CS447-1 : Networks and Data Communications *(sec. 01)*

Programming Assignment #01

Total Points: 150

Assigned Date : Wednesday, February 02, 2022
Due Date : Wednesday, February 23, 2022 @ 11:59:59 a.m.

Overview

Your first programming assignment this semester is to **implement an elementary network news application** using the socket API. This assignment is a multi-objective assignment and is fairly loaded. Moreover, your PR02 is expected to be an extension of PR01. Thus, start early. The assignment objectives are:

a. to start developing socket programming skills and knowledge;
b. to reinforce the concept of "*protocol*" using hands-on programming;
c. to refresh multi-threading, system calls, and other systems programming concepts;
d. to learn how to implement protocols according to their RFC specifications; and
e. to gain experience in developing application layer network programs.

Take note that the implementation language is fixed to C/C++. Here the back story.

Back Story

As it turns out, news doesn't travel fast enough for Captain Haddock when he is on his routine sea voyages. He just found out that there was a recent public online auction in which a rare mini replica of the *Unicorn* was sold. He reaches out to his trusty computer scientist friend Mik for some advice. "*I think I can build you a network news service, Haddock*", said Mik. He promises Haddock that this new news solution will provide reliable delivery and will even allow all of his crew-mates to also retrieve articles they are interested in without a hitch.

Technical Requirements

1. You will need to write a client-server application to support Mik's network news system.
2. Your `reader` → `server` interaction as well as the `server` → `reader` interaction should follow the NNTP protocol
3. Your `server` implementation must support the following capabilities and, at minimum, the listed set of associated commands:
   - **`READER`** – **`GROUP`**, **`LISTGROUP`**, **`NEXT`**, **`ARTICLE`**, and **`DATE`**
   - **`HDR`** – **`HDR`** *(first form)*, and **`LIST HEADERS`**
   - **`LIST`** – **`ACTIVE`** and **`NEWSGROUP`**
     - The **`LIST`** command accepts ***`wildmats`*** syntax for arguments (see RFC Sec. 4.1) Thus, it is highly recommended to incorporate `regex` in your implementation (instead of trying to parse manually).

4. Additionally, the following basic commands should be supported: **`CAPABILITIES`**, **`HELP`**, and **`QUIT`**. A description of each of these commands along with examples can be found in the NNTP RFC https://tools.ietf.org/html/rfc3977.
5. Closely related to the NNTP commands are the corresponding reply codes. Read Section 3.2 of the NNTP RFC, select appropriate reply codes, and implement them. Based on Graders' preliminary testing, they are expecting you'd use at least 10-15 reply codes in your implementations. Explain your reply code selection and the justification in your report.
6. Articles at the `server` are immutable, meaning, `client/reader` gets a copy each time an article is fetched.
7. More than one `reader` should be capable of reading news articles without interrupting any other `reader`. As such, your server needs to be **multi-threaded**.
8. All interactions between the `readers` ↔ `server` should run over TCP.

Logistics

1. IP addresses/hostnames and port numbers should not be hard coded.
   - Your `server` executable will accept the necessary listening port from a configuration file[†] (e.g. `server.conf`) present in the same working directory. For example:

```
./server server.conf
```

```
#server.conf
NNTP_PORT=
```

   - Your client executable also follows a similar pattern and accepts the server socket address from a configuration file (e.g. `client.conf`)

```
./client client.conf
```

```
#client.conf
SERVER_IP=
SERVER_PORT=
```

2. Client(s) and server should be able to run on two separate end systems but still communicate with each other. Use the zone server `zone.cs.siue.edu` to spawn multiple containers to test simulated over-the-network communication behavior. At minimum, at least two concurrent readers should be able to read articles without any interruption or interference.
3. All clients should exit gracefully. Server process is permitted to be forcefully killed.
4. Use the following article management strategy:
   - All articles are stored in a folder named **`db`** under `server's` present working directory (`$pwd`).
   - Each **`NEWSGROUP`** has it's own folder inside the **`db`** folder. **`ARTICLES`** are stored under these sub folders.
   - All `ARTICLES` have numeric file names. For implementation simplicity, `ARTICLES` can be stored as `.txt` files. (NOTE: This might change in PR02 and beyond).
     - Each `ARTICLE` has a `HEAD` separating the `BODY` by a single blank line (i.e. `CRLF`).
   - A sample **`db`** folder will be provided to your (thru Discord) for testing purposes.
5. Here's a sample `article.txt` file stored at the `server`.

Answer

```
Feed:  Wired | Science
Title:  To Learn More Quickly, Brain Cells Break Their DNA
```

[†]**Note**: A configuration file is simply a text file with a `.conf` extension.

```
Author:  Jordana Cepelewicz
Date:  Sat, 05 Feb 2022 07:00:00 -0500
Link:  https://www.wired.com/story/to-learn-more-quickly-brain-cells-break-their-dna

DNA double-strand breaks are associated with cancer and aging.  A new study shows
neurons can use them to quickly express genes related to learning and memory.
```

6. Print all reply codes, on all interactions, to the standard output (only).
7. **Minimum Testing Plan**: At the end of your implementation, you should be able to:
   - Compile and run your code on a typical Linux machine.
   - Run your server program(s) first.
   - Run several clients concurrently to connect to the server and be able make independent progress without interference from each other to read articles.
   - Exit the client(s) gracefully.

Instructions

- **Start early!!**. This is a fairly loaded assignment.
- **Take backups of your code often!!**.
- 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.
- Implementation language must be C/C++.
- **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, .tar.gz`**) will be strictly monitored and are subject to penalties.
- The due date of this assignment is **Wednesday, February 23, 2022 @ 11:59:59 a.m.**. A Moodle dropbox will be opened for submission.
- Based on past student experience, multi-threading is not as obvious as it may look. 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.
  - 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. Python based submissions should use the `makefile` to echo the content of your `README` file.

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/`

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. 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 direclty 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.

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`
- Network News Transfer Protocol (NNTP) RFC #3977 `https://tools.ietf.org/html/rfc3977`