Program Assignment 1

CMSC 417 Section 0101 Fall 2015

Feb. 3 2015

1 Deadline

Feb. 15 , 2015. This assignment is intended to make sure that you have CORE installed and configured properly - directions for this have been posted to Piazza. Post general questions to Piazza.

2 Objective

In this project, you will make an "echo client". The goals of this mini-project are:

- to de-mystify the process of generating code for networks.
- to test out the submit server and NRL CORE.
- to familiarize you with basic shell scripting

3 Specification

For this project, an echo server program will be provided by us and you will write a client program which will communicate with the server using socket. You must write the program in C.

1. Connect using TCP to argv[1] with port argv[2].
2. Read stdin and write that stuff to the TCP socket until stdin ends (end of file).
3. Shutdown the socket for writing. See shutdown(2). (i.e., man shutdown)
4. Read what the TCP socket has and write that stuff to stdout until the socket ends (end of file because the other side closed). **No additional output should be sent to stdout.**
5. Exit. Important! If you don't exit, the tests will fail.
6. Your source file should be named: "p1.c" and your executable should be named "p1".

You must also write a shell script that demonstrates functionality of your code. In general, your shell script must do **exactly** the following:

1. Compile your code using your Makefile.

2. Read a comma separated list of $< host >, < port >, < data >, < name >$ (with no header) from the file specified as the first argument.

3. For each line in the file that is read, your shell script should run your echo client with the host and port, pipe "data" (the string that is send by your echo client) into your client, and append the output of your client to $< name >$.out (creating $< name >$.out if it does not exist.

Your shell script must be named *tests.sh*.

4 Example Tests

4.1 Server

```
<Server Setup>
chmod a+x server  (run this command line if needed)
./server <port>

<Client Test>
echo "echo test" | ./p1 localhost <port>
echo "GET /" | ./p1 www.google.com 80
cat p1.c | ./p1 localhost <port>
```

Your echo client will get exactly what you send to the echo server. If you ask for a web page, expect to see </html> or </script> at the end.

5 Over-specification

Don't make this harder on yourself than it has to be. You don't have to alternate reading stdin and writing. You don't have to call select() or poll() to figure out what's available. Non-blocking I/O is not needed.

You may not use any glorious library that makes everything easy that does not come with the language.

Stdin may be arbitrarily large. What comes from the server may be arbitrarily large. (That is, there's a reason step 2 is a composite of two things.) In general, don't ask me how big a buffer needs to be.

Your shell script must work in bash.

6 Hint

Try to break your code. You learn more about the behavior of your code from a single failure than from 1,000,000 successes.

Also, use **MEANINGFUL** error messages. If your code does not run, I will not try to debug it for you; I will post the output to the comments section on grades.

7 To Turn In

READ THIS CAREFULLY! NO REGRADE WILL BE GRANTED FOR FAILURE TO FOLLOW THESE DIRECTIONS

- a single zip file p1.zip containing **NO SUB-FOLDERS**
- p1.zip should contain the source code (p1.c), the Makefile (Makefile), and your shell script (*tests.sh*), an example config file (config.csv) with three host,port,data combinations of your choosing, demonstrating the functionality of your code (you must name the outputs t1.out, t2.out, t3.out).

Your code **MUST RUN** in CORE.

8 Finding Code On-line

If you find precisely this online, don't copy it. Cite sources in the top of your file. Basic documentation should be 75% of the code here.

We reserve the right to run your code through plagiarism detection software and may do so.