Project #1

Implement a TCP protocol

CMPS 329

**Description:**

Your task for this project is to implement a simple TCP protocol following an RFC-style specification. You will write a server application that listens on TCP port 8008. You must use port 8008. You will also write a corresponding client application that connects to this port to run the commands that the server implements.

The server can run in the foreground and should take no arguments. It should print informational messages to the screen as clients connect. The server can be single-threaded and only accept one client connection at a time. If you wish, you can make it multi-threaded or use fork() to spawn new processes, which will enable it to service multiple clients simultaneously. Example:

[ray@cmps329 ~]$ ./server

The client should take two arguments, the IP and port number of the server. The client should also run in the foreground and print informational messages as it runs different commands over the TCP connection. Your client should run each command the server implements at least once. Example:

[ray@cmps329 ~]$ ./client 147.174.76.233 8008

**Requirements**

You may work in groups of two, or by yourself. You can write your programs in C/C++ or python. I will compile/run them on a standard Centos Linux server using GNU gcc/g++ or the python interpreter. While you can develop on Windows, OSX, or Linux, be sure that your programs compile/run on Linux. Programs that don’t compile get zero points. If you need access to a Linux account on campus, let me know. Your code should use the standard socket system calls discussed in class. Refrain from using any third party network libraries.

You will turn in a zip or tar.gz file containing all source code for the server and client. It should have server.c, client.c, and supporting header files if needed. It could also contain a README file that contains any other notes or compiling instructions.

In class, you’ll see an example of the server and client in action.

Your server must implement the following commands.

0. Initial connection

When a client initially connects, the server MUST announce itself by printing out a status code 200 followed by the name of the protocol and it’s version. Example:

Server: 200 SRP (Simple Ray Protocol) version 1.0 ready

1. HELO

Upon successful connection, a client MUST say hello. The client MUST send the string “HELO <client name>” where <client name> is the client’s hostname.

The server MUST respond to the HELO command with one of two statuses. Status 210 indicates the server received the hello and is ready for more commands. Status 510 indicates the server is having an issue and the server should terminate the TCP connection. The server SHOULD send further information after the status code. Examples:

Client: HELO raypc.fayard.selu.edu

Server: 210 Hello raypc.fayard.selu.edu, pleased to meet you.

Client: HELO raypc.fayard.selu.edu

Server: 510 Sorry I can not service your request at this time.

Connection closed.

2. REQTIME

The client can request the server’s current time with the REQTIME command. The server MUST respond with one of two statuses. Status 220 MUST be followed by the current time, in ISO8601 format **hh:mm:ss**. Status 520 indicates the server is having problems getting the current time. Examples:

Client: REQTIME

Server: 220 22:23:14

Client: REQTIME

Server: 520 Sorry, time is not available right now

3. REQDATE

The client can request the server’s current date with the REQDATE command. The server MUST respond with one of two statuses. Status 230 MUST be followed by the current date, in ISO8601 format **YYYY-MM-DD**. Status 530 indicates the server is having problems getting the current time. Examples:

Client: REQDATE

Server: 230 2014-01-29

Client: REQDATE

Server: 530 Sorry, date is not available right now

4. ECHO

The client can request the server echo back a text string using the ECHO command. This is useful when debugging a client. The server MUST respond with one of two statuses. Status 240 MUST be followed with the string that the client originally sent. Status 540 indicates the server is having problems with the ECHO command.

Client: ECHO this is some random text.

Server: 240 this is some random text.

Client: ECHO this is some random text.

Server: 540 Sorry I can’t seem to echo right now!

5. REQIP

The client can request the server print the client’s IP address. This is useful to find the client’s real IP if behind a NAT router. The server MUST respond with one of two statuses. Status 250 MUST be followed with the client’s IP address. Status 550 indicates the server is having problems getting the client’s IP address.

Client: REQIP

Server: 250 147.174.73.237

Client: REQIP

Server: 550 Sorry, I can’t get your IP address.

6. BYE

The client can request the server end the TCP connection. The server MUST respond with status 600 and immediately close the connection.

Client: BYE

Server: 600 See ya later!