Program Assignment 1 (PA1)

CMSC 417 Section 0201 Spring 2017

Jan. 31 2017

1 Deadline

Feb. 14, 2017. This assignment is intended to make sure everything works. Please 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, NRL Core and class accounts on submit.cs.umd.edu (if needed).

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 sockets. The echo server has been compiled in NRL Core.

1. You MUST complete this assignment using C
2. Your code must compile and run on the submit server.
3. Connect using TCP to argv[1] with port argv[2].
4. Read stdin and write that stuff to the TCP socket until stdin ends (end of file).
5. Shutdown the socket for writing. See shutdown(2). (i.e., man shutdown)
6. 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.**
7. Exit. Important! If you don't exit, the tests will fail.

4 Example Tests

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

<Client Test>
echo "echo test" | ./pa1 localhost <port>
cat pa1.c | ./pa1 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 Overspecification

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. (For example, glib and the apache portable runtime are forbidden.) If you wrote it yourself, you're welcome to turn it in, though.

Stdin may be arbitrarily large. What comes from the server may be arbitrarily large. In general, don't ask me how big a buffer needs to be.

6 Hint

The send() and recv() functions are not necessary; read() and write() are sufficient.

Start early! The submit server will not give you much information about any failed tests. This is a senior level course, and we expect you to be able to debug your own code. The submit server will tell you if you should still be looking for bugs, but it won't help you find them.

7 To Turnin

- Please turn in "pa1.zip" a zip file (no sub folders) containing only pa1.c.
- If you have any special grading circumstances, please email the TA, and include a grading_notes.txt file in your submission.

Your code MUST RUN on the submit server. Errors can occur. You may need more #include lines than you expect or they must be in the order on the man page. It is not sufficient that your code work on some machine you happen to know about. **You will NOT receive credit for any submission that does not run on the submit server**

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.

9 ProTips

9.1 Length of Response

One of the most common mistakes students make in this project is assuming that the length of the data that the server sends back will be the same length as the data that the echo client sends to the server. This is **NOT** true. Assuming the data are the same length can lead to your code either hanging (if it's expecting more data) or failing to report all of the output (if it's closing prematurely). **Note**: the server we provide you for testing will always send back exactly what your client sends it, but we **will** test your client with servers that do not behave in this fashion.

9.2 Reading Input

Another mistake students commonly make is trying to read all of the input from STDIN before writing anything to the socket (and similarly trying to read everything from the socket before writing to STDOUT). While this usually won't produce incorrect results, it is unnecessary for the purpose of this assignment and requires a lot of extra work. **Hint**: if you're calling malloc or calloc, you are doing something wrong. **Hint Hint**: if you are calling realloc, you are doing something *really* wrong.

9.3 Closing the Socket

The most common question I receive in office hours is: how do I know when I've read all of the data from the socket? I will leave figuring this out as part of the assignment... **Hint**: it has to do with the return value of calling *read()* on the receive end of the socket.