CMSC 417 Computer Networks **Fall 2026**

Programming Assignment 3

Assigned: October 16 *Due: October 29, 11:59:59 PM.* *Weight: 1.5x*

1 Introduction

In this project, we will be providing a chat client and will host a chat server. Clients allow users to communicate with one another by connecting to the server and interacting with it in accordance to an application protocol. Through this protocol, the server allows clients to engage in group chats in chat rooms and send private messages to one another.

Your task will be to reverse engineer the chat server and its protocol and use this information to write a compatible replacement. Alternatively, you may elect to take a 20 point penalty and instead implement a compatible client. You are only required to implement the server **or** the client, **not both**.

2 Client (Provided)

The client is available in the 'assignment3' repository on gitlab. The provided client takes single optional command line argument:

1. **-u** = Run with limited ncurses support for a slightly better chat experience.

While running, the client takes commands directly from the user. All commands are preceded by a backslash. Not every command is available in every context. The client supports the following commands:

1. **\connect <IP Address>:<Port>** = Instruct the client to connect to a new chat server, specified by the IP address and port.
2. **\disconnect** = If connected to a server, disconnect from that server.
3. **\join <Room> <Password>** = Join the specified chatroom, creating it if it does not already exist. The Password is optional, with the default being the empty string. Users may only join rooms for which they know the password. Both Room and Password must be less than 256 characters in length.
4. **\leave** = If in a room, this exits the room. Otherwise, it leaves the server.
5. **\list users** = List all users. If in a room, it lists all users in that room. Otherwise, it lists all users connected to the server.
6. **\list rooms** = List all rooms that currently exist on the server.
7. **\msg <User> <Message>** = Send a private message to the specified user. User must be less than 256 characters in length, and the Message must be less than 65536 characters in length.

8. **\nick <Name>** = Set your nickname to the specified name. Name must be less than 256 characters in length.

9. **\quit** = Exits out of the client.

All other input is interpreted as a message being sent to the room the user is currently in.

3 Server (Hosted)

We will be running a server; see Piazza for details.

4 Replica Implementations

4.1 Client

Your replica client must not take any required arguments, and does not need to support the -u option. The output of your replica client must *exactly* match that of the provided client for all sequences of commands and messages. You can test your replica client implementation by comparing the output to that of the provided client using `diff`.

4.2 Server

Your replica server must support the following command line arguments:

1. **-p <Number>** = The port that the server will listen on. Represented as a base-10 integer. Must be specified.

5 Grading

Your project grade will depend on how much client functionality is maintained when connecting to your server. We will test your server by running the client in standard input and output mode (i.e., without the **-u** option) and directly comparing the output with identical input against your server and the reference implementation.

If you elect to implement the client instead of the server (and take the 20 point penalty), we will `diff` the output of your client when fed identical input against the reference client in standard input and output mode.

We strongly encourage you to write tests against the reference implementation to compare against your own implementation.

6 Additional Requirements

1. Your code must be submitted as a series of commits that are pushed to the origin/main branch of your Git repository. We consider your latest commit prior to the due date/time to represent your submission.

2. You must provide a Makefile that is included along with the code that you commit. We will run 'make' inside the repository root, which must produce either a 'rserver' or 'rclient' executable also located in the repository root.

3. You must submit code that compiles in the provided Docker setup, otherwise your assignment will not be graded.

4. Your code must be -Wall clean on gcc/g++ in the provided Docker setup, otherwise your assignment will not be graded. Do not ask the TA for help on (or post to the forum) code that is not -Wall clean, unless getting rid of the warning is the actual problem.

5. You are not allowed to work in teams or to copy code from any source.