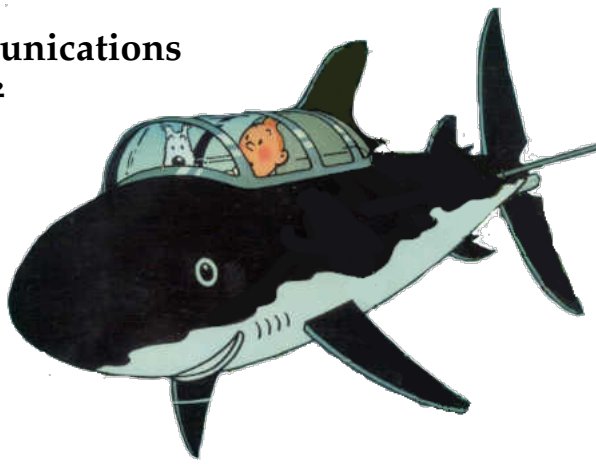
CS 447 : Networks and Data Communications

Programming Assignment #02

Total Points: 150



Assigned Date : Wednesday, March 14, 2018
Due Date : Wednesday, March 28, 2018 @ 11:59:59 a.m.

Overview

Your second programming assignment is to **implement a Internet Relay Chat (IRC) application** using the socket interface. The IRC protocol description is fairly lengthy and, over the years, have expanded into multiple RFCs. Given our 2 week window, you are not expected to implement a fully-pledged IRC application, but rather a simplified basic IRC application. The assignment objectives are:

a. to build up on your basic socket programming experience from the first assignment;
b. to further reinforce the concept of "*protocol*" using hands-on programming;
c. to learn how to read and understand RFCs; and
d. to gain experience in UDP concurrency management.

Back Story

After discovering Red Rackham's treasure, Haddock started an international shipping business. While business is booming, Haddock is finding to be a nightmare to communicate with his captains while they are on voyage. "*Blue blistering barnacles!!*". Screamed Haddock. "*These Sea-gherkin Carrier Pigeons are not fast enough. I should have read this message 2 weeks ago*". Professor Calculus, after over hearing Haddock cursing and kicking around the furniture in frustration, (again) approaches him with a solution. "*Let me build you a relay chat network Haddock. You'll never have to deal with the carrier pigeons again. You can chat with any of your captains when you want, and they can also chat with any other captains as they wish.*". Calculus said proudly. "*Everyone will know about everyone's whereabouts all the time*".

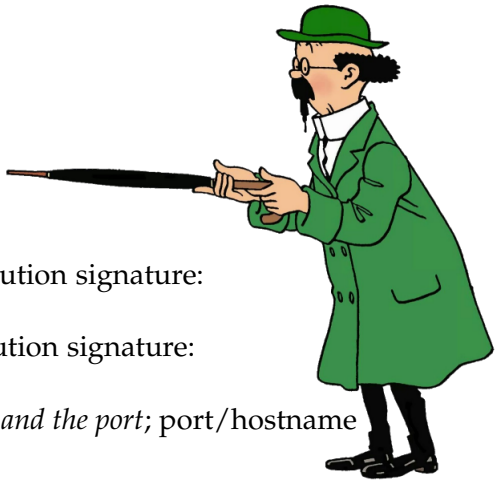
Technical Requirements

1. Just as in PR01, you will need to write a client-server application to support Haddock's IRC system.
2. Implement a single central server IRC network. `server-to-server` communication is outside the scope of this project. Concurrent multiple clients, however, should be supported.
3. Follow the protocol grammar listed under **RFC #2812 Internet Relay Chat: Client Protocol** https://tools.ietf.org/html/rfc2812 Sec. 2.3.1. You are strongly encouraged/recommended to use the `regex` support of your implementation language for efficient parsing.

4. Support the following Connection Registration (#RFC 2812 Sec. 3.1) messages: `NICK`, `USER`, `MODE` and `QUIT`.
5. Support the following Channel Operation (#RFC 2812 Sec. 3.2) messages: `JOIN`, `PART`, `MODE`, `TOPIC`, `LIST` and `QUIT`.
6. Support `PRIVMSG` (#RFC 2812 Sec. 3.3).
    - Closely related to the above message types are their corresponding reply codes. Read **RFC #2812** Sec. 5. I expect, at minimum, you will need the following reply codes: `001`, `002`, `004`, `301`, `322`, `323`, `401`, `403`, `404`, `431`, `432`, `433`, `461`, `501`, `502`. Additional useful/necessary reply codes are welcome with a proper justification in your report.
7. All `server` ↔ `client(s)` communication should be over UDP. Note that this is in direct contrast with the original specification, which implies (or intends) TCP.

    Note:

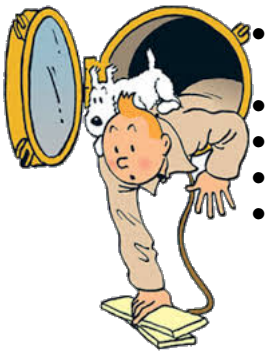
    Proper UDP multi-threading and socket management is not trivial nor straight-forward, thus you must carefully plan your design; for most part, you are to pseudo mimic TCP with UDP. Also, keep room in your overall design for a possible `server-to-server` communication expansion in PR03.
8. Trivially, more than one user should be able to chat with any number of other users at the same time.



Logistics

1. IP addresses/hostnames and port numbers should not be hard coded.
    - A typical server executable is expected to have the following execution signature: `./server <udp-listen-port>`
    - A typical client executable is expected to have the following execution signature: `./client <server-hostname> <server-port>`.
      Your may assume that your client knows the server hostname/ip address and the port; port/hostname discovery is outside the scope of this project.
2. Your client(s) should be able to interact with each other through separate end systems. Bare minimum, you should verify an interaction between one client(s) running on a lab machine(s) (EB 1036 dual boots to Linux) and another on the "`cs home`" server and vice-versa. The server may run on a third end system or on one of the end systems running a client. 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.
3. All clients should exit gracefully. Server process is permitted to be forcefully killed.
4. I will test **several** clients with different permutations of `nicknames`, `channels` and `topics`. Properly and adequately test and document (with screenshots) as many different scenarios as time permits.
    - Have a look at the following page for some *(very nicely illustrated)* sample interactions. **ABSOLUTELY** do not use any of the code listed in this page or any of the parent pages (or any other online source).
      http://chi.cs.uchicago.edu/chirc/irc_examples.html.

5. You may consider a file-based server management strategy (for users, channels, and topics). This is not a requirement, but likely the most efficient approach given the 2 week deadline. If you do decide on a file-based strategy:
   - Programmatically (not manually) create a folder, with the **`current time`** as folder name, every time the server is run and store all files created under that (new) execution inside that folder. e.g. Folder `18:23`, implies the server was run at 06:23pm and the files inside part of that execution.
   - Any folders/subfolders your program creates should be programmatically created. I will immediately delete (and penalize you for submitting) any folder structures that you include with your submission.
6. At the end of your implementation, you should be able to:



   - Compile and run your code on a typical Linux machine(s). Include a `readme` file with clear compilation instructions and any additional software I might have to install, if there are any.
   - Run your server program first.
   - Run one or more clients and register with the server.
   - Create channels and topics. Engage with other users through IRC.
   - Exit the client(s) gracefully.

Instructions

- **Start early!!**. This is a loaded assignment with a fair bit of RFC reading, **especially** for those who didn't quite complete PR01.
- **Take backups of your code often!!**. Practice good version control habits.
- Follow a good coding standard. Use the Google C++ coding standard found here `http://goo.gl/1rC1o`, if you are not already following one.
- First, carefully read **RFC #2810 Internet Relay Chat: Architecture** `https://tools.ietf.org/html/rfc2810` to familiarize yourself with the lingo and to get an overall understanding of how IRC work.
  - You are not expected to implement **`operators`** or **`server-to-server`** communication (*yet!*), but basic `server` and `client` functionality must be implemented. You are allowed to absorb `operator` functionality into your `server`, if it deems necessary for your design.
  - Feel free to have a look at `http://chi.cs.uchicago.edu/chirc/irc.html` as it has a more visual (and bit clearer) presentation of IRC than RFCs. Stay away from any code that you see in that website. Also, your project specifications are not necessarily the specification of this particular website.
- At minimum, fully read all sections relevant to the technical requirements above from **RFC #2812 Internet Relay Chat: Client Protocol** `https://tools.ietf.org/html/rfc2812`.
- Spend adequate amount of time on planning and designing your server management strategy.
- You are strongly encouraged to answer your own design and/or implementation questions based on the the RFCs, rather than defaulting to the instructor immediately. Make sure to properly document any RFC based design choices you had to make on your own that's not explicit in this assignment manual. Be as much descriptive as possible with proper sectional citation(s) from the relevant RFCs (*that way I will know why you did what you did*).

- The due date of this assignment is **Wednesday, March 28, 2018 @ 11:59:59 a.m.**. A dropbox will be opened for submission on Moodle.

Deliverables

A complete solution comprises of:

- A short report of the design and implementation of your system. The report should be **PDF** format. At minimum, your report should include the following sections:
  - Introduction: Your objective and what you hope to gain from assignment.
  - Overall design, specific design choices, and reply codes used.
  - The output of a sample run (including screenshots where applicable).
  - 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. **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 instructions.
  - A `makefile` to automate compilation (except python).

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 others 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 `http://beej.us/guide/bgnet/output/print/bgnet_USLetter.pdf`
- Internet Relay Chat: Client Protocol RFC #2812 `https://tools.ietf.org/html/rfc2812`
- Internet Relay Chat: Architecture RFC #2810 `https://tools.ietf.org/html/rfc2810`
- Internet Relay Chat: Channel Management RFC #2811 `https://tools.ietf.org/html/rfc2811`
- Internet Relay Chat: Server Protocol RFC #2813 `https://tools.ietf.org/html/rfc2813`
- The University of Chicago χ-Project `http://chi.cs.uchicago.edu/chirc/irc.html`