MP1: A Chat Room Service

150 points
Due: February 9th, 2023, at 11:59pm

1 Overview

The objective of this assignment is to refresh your memory with the network programming experience you gained in CSCE 313. To accomplish this, you will build a **Chat Rooms Service**.
The system can have several Chat rooms, and clients join chat rooms by issuing `JOIN` commands. Clients can also create and delete chat rooms remotely by issuing `CREATE` and `DELETE` commands. All communication between chat rooms and chat clients will be through network sockets, a topic that you learned in CSCE 313.

1.1 Chat Room Server

The program/process implementing the **Chat Room Server** (call it `crsd`) would manage chat room creation/deletion/listing requests and communication between **Chat Room Clients**. The server program takes the port number of where it will be listening to client requests as the first and only command line argument.
The **Chat Room Server** would do the following steps in a loop, in any order:

- Listen on a port number, passed as argument via command line, to accept `CREATE`, `DELETE`, or `JOIN` requests from Chat clients.
- For a `CREATE` request, check whether the given chatroom exists already. If not, create a new master socket (careful about picking the port number!) Create an entry for the new chat room in the local database, and store the name and the port number of the new chat room. Inform the client about the result of the command.
- For a `JOIN` request, check whether the chat room exists. If it does, return the port number of the master socket of that chat room and the current number members in the chat room. The client will then connect to the chat room through that port.

- For a `DELETE` request, check whether the given chatroom exists. If it does, send this warning message `"Warning:the chatting room is going to be closed..."` to all connected clients before terminating their connections, closing the master socket, and deleting the entry. Inform the client about the result.
- For a `LIST` request, check if chatrooms exist. If it does, send the client a string containing comma separated list of chatroom names (e.g., `"r1, r2, r3, r4"`). Else, send the string `"empty"`.
- Incoming chat messages are handled on slave sockets that are derived from the chat-room specific master socket. Whenever a chat message comes in, forward that message to all clients that are part of the chat room.
- Clients leave the chat room by (unceremoniously) terminating the connection to the server. It is up to the server to handle this and manage the chat room membership accordingly.

1.2 Chat Room Client

The program/process implementing the **Chat Room Client** (`crc.cpp`) would issue requests to and exchange messages with the chat room server. The client program takes the hostname and the port number of the chat room server as first and second command line argument, respectively. The client program provides a simple command line interface of your design, which reads both commands to create, delete, or join chat rooms, and messages to the currently joined chat room.
The chat room client has already been implemented and would create, delete, and join chat rooms in the following way:

- To create a chat room, connect to the well-known port of the chat server, and send a `CREATE <name>` message. Display the reply and close the connection.
- To delete a chat room, connect to the well-known port of the chat server, and send a `DELETE <name>` message. Display the reply and close the connection.
- To join a chat room, connect to the well-known port of the chat server, and send a `JOIN <name>` message. Read the information with the port number of the chat room and the current number of members. Display the information.

- If the `JOIN` operation was successful, connect to the port returned by the server, and begin exchanging messages.
- Leave the chat room by terminating the connection.

2 What to Hand In

The running system will consist of the chat room server (`crsd`) and the chat room client (`crc`). As platform, you should be using the programming environment mentioned below and develop the program in C++.

2.1 Design

Before you start hacking away, plot down a design document. The result should be a system level design document, which you hand in along with the source code. Do not get carried away with it, but make sure it convinces the reader that you know how to attack the problem. List and describe the components of the system: Chat Client Program, Chat Room Server Program, and their interaction. In particular, describe how you implement the server program (iterative, multi-threaded using thread, multi-threaded using processes, multi-threaded using `select()`, others.)

2.2 Source code

Hand in the source code, comprising of CMakeLists.txt, `crsd.cpp` and `crc.cpp`, and any auxiliary files (for example, socket handling). The code should be easy to read (read: well-commented!). The instructor reserves the right to deduct points for code that he/she considers undecipherable.

3 Programming Environment

For all Machine Problems in this class you will use Ubuntu installed on VirtualBox/UTM. Using an environment like this will significantly reduce the likelihood of environment-related errors for your submissions(when compared to the environment we will be using to test your code). All the libraries required for the problems are already pre-installed in the image provided.

To setup the VM, please follow the steps below:

Note: Before proceeding with the below steps. Make sure you have atleast 30-40 GB of free space on your machine.

3.1 Mac M1/M2 users

1. Download and install UTM.
2. Download the image from here. Since the file is huge, it might take a few minutes to download.
3. Follow the steps in this video to install the VM. Set the **Storage size** to 35 GB.

3.2 Others

1. Install VirtualBox on your machine.
2. Download the image from here. Since the file is huge, it might take a few minutes to download.
3. Open VirtualBox, go to `File > Import Appliance`. Under `File`, choose the OVA file that you just downloaded. Keep the rest of the details as is and import the appliance.
4. Once it is installed, you can start the VM and login to the machine. The user password for csce438 user is 123 .
5. You can now use this VM for all your machine problems.

Once the installation is done, you can start the VM and login to the machine. You can now use this VM for all your machine problems. Note that the user password for csce438 user is 123 .

4 Programming Guide

This is a quick guide how to start your MP1 with the given client code. The objective of the skeleton is to let students focus on socket programming. The client code has been implemented in the starter code and you need to work on the server side of the program in `crsd.cpp`.

- The starter code consists of:

  crc.cpp

  The client code has already been implemented for you. You don't have to modify anything here. The code is well commented for your understanding. You will only be working on the server side of the Chat Room Service.

  Interface.h

  It contains functions that main function uses and Reply (user defined structure) that is used in process_command function. You don't have to modify this file.

  crsd.cpp

  This is where most of your code goes in. The server should be able to process and return responses to all the commands being sent by the client(s). Make use of the failures defined in Interface.h to report any issues.

- The three main functions in client are:

  ```
  int connect_to(const char *host, const int port);
  struct Reply process_command(const int sockfd, char* command);
  void process_chatmode(const char* host, const int port);
  ```

  There are descriptions about these functions in the skeleton. Please read the description carefully and implement the server side accordingly.

- All output/logging from the server must be done through the glog logging library. The library is installed for you on the provided appliance.

  Please make sure you understand the different logging levels that glog provides (e.g., DEBUG, INFO, ERROR, etc) and make use of them appropriately. A good reference is: https://github.com/google/glog. For this assignment, a simple logging like the following:

  ```
  LOG(<Severity Level>) << "Found " << num_cookies << " cookies";
  ```

  will suffice.

- All output/logging from the client, other than the I/O used for the User Interface, must also be done through the glog library.
- Test your program with given test cases below. There are five test cases that cover most scenarios. You can see test cases and its results in the provided excel file. Before test each test cases, restart your server/client program to have the same output with the given output in the excel file. Evaluation will be done with different test cases.

5 Testing Scenarios

Find the attached excel sheet titled `normal_test_cases.xlsx` detailing five different test cases that are expected to work.

If you have any questions, please contact let us know.