Team 15

Report by: Jeremy Polansky

Problem Statement

This program’s purpose is to develop a system in which a server is used to allow clients to relate by sending messages and associating themselves with certain other clients. It will include writing code from two sides of a communication between the server and client sides where both sides will be able to understand the other.

Approach to Solution

This program was written using C++, and then later would be executed in a Linux environment. There is code written for the server side and for the client side, of which the client code will be used for any client that would talk with the server. From the client side, it will talk with the server and send over authenticity information (user name and password) and send commands to do certain things, including sending messages to other users. On the client side, it will, from startup, take in credentials from the client to “login” and authenticate a certain user, then take in commands to run on behalf of that certain user, such as sending messages to certain users on behalf of the user sending the message. It will save information about how the users are related so that it will be able to relay the information later as needed. Because the server and client sides are not able to communicate directly but only can send binary messages between them, a specific order of communication is used so that each set of code (for server side or client side) is pre-programmed to follow suit with what the other is asking/giving across the connection. In other words, from the main menu on the client side, the client must send information to the server depending on what the user selects because the client is able to send information to the server or else the server would not know what that information applies to (if it’s a username for login credentials or to request to dictate to whom to send a message). The tool used was g++ version 5.2.1 for Linux. Our test environment involved three Ubuntu 15.10 vm on a single hypervisor( kvm to be exact).

Solution Description

To build the entire solution, simply execute

“make”

To remove the generated .o object files after compilation, simply execute

“make clean”

To remove all files generated after compilation, and simply execute

“make cleanall”

To run the server program execute

“./Program6-Server.out” <arg-Port num>

To run the client program execute

“./Program6-Client.out” <arg-ip Address, arg-Port Num>

This program meets most course learning outcomes. Most of their uses are trivial (it’s either there or not, such as a recursive algorithm) but here are a couple specific locations of them. For the fourth learning outcome, utilizing file I/O, for example, is used for prompting and receiving user input on the client side for things like login credentials sent to the server. The eighth, dynamic data structures, was utilized for the information on the server.