: The server code is run on a server host, referred to as the Chat Room Directory Server (CRDS). The main function of the CRDS is to maintain a chat room directory (CRD) for the clients. The details are as follows.

1. The Chat Room Directory Server (CRDS) is started from the command line in a shell window.
2. After startup, the server continually listens for client connections on its Chat Room Directory port (CRDP). When a CRDP connection occurs, the CRDS accepts commands from the client that involve listing, creating and deleting chat rooms. The server must be able to handle multiple client connections at the same time.

   The client-to-CRDS commands are as follows:

   - `getdir` : The server returns a copy of the current chat room directory (CRD). Each entry in the directory includes the chat room name, multicast IP adddress and port.
   - `makeroom <chat room name> <address> <port>` : The server creates a chat room directory (CRD) entry that includes the above information. Note that `<address>` is an IP multicast address used by the clients for chat messages (Use the administratively scoped IP multicast range: `239.0.0.0` to `239.255.255.255`).
     The CRDS should check that the new address/port combination is not currently in the directory.

- `deleteroom <chat room name>` : The CRDS removes the chat room entry from the CRD.

Client: The client code is run on a user host to access the Chat Room Directory Server (CRDS) and to chat with other clients using IP multicast communications. The details are as follows.

The client is started from the command line in a shell window. When the client starts, it presents a prompt (the "main command prompt") to the user, and awaits commands. The commands are as follows:

- `connect` : The client connects to the CRDS on the CRDP. Once connected, the client can issue the CRDS commands discussed above, i.e., `getdir`, `makeroom` and `deleteroom`. The CRDS responds as discussed previously. You should use a different command line prompt to show that you are connected to the CRDS.
- `bye` : This closes the client-to-CRDS connection, returning the user to the main command prompt.
- `name <chat name>` : This command sets the name that is used by the client when chatting, e.g., `name Mel`. All chat messages will be automatically prefixed with this name, e.g., "Mel: Good morning!".
- `chat <chat room name>` : The client enters "chat mode" using the multicast address/port for the associated chat room. You should change the command line prompt so that it is clear when chat mode has been entered.

  Text typed at the command prompt is sent over the chat room. Text received from the chat room is output on the command line.

  While in chat mode, a defined control sequence (e.g., `<ctrl>]`) is used to exit chat mode and return to the main prompt.

2 Requirements

Teams: You can work in teams of up to 4 students.

Demonstration: The working system must be demonstrated by the team, with all members in attendance.

Each team needs to immediately reserve a 20 minute time slot in one of the designated lab sessions. Time slots will be assigned on a first-come-first-served basis using the Doodle scheduler. Make sure you are certain of the team's schedule, since once a time slot is booked, it cannot be changed. Further details of the lab demo sign-up are on the course web site page for Laboratory 4.

Marking Scheme: The assigned mark consists of two parts, i.e., 90% for demonstrating the system and a 10% discretionary component for questions answered, based on your Python code, during the demonstration. Make sure that you are on time for the demonstration.

The demonstration consists of the following, where each step below is weighted equally in the demonstration component of the mark. Note that you should not have to modify your code during the demonstration session.

1. Start the server in a shell window. The server should print output indicating that it has started and is listening on the CRDP for incoming connections, e.g., "Chat Room Directory Server listening on port `<port number>`..."
2. Start the client in a shell window on the client laptop. The client will prompt the user for commands. The user issues a `connect` command to connect to the CRDS. The CRDS should output a status message indicating that a client connection has occurred.
3. The client issues a `makeroom` command to create a new chat room. The client then issues a `getdir` command to show that the new chat room is included in the CRDS directory.
4. The user issues a `bye` command. The server should output that the connection has been closed.
5. Two clients set their names with the `name` command.
6. Two clients then issue `chat` commands and demonstrate that chatting is working.
7. An example of a client deleting a chat room should be given (i.e., `deleteroom` and `getdir` commands).
8. The group should show that the server can interact with multiple concurrent client connections.

Writeup: You must upload the following two documents to your Avenue To Learn Lab 4 dropbox:

1. A PDF report that briefly describes how you implemented the client and server applications. Make sure that the names and student ID numbers are on the front page of the report.
2. A single Python source code (.py) file that includes both the client and server classes that you created.