CS447 : Networks and Data Communications *(sec. 02)*

Programming Assignment #02

Total Points: 150

Assigned Date : Wednesday, October 20, 2021
Due Date : Wednesday, November 03, 2021 @ 02:59:59 p.m.

Overview

For your second project, you will extend your network news application from PR01 and make servers communicate among themselves in a **P2P** model. Specifically, PR02 adds the following new major capabilities:

- Ability to **`POST`** articles from `client-to-server`, and
- `Server-to-Server` communication to synchronize newsgroups across different servers.

Additionally, this assignment provides you with an opportunity to learn about `patch` files and how to use them in your workflow. Here is a fairly straightforward and simple introduction to `patch` files for those of you who are new to the topic: https://www.shellhacks.com/create-patch-diff-command-linux/.

Technical Requirements

ALL PR01 technical requirements fully apply to the PR02. Additionally, following new requirements should be met.

1. Support the **`NEXT`** command (RFC #3977 Sec. 6.1.4).
2. Support the **`NEWNEWS`** command (RFC #3977 Sec. 7.4.1).
3. Support `server` mode switching using the **`MODE READER`** command (RFC #3977 Sec. 5.3).
   - The easiest way to think about server modes is to consider interactions with a `server` in *"Transit"* mode akin to a HTTP `GET` request, and an interaction with a `server` in **"`Reader`"** mode akin to a HTTP `POST` request.
   - The default mode is *"Transit"*.
4. Support the **`POST`** command (RFC #3977 Sec. 6.3.1).

Note:

- Article posting is only possible when the `server` is in the **`READER`** mode.
- Not all servers necessarily support mode-switching. Consequently, not all servers will

allow article posting. This is indicated by a `server.conf` option. *(See Logistics.#6 below)*

5. Support the **`IHAVE`** command (RFC #3977 Sec. 6.3.2) to allow servers to automatically share/sync newly posted articles with other servers.

   Note:

   - The newsgroup database (`db`) is not necessarily replicated one-to-one across all servers. As such, different servers will have different **`NEWSGROUPS`**. Synchronization is only performed when both the sending and receiving server share the same `NEWSGROUP` in its `db`. In otherwords, **`IHAVE`** command will not create new newsgroups.
   - Each server is only aware of (and connected to) other servers listed on its `server.conf` configuration file. *(See Logistics.#6 below)*

6. Server Incident Management: For auditing purposes, each server should keep an active log file (named `srv.log`) that records all interactions. Every time a server sends or receives a message, a log entry is added to this log file. More specifically, each log entry (single line) has the following format:

   Answer

   ```
   timestamp from-ip to-ip protocol-command numeric reply short-description
   ```

Logistics

All PR01 Logistics requirements applies to PR02. The following additional requirements are listed mostly for clarification purposes.

1. Address any pending issues from PR01 before starting PR02 implementation.
2. All `server-to-server` communications should be automated. In other words, there is no user interaction with servers (through the terminal or otherwise).
3. All file creations, including log files, should be done programmatically.
4. Print all reply codes on all interactions to the standard output at the `server`.
5. Submit only `patch` files. We will `patch` your PR01 with the provided patch files and consider your "*patched*" PR01 as PR02.

   Note:

   Do not submit any source code for PR02. We are strictly looking for `patch` files to gauge how much of your PR01 you had to change to accommodate PR02 requirements, and what pending issues did you resolve from your PR01 submission. Any source code found in PR02 submissions will be immediately discarded (and will not be considered in grading).

6. Your augmented `server.conf` files will look similar to the following:

   `server.conf`

   ```
   NNTP_PORT=
   MODE_SWITCH= //valid arguments ON and OFF
   RSOCK_ADDR= // in IP:PORT format.
   ```

```
⋮
RSOCK_ADDR=
```

Here, **RSOCK_ADDR** is the socket address of a remote (p2p) server. The **MODE_SWITCH** option signifies if the server supports mode switching (or not). The default is **OFF** indicating a server in strict *"Transit"* mode operation.

7. At the end of your implementation, you should be able to:
   - Compile and run your code on a typical Linux machine(s).
   - Run your `server(s)` first. It's advised to at least test with 3 different servers running on their own container.
   - Run several concurrent `clients` and connect them with with different `servers`.
   - Perform article retrieval (just as in PR01) and post new articles.
   - Synchronize/share articles among servers.
   - Retrieve new articles (as `client`).
   - Exit the `client`(s) gracefully.

Instructions

- **Start early!!**. This is a fairly loaded assignment for 2 weeks.
- **Take backups of your code often!!**. Practice good version control habits.
- 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, source code, folders created by your programs, hidden files, version control repositories, or any irrelevant files in this tarball. As mentioned previously, we are only looking for patch files. All project relevant file formatting standards (**PDF, README, .txt, .tar.gz**) will be strictly monitored and are subject to penalties.
- Test and ensure your patch files work as intended before your final submission.
- The due date of this assignment is **Wednesday, November 03, 2021 @ 02:59:59 p.m.**. A Moodle dropbox will be opened for submission.
- 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 patch files.
  - 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-pr2.tar.gz source/`.
e.g. `tar -zcvf tgamage-pr1.tar.gz PR02/`

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. The exercise is meant for you to learn network programming, not to test your googling abilities. 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 for C/C++ https://beej.us/guide/bgnet/
- The Linux HOWTO Page on Socket Programming – https://www.linuxhowtos.org/C_C++/socket.htm
- Python
  - Socket Programming in Python (Guide) https://realpython.com/python-sockets/
  - Socket Programming HOWTO https://docs.python.org/3/howto/sockets.html
- The Java Tutorials – All About Sockets https://docs.oracle.com/javase/tutorial/networking/sockets/index.html
- Network News Transfer Protocol (NNTP) RFC #3977 https://tools.ietf.org/html/rfc3977

If you use any other resources, make sure to cite those in your report. Using online resources does not mean you are allowed to copy and use someone else's code for your purpose. Such incidents, if detected, will be treated as academic dishonesty.