Ramzey Ghanaim

CMPE 156 Lab 1

High Level Collaborators: Andrea David

January 26, 2018

Introduction

The purpose of this lab was to create a basic client-server remote shell program. The client will take shell commands and pass them to a server. The server will execute the command and pass the resulting output back to the client. The client will then display the data.

Client-Server model

Following the standard Linux/Unix client server model and standard C libraries, I first made a client server model. Both the socket and client created a socket over a TCP connection. The server will then bind its' IP address with the specified port number. Both the IP address and port number are arguments of the program that the user enters. Next, the server listens for incoming connections. The client will the connect to the server and the server will accept the connection request. Once the set up is complete the client can start sending commands to the server with the send() function. The server will receive the commands with the recv() function and send the result of the commands as a string to the client with the write() function. The client will receive the data with the recv() function. When the client wants to quit, I purposefully designed my program so that the exit function only affects the client closing the connection such that the server can stay active for another client to connect. When exiting my server, I catch the CTRL+C from the user and verify they want to quit. I then close the file descriptor and free any mallocs I made.

Protocol

First off buffer size of 51 for both the user input and result buffers. However, I only allow the use of 50 for data that is being sent. This is to have space for my “done sending” or “there’s more to send” chars. I assumed a user command would not go above 50 chars, and thus could be sent with a single send() command. However, I still sent an end of command signal which was the “@” character. I have checks to ensure the character is the end character and not just an “@” that is part of a command. This involved making the buffer sizes one size bigger. This last space is reserved for the @ char for when the buffer is filled completely.

I implemented a similar protocol when sending the result from the server to client. Once the output is ready to send from the server to the client, I check to see if there is another string that can be sent. If there is no more strings to send I send the done char which is “@”. If there are still more data to send, I send an “!”. Again, I have the actual size of the buffer one char extra, so I always have a reserved space for the end or continue chars even when the string is of size 50. Because of this implementation I was able to keep my “done” and “continue sending” down to a single character.

Usage

These instructions assume the user is in the main directory of this project

1. To build the program simply type:

   “make”

2. To run the server, in one terminal type:

   “bin/myserver 1234”

3. Next run the client in a different terminal with:

   “bin/myclient 127.0.0.1 1234”

4. When one wants to clean the projects simply type:

   “make clean”

To use the included test file, follow these instructions:

1. Build the programs with:

   “make”

2. In the bin directory, type:

   “python3 test.py”