Lab 1: Client/Server Model

Deadline: September 15

Goal

The goal of this lab is to learn the basic programming of multi-thread Java applications to understand how the network plays an essential role in software communication. In this lab, you will build a Java multi-threaded chat server that serves multiple clients. Also, you will develop a java client application to simulate various concurrent users attempting to chat with each other, and test the network performance with a large number of simultaneous users.

Note: you must program this lab using Java. I prefer to do it in a Linux environment, where you can write shell scripts to analyze the network performance.

Tips and Guidelines

- At a fundamental level, the client and server are multi-thread Java programs that communicate over TCP network sockets.
- A non-threaded server basically can handle one connection at a time. Therefore, an excellent way to build servers capable of handling multiple clients, you should create various threads where each thread can handle one connection. A thread will maintain a single client it was intended for, and terminate when it has completed serving the client. Meantime, the primary server process will resume to wait for and accept new connections. Your server must do the same, learn about how to create, execute, and terminate threads in Java.
- Your client will simulate many users they want to chat with each other, where each client must have one chat window. Like any basic chat windows, you should provide the user with a form. This form should include a single text box to write their messages and a scrollable text box to view the received messages. You should learn about JAVA Swing.
- The most crucial part of the arrangement for this lab is to use two separate computers to run the client and server programs. We will talk about the options in class.

- Ideally, we would like the clients to chat with each other using identical user names, and the server should refuse the connection if the name exists in the user list. The server should maintain a user list, and the client should ask for a username from the user as input. However, the client should not maintain usernames list or validate the username.

Building your server

The server should accept connections from a designated port, and this port should be assigned by default or as a user arguments before running the server. Here is an example below:

```
$> java server <port number>
```

The server should maintain a high availability and prevent from crashing. Additionally, the server needs to provide a meaningful reply to clients that fails to connect.

Building your client

The client should request a connection from the server using the server IP address and port number. If the connection was successful, the chat window should appear asking the user for the another username to connect with, or the user can choose later. Here is an example below:

```
$> java client <IP address> <port number> <your username>
```

If you are running both applications on the same machine, you can change the IP address to "hostname". The client should maintain a high availability and prevent from crashing. Additionally, the client needs to provide the user with meaningful reply if it fails to connect to the server. The client should inform the user if the other chat partner is available to chat.

Graduate Students

The requirements in this section intended for graduate students but undergraduates are welcome to do it for extra credit.

- The server should save chat messages if the target clients are not connected and deliver them when they are connected.
- Record the time needed for each message delivered from client A to client B, and draw a chart showing the average delivery time of messages between 10, 100, 1000, 10000 clients. For this requirement, you should automate the process.

Submission

You have to solve this assignment in groups of two to four students. You must use a git repository (LoboGit) to maintain your code and final submission. After Me and the TA, cc'd, receive an email from each group with members username and UNM ID's. We will send you a git project link by email. When you submit your work, the repository should contain the following:

- A wiki page, which includes a brief explanation on how your group solved the practices above. Be precise and brief in your writing.
- Your code ChatServer.java, ChatClient.java, and other Java files. Make sure your code is well documented and readable, I will deduct points if the code misses documentation.
- **Extra credit:** an optional makefile to build your code.