CS447 : Networks and Data Communications

Programming Assignment #0

Total Points: 75

Assigned Date : Wednesday, February 01, 2023
Due Date : Wednesday, February 15, 2023 @ 01:29:59 p.m.

Overview

Your first programming assignment is to **implement a basic client/server application** using the socket interface. This programming assignment is meant as a *setup assignment* for PR01 and PR02, and has the following learning objectives:

a. to get your feet wet on 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

Captain Haddock and his crew are not big fans of unit conversions, but regularly have to convert from/to imperial to/from metric during their sea voyages. *Professor Calculus*, who recently took CS447 at SIUE, thinks he has a solution. Calculus agreed to create an online unit conversion calculator application for Haddock that has a **rapid response** but not necessarily reliable; Haddock is fine with that as he is *all about speed*. Not only that, Calculus agrees to create this calculator in a way that **more than one person** can use it at the same time. For this initial release, Calculus plans to support conversions between 14 types of units, grouped into 4 modes.

1. **`AREA`** – Conversions between Square Meters, Square Miles, Square Inches, and Square Foot
2. **`VOL`** – Conversions between Liters, US Gallons, Imperial Gallons, and Cubic Meters
3. **`WGT`** – Conversions between Kilograms, Pounds, and Carrots
4. **`TEMP`** – Conversions between Celsius, Fahrenheit, and Kelvin

Technical Requirements

- All `server` ↔ `client` interactions should use UDP as the underlying transport layer protocol.
- Server should be capable of accepting requests multiple concurrent UDP clients. In other words, more than one client should be able to use the calculator without interfering with each others calculations.
- 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`).
  2. `HELP` – This command can be issued anytime after the `HELO` command.

3. There are 4 *mode selection* commands – `AREA`, `VOL`, `WGT`, and `TEMP` – which correspond to *Area*, *Volume*, *Weight*, and *Temperature* respectively. A successful mode selection is acknowledged by the server with a mode specific unique reply code.
4. Each mode has a certain number of *sub-commands*. All sub-commands have the following command syntax format: `<from-unit> <to-unit> <value>`
   (a) The `AREA` mode should support conversions between `SQRMT`, `SQRML`, `SQRIN`, and `SQRFT` (corresponding to *Square Meters, Square Miles, Square Inch, and Square Foot* respectively).
   (b) The `VOL` mode should support conversions between `LTR`, `GALNU`, `GALNI`, and `CUBM` (corresponding to *Liters, US Gallons, Imperial Gallons, and Cubic Meters* respectively).
   (c) The `WGT`† mode should support conversions between `KILO`, `PND`, and `CART` (corresponding to *Kilograms, Pounds, and Carrot* respectively).
   (d) The `TEMP`† mode should support conversions between `CELS`, `FAHR`, and `KELV` (corresponding to *Celsius, Fahrenheit, and Kelvin* respectively).
5. Users may switch the mode anytime during the interaction.
6. `BYE <server-hostname>` – This command closes the connection and requests a graceful exit. Similar to mode switching, users may request a graceful exit anytime during the interaction.

- **Server Reply Codes:**
  1. `200/210/220/230`†`/240`† Command Success. The command success reply code is issued only when the interaction happens according to the correct specification. Examples:
     – `200 HELO 192.168.0.11(UDP)` – If the `HELO` command is issued as the first command. **Note**: The client's IP address and the connection type should be derived from the incoming connection (and should not be hard coded).
     – `200 <menu>` – If the `HELP` command is issued after `HELO`. the calculator menu is sent with this reply code.
     – `210 AREA Mode ready!` – If the `AREA` command is issued at the correct point of interaction. Reply codes, `220,230`†`,240`† correspond to `VOL,WGT,TEMP` mode selections respectively.
  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.
  6. `504` – Bad conversion request.

Functional Requirements

1. IP addresses/hostnames and port numbers should not be hard coded. They are provide at runtime through configuration files‡ (e.g. `server.conf`) present in the same working directory.
   - Your server executable will accept a single runtime argument through a configuration file as follows: `./server server.conf`

     server.conf
     `UDP_PORT=xxxx`
   - Similarly, your client executable will two runtime arguments through a configuration file as follows: `./client client.conf`

     client.conf

†Extra Credit
‡**Note**: A configuration file is simply a text file with a `.conf` extension.

```
SERVER_IP=y.y.y.y
SERVER_PORT=xxxx
```

This are considered the default execution behavior for the servers and clients. No deviations allowed.

2. We will test with **at least** 2+ concurrent client connections, thus, your server should be multi-threaded.
3. Client's should exit gracefully. Server process is permitted to be forcefully killed.
4. Here's a sample (non-comprehensive) interaction. Assume the client's IP address is 192.168.0.11 and running UDP and the server's hostname is `calco`. Note: The server must recognize client IP and connection type.

```
    Client       Server
HELO calco   →
             ←   200 HELO 192.168.0.11(UDP)
LTR CUBM 2   →
             ←   503 Wrong Command Order:  Mode not selected
      HELP   →
             ←   200 <menu-sent-back>
       VOL   →
             ←   230 VOLUME Mode ready!
CUBM LTR 2   →
             ←   250 2000
 BYE calco   →
             ←   200 BYE 192.168.0.11(UDP)
```

5. 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 one `zone server` container while the server is running on another.
6. 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 and any non-standard Linux software requirements.
   - 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.

Extra Credit†

- The `WGT` and `TEMP` modes are for extra credit and are not considered as part of the core implementation of this assignment.
- Only assignments that grade `90%+` will be considered for extra credit evaluation. In other words, the extra credit will not be used to supplement deficiencies in your core implementation.
- The extra credit has no partial credit. You must implement both modes along with all corresponding sub-commands and reply codes correctly to be eligible for extra credit.

Instructions

- **Start early!!**. This is a simple yet fairly loaded assignment.
- **Take backups of your code often!!**.
- Implementation language must be C/C++.
- Use the `talker.c` and `listener.c` from the Beej's Guide to Network Programming (see Resources section below) as your **ONLY** permissible starter code. Absolutely no deviations. Although this code is earmarked as a `C` code, it will readily compile for `C++` as is.

- Follow a good coding standard. Use one of Google's coding standard found here https://google.github.io/styleguide/, if you don't already follow one.
- Your code must compile and run on a typical Linux setup. Neither the instructor nor his graders will use (or entertain the use of) any IDE to test your implementation. Be sure to test command line compilation and run before submission.
- **Absolutely do not** include executables, folders created by your programs, hidden files, version control repositories, or any irrelevant files in this tarball. All project relevant file formatting standards (**`PDF`, `README`, `.txt`, `.tgz`**) will be strictly monitored and are subject to penalties.
- The due date of this assignment is **Wednesday, February 15, 2023 @ 01:29:59 p.m.**. A Moodle dropbox will be opened for submission.
- Based on past student experience, multi-threading, especially handling multiple parallel UDP clients at the server end, is not trivial. I suggest you to first get a single-threaded version working correctly and then think about extending it to multi-threading.
- This assignment can be fully developed using the socket API of your programming language and basic I/O API. Use of other packages/libraries without the instructors permission is not permitted.

Deliverables

A complete solution comprises of:

- A short report of the design and implementation of your system. The report should be **`PDF`** format. At a minimum, your report should include the following sections:
  - Introduction: Your objective and what you hope to gain from the assignment.
  - Overall design, specific design choices, and reply codes used. Specifically explain how you achieved concurrent client support.
  - The output of a sample run. Include plenty screenshots wherever applicable. In situations were we can't verify expected behavior, your screenshots maybe considered for partial credit.
  - Summary and Issues encountered. What you were able to achieve from your own objectives (from the introduction) as well as project specifications. Make sure to explicitly list functionality you failed to implement (or buggy).
- A compressed tarball that contains:
  - a directory containing (only) your source code and config files. **Do not** include executables, folders created by your programs, or any other files not specifically listed here as required.
  - A short `README` file with compilation and run instructions.
  - A `makefile` (**mandatory**) to compile your code especially if it involves compiling multiple executables with flag options.

> To create a compressed tarball of the directory source, use the following command:
> `tar -zcvf siue-id-p0.tgz source/`.
> e.g. `tar -zcvf tgamage-p0.tgz P0/`

Collaborating on ideas or answering each other's 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, including online sources AND AI assisted code generators. If you find a useful article or a source code online, firstly cite it in your report, and secondly, get the idea from it and do your own implementation. Absolutely do not directly copy-paste code from online sources.
The exercise is meant for you to learn network programming, not to test your googling abilities. The instructor actively uses MOSS http://theory.stanford.edu/~aiken/moss/ to check for software similarity. Issues related to academic integrity and plagiarism have **ZERO** tolerance and will result in a failing grade for the course.

Some 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 for C/C++ https://beej.us/guide/bgnet/
- Linux Socket Programming In C++ – https://tldp.org/LDP/LG/issue74/tougher.html
- The Linux HOWTO Page on Socket Programming – https://www.linuxhowtos.org/C_C++/socket.htm