MP1: A Chat Room Service

150 points
Due: January 31st, 2022, 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 use the network sockets that you learned in CSCE 313.

1.1 Chat Room Server

The program implementing the **Chat Room Server** (call it `crsd`) would do the following steps in a loop, in any order:

- Listen on a well-known port to accept `CREATE`, `DELETE`, or `JOIN` requests from Chat clients.
- For a `CREATE` request, check whether the given chat room exists already. If not, create a new master socket (make sure the port number is not usedr) 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 chat room exists. If it does, send a warning message (e.g., `chat room being deleted, shutting down connection...`) to all connected clients before terminating their connections, closing the master socket, and deleting the entry. Inform the client about the result.
- 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.

The chat room server is invoked as below:

```
$crsd <port_number> &
```

1.2 Chat Room Client

The program implementing the **Chat Room Client** (call it `crc`) issues requests to and exchanges messages with the chat room server.

The client program is invoked as below:

```
$crc <host_name_or_ip_address> <port_number>
```

The client program provides a simple command line interface of your design, which reads both commands (create, delete, or join chat rooms), and messages to the currently joined chat room.

The chat room client creates, deletes, and joins 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.

Note: There is a little complication, as the client program has to read input both from the command line and from the connection to the chat room. You have to find a solution to handle this. (One way is to use separate threads to handle the two input sources, or you can simply use `select()` to handle them in a single-thread solution.)

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 described below and develop the program in C/C++.

2.1 Design

Before you start hacking away, write down a design document. The result should be a system level design document, which you hand in (on canvas) along with the source code (in github) - more about submissions on canvas/github below. 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

Your source code, comprising of a Makefile, a file `crsd.c`, a file `crc.c` and any auxiliary files (for example, socket handling) MUST be checked in the public Github repository here (please note, NOT TAMU Github), using your Github account and with the project name **MP1**. Please provide the TA your Github account name so he can checkout your submission.

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.

The design document mentioned above must be submitted through canvas.

3 Programming Environment

For all Machine Problems in this class you will use a free AWS Cloud 9 environment. 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).

To setup an AWS Cloud 9 environment, please follow the steps below:

1. Go to aws.amazon.com/cloud9/
2. Click on **Sign In to Console** button on the top right corner
3. For those without an account please go ahead and click the **Create a new AWS** account button towards the bottom, while others can proceed to login with their credentials
    - It is advisable to use university email addresses (tamu.edu) domain to create an account here.

- All accounts created will have free services for upto one year since the time of initiation of the service
- For more information regarding AWS Free Tier

4. On the search bar up top, proceed to enter **Cloud9** and click the relevant option from the auto populated dropdown.
5. You can go on and click on the orange button titled **Create Environment** on the right hand side of the screen.
6. Please feel free to name the environment as per your liking and the click on the **Next Step** button.
7. In the Configure Settings (the screen that appears after hitting the Next Step button in the prior step) you can leave the settings as is. But you can confirm them to ensure that your service usage or request falls under free tier constraints :
   - Environment type : Create a new EC2 instance for environment (direct access)
   - Instance type : t2.micro (1 GiB RAM + 1 vCPU)
   - Platform : Amazon Linux 2
   - Cost-saving setting : After 30 minutes (default)
8. Once you have confirmed the settings in the above mentioned point, proceed to click **Next Step** button, this leads to the review page that shows all the chosen options. After you review them, you can hit the **Create Environment** button
9. Your AWS Cloud 9 environment is created and ready to be used
10. gcc is already setup and ready to be used here

4 Programming Guide

This is a quick guide on how to start your MP1 with a given skeleton code of a client program. The objective of the skeleton is to let students focus on socket programming, and also to evaluate your MP fairly by having the same skeleton. The skeleton has been implemented with almost everything except the socket programming code.

Please follow the steps below:

- Study the skeleton code provided:

  crc.c There is a main function that already has been implemented. You don't have to modify the main, but you need to understand what main function does so that you fill your own code in the skeleton. Also, there are three functions that have not been implemented yet. Your job is implementing these three functions.

  Interface.h It contains functions that main function uses and Reply (user defined structure) that you will use in a **process_command()** function. You don't have to modify this file.

- Implement three functions:

```
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 of these functions in the skeleton. Please read the description carefully and follow the suggestions.

- Do not print anything on the screen. The skeleton does print everything on screen for you to interact with a user. What you need to do is use the **get_message()** function to get a message from a user during chatting and **display_message()** function to print messages from other members in a chat room. You can print some debugging information during the development phase, but you must erase them when you submit your MP.

- Test your program with the given test cases. There are five test cases that cover most scenarios. You can see test cases and their 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.