Data Communication and Networks	Project 1: Simple Chat Application
CSCI-351 Fall 2019	September 5, 2019

This project is due at 11:59:59pm on September 19, 2019 and is worth 15% of your project scores. You must complete it with a partner. You may only complete it alone or in a group of three if you have the instructor's explicit permission to do so for this project.

Note that there is a milestone deadline for this project, at 11:59:59pm on September 19, 2019. More details are in the Milestone section below.

1 Description

In this project, you will implement a (1) server and (2) client-side program of a chat application using C-socket programming. Each client can send messages to the server, which broadcasts all the messages to the rest of the clients.

2 Requirements

You will use **C-Socket libraries** to write the chat server and client application. You can use any transport layer protocol to implement the project. Your chat program must meet the following requirements:

- Allow clients to have smooth communication without any interuption.
- Support multiple clients (more than 10 clients) at a time for their communication.

Note that more sophisticated and properly tuned structure of program will be given higher credit.

Regardless, correctness (working code) matters most; performance is a secondary concern. We will test your code by introducing a variety of different errors and failures; you should handle these errors gracefully, recover, and *never* crash.

3 Your program

For this project, you will submit two programs: a client and a server program that implement a chat application. You must use C-Socket libraries that we discussed in the class. You may *not* use any other third-party libraries in your project; you must implement all of the chatting logic yourself. Your executables should be named `351ChatClient` and `351ChatServer`.

3.1 Requirements

You should develop your client program on the glados Linux machines, as these have the necessary compiler and library support. You are welcome to use your own Linux/OS X/Windows machines, but you are responsible for getting your code working, and your code *must* work when graded on

the glados Linux machines. If you do not have a glados account, you should get one ASAP in order to complete the project.

The command line syntax for your server and client program is given below. The server program takes command line arguments representing (a) port number of the program running, and (b) the maximum number of clients allowed to connect. The syntax for launching your server program is therefore:

```
./351ChatServer <port> <num_clients>
```

`port` (Required) The port number of the chat server. You should handle the errors gracefully (e.g., the port requires privileges, port is already being used, etc.).

`num_clients` (Required) The maximum number of clients that can join the chat. The number must be larger than 10; the other clients that would join after this limit have to wait until other clients in the chat leave.

The server program must meet the following requirements:

- The server must support multiple clients (more than 10 clients) at a time for their communication.
- If the number of users who are currently in the chat server reaches the limit `num_clients`, it has to let the other incoming users know how many waiting users are ahead of. It also has to be updated whenever the number of users in the chat changes (See an working example). Finally, the server allows them to join once any vacancies are available.
- When new users join the chat, the server broadcasts who and when they have joined.

```
./351ChatClient <ip> <port> <name> <password>
```

`ip` (Required) The IP address of the chat server. You may use a local IP address.

`port` (Required) The port number of the chat server.

`name` (Required) the username which will be used in the chat application.

`password` (Required) the password bound to the username.

The client program must meet the following requirements:

- The client can re-join the chat room using the same username and its credential.
- The client can exit the chat using "#EXIT" command.

3.2 Working examples

The below example shows the overall behavior of how the chat program works. However you can design your own interface to meet all the requirements.

```
$./351ChatServer 8080 10
$./351ChatClient 127.0.0.1 8080 Tracer pa$$word
[ChatBot (08:14:21)] You're in the waiting queue: 3 users ahead of you
[ChatBot (08:15:51)] You're in the waiting queue: 2 users ahead of you
[ChatBot (08:30:21)] You're in the waiting queue: 1 user ahead of you
[ChatBot (08:31:22)] Tracer has joined the chat.
> Cheers love! The cavalry's here!
[Tracer  (08:31:22)] Cheers love! The cavalry's here!
[JB      (08:31:24)] Hi!
[Jaehyuk (08:31:27)] Bye guys.
[ChatBot (08:31:27)] Jaehyuk has left the chat.
[BBirec  (08:31:28)] Please just leave.
> #EXIT
$./351ChatClient localhost 8080 Tracer pa$$word
[Jaehyuk (08:31:27)] Bye guys.
[ChatBot (08:31:27)] Jaehyuk has left the chat.
[BBirec  (08:31:28)] just leave, I don't like you.
[ChatBot (08:32:00)] Tracer has joined the chat.
> :(
[Tracer  (08:32:01)] :(
```

3.3 Extra Credits (15%)

- *Some of you would notice a limitation of this framework: if someone sends a message during your input, your input will be garbled. In such case, you may consider using other interfaces for user inputs such as* `ncurses`*, which is the only permitted third-party library for this project.* **There will be a 10% bonus point if this corner case is gracefully handled.**
- When the same client re-join the chat room with the same credential, you can show them up to the last 3 messages before they left.**There will be a 5% bonus point if this is gracefully handled.**

4 Grading

The grading in this project will consist of

75% Program correctness

10% Error handling

15% Style and documentation

5 Submitting your project

5.1 Final submission

For the final submission, you should submit your (thoroughly documented) code along with a plain-text (no Word or PDF) README file. In this file, you should describe your high-level approach, the challenges you faced, a list of properties/features of your design that you think is good, and an overview of how you tested your code. You should submit your project to Project1 folder in the Mycourses Dropbox. Specifically, place all of your code and README files into one folder (Project1) and zip it (Project1.zip) and upload it to the Dropbox.

You must submit your project by 11:59:59pm on September 19, 2019.

6 Advice

A few pointers that you may find useful while working on this project:

- To support multiple clients at the same time, you may want to look `select()` and `fork()` functions.
- Check the Mycourses discussion board for question and clarifications. You should post project-specific questions there first, before emailing the instructor.