CS 447 : Networks and Data Communications
Programming Assignment #01

Total Points: 150

Assigned Date	: Thursday, February 06, 2019
Due Date	: Thursday, February 20, 2019 @ 11:59:59 a.m.

Overview

Your first programming assignment is to **implement a basic client/server application** using the socket interface. There are several objectives of this assignment. These are:

a. to get yourself familiarized working with the socket programming basics;
b. to understand the ordering of the socket interface primitives;
c. to get you exposed to linux system calls (if you already haven't);
d. to gain a basic understanding of network protocols; and
e. to set yourself up for the rest of the course.

Back Story

Professor Calculus, returning from his latest conference, has just learned about the "*power of the cloud*" and now wants to move his scientific calculator application to "the cloud". He has learned enough networking (to get by at least) and is prioritizing **performance over reliability** for his application. He also wants his cloud-based calculator to accommodate requests from **more than one person** at a time. Given that this is his first time network programming, he just wants to provide support for only three basic operations:

1. The power function (**`POWER`** (x^e)) for a given base x and an exponent e;
2. The cubic square root function (**`CUBE`** ($\sqrt[3]{x}$)) for a argument x; and
3. The factorial function (**`FACT`** ($x!$)).

Technical Requirements

- Server should be capable of accepting requests from UDP clients.
- Server should support **multi-threading** (more than one client should be capable of using the cloud-calculator).
- Your protocol interaction should adhere to the following specifications.
- **Client Commands:**
    1. `HELO` <*`server-hostname`*> – This is the **first** command issued by the `client` (→ `server`). A successful/valid exchange is marked by reply code (see section on reply codes below) `200`.

2. `HELP` – This command can be issued anytime after the `HELO` command. A successful/valid exchange is marked by reply code `200`.
3. `CALC` – This command must be issued before any of the calculator functions (`POWER/CUBE/FACT`) can be used. Reply code `200`.
4. `POWER` <*x*><*e*> – The `POWER` command requests x^e calculation. Reply code `250`.
5. `CUBE` <*x*> – The `CUBE` command requests the cubic square root of a the given argument `x`. Reply code `250`.
6. `FACT` <*x*> – The `FACT` command requests the factorial value of `x`. Reply code `250`.
7. `BYE` <*`server-hostname`*> – This command closes the connection and requests a graceful exit. This command can be issued anytime during the interaction. The correct server reply code is `200`.

- **Server Reply Codes:**

  1. `200` Command Success. The command success reply code is issued only when the interaction happens according to the correct specification. Examples:
     - `200 HELO 10.1.2.3(UDP)` – If the `HELO` command is issued as the first command.
     - `200 BYE 10.1.2.3(UDP)` – If the `BYE` command is issued.
     - `200` <*menu*> – If the `HELP` command is issued after `HELO`. the calculator menu is sent with this reply code.
     - `200 CALC ready!` – If the `CALC` command is issued at the correct point of interaction.
  2. `250` <answer> – This reply code is issued in response to a correct calculator command syntax received in the previous message from client. `answer` is the calculated value.
  3. `500` – Syntax Error, command unrecognized.
  4. `501` – Syntax error in parameters or arguments.
  5. `503` – Bad sequence of commands.

Functional Requirements

1. IP addresses/hostnames and port numbers should not be hard coded.
   - Your server executable will following the following execution signature:
     `./server <udp-port-number>`
   - Your client executable will accept two command line arguments as follows (assume your client to know the correct hostname port-number combo):
     `./client <server-hostname> <server-port>`
2. `client-server` connection is `UDP`-based (unreliable).
3. I will test with **at least** 2 simultaneous client connections, thus, your server should be multi-threaded.
4. Client's should exit gracefully. Server process is permitted to be forcefully killed.
5. Here's a sample (non-comprehensive) `UDP` interaction. Assume the client's IP address is 146.163.150.234 and running UDP and the server's hostname is `calco`.

```
     Client       Server
 HELO calco  →
             ←   200 HELO 146.163.150.234(UDP)
    CUBE 64  →
             ←   503 CALC before CUBE
       HELP  →
             ←   200 <menu-sent-back>
       CALC  →
             ←   200 CALC ready!
    CUBE 64  →
             ←   200 4
  BYE calco  →
             ←   200 BYE 146.163.150.234(UDP)
```

6. Your client and server should be able to run on two separate end systems. Bare minimum, you should verify an interaction between a client running on a lab machine (EB 1036 dual boots to Linux) and the "`cs home`" server and vice-versa. Depending on the firewall rules, you might also be able to test from off-campus using your own laptop/desktop as one end system as well.
7. At the end of your implementation, you should be able to:
   - Compile and run your code in a linux machine. Include a `readme` file with clear compilation instructions.
   - Run your server program first.
   - Run one or more clients to connect to the server.
   - Perform calculator functionality while meeting the technical requirements mentioned above.
   - Exit the client(s) gracefully.

Instructions

- This is an individual assignment. **Do your own work**.
- **Start early!! Take backups of your code often!!**. Use of a version control software is highly recommended! **Note**: Per course policy (see syllabus), keep any online repositories private if you intend to use them for course material storage.
- Make sure to test your program properly before your final submission. It is **highly** recommended to test build and run your submission on the home server, **home.cs.siue.edu**.
- You may use any programming language of your choice out of `C, C++, Java, or Python`. However, your code **must** compile and run on Linux.
- Absolutely **DO NOT** include executables with your submissions.
- A **Makefile** is mandatory. Whether or not your program needs to be compiled, have it echo instructions to run the program.
- Follow a good coding standard. Use the Google C++ coding standard found here `http://goo.gl/1rC1o`, if you don't already follow one.
- The report part of your solution must be produced using a word processor. LaTeX is highly recommended but not a requirement.
- Your final report should be in **PDF** format. No exceptions.
- Any figures, graphs, plots, etc., should also be produced using appropriate computer applications. If using LaTeX, the pgfplots package is very useful for making all sorts of graphs.

- The due date of this assignment is **Thursday, February 20, 2019 @ 11:59:59 a.m.** A dropbox will be opened for submission on Moodle.

Deliverables

A complete solution comprises of:

- A short report (max 5 pages) of the design and implementation of your system. Your report should include the followings:
  - Introduction
  - Design choices and protocol/reply codes used.
  - The output of a sample run with proprerly annotated screenshots. where applicable.
  - Summary and Issues encountered (if applicable).
- A short `README` file with compilation and run instructions.
- A `makefile` to compile your code, especially if it involves compiling multiple executables with flag options.
- A compressed tarball of the directory containing your source code, report, REAME, and makefile. **Absolutely do not** include executables, folders created by your programs, your version control repositories, or your test emails in this tarball. To create a compressed tarball of the directory source, use the following command: `tar -zcvf siue-id-pr1.tar.gz source/`.
  e.g. `tar -zcvf tgamage-pr1.tar.gz PR01/`.
- File formatting standards (**`pdf, README, .email, .txt, .tar.gz`**) will be strictly monitored and is subject to penalties.

Collaborating on ideas or answering questions is always encouraged. Most times, I find that you learn a lot from your peers. However, do not share/copy/duplicate code from others. If you use code found online, remember to site their source in your report. Issues related to academic integrity and plagiarism have **ZERO** tolerance.

Useful Resources

- Linux Man pages – found in all linux distributions
- Beej's Guide to Network Programming – A pretty thorough free online tutorial on basic network programming `http://beej.us/guide/bgnet/output/print/bgnet_USLetter.pdf`