Overview

Your second programming assignment this semester is to implement an Internet Relay Chat (IRC) application using the socket interface. The IRC protocol description is fairly lengthy and spread over multiple RFCs. Given our 2 week window, you are not expected to implement a fully-fledged IRC application, but rather a simplified basic IRC. In essence, what you are working on is a very light-weight version of Discord. Moreover, your P2 will be a direct extension of this project. Specific learning objectives of this assignment are:

a. to learn how to read RFCs and implement protocols to their specifications
b. to further reinforce the concept of “protocol” using hands-on programming; and
c. to gain experience in developing TCP-based multi-threaded applications.

Back Story

Captain Haddock started an international shipping business using the Red Rackham’s treasure. Although business is booming, his lack of a reliable group communication method with his captains is causing all sorts of headaches. “Thundering tyoons!!”. Screamed Haddock. “That’s the 4th missed delivery this week!! Whose orders are these guys following?” Professor Calculus, after overhearing Haddock cursing and kicking around the furniture in frustration, approached him with a potential solution. “Let me build you a relay chat network Haddock. You can chat with any and all 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. You are to write a client–server application to support Calculus’s IRC system.
2. Implement a single central server IRC network. server-to-server communication is outside the scope of P1.
3. The client ↔ server interaction should happen over TCP.
4. Follow the protocol grammar listed under RFC #2812 Internet Relay Chat: Client Protocol https://www.rfc-editor.org/rfc/rfc2812 Sec. 2.3.1. It’s highly recommended to use <regex> for efficient parsing and overall code readability.
5. Support the following Connection Registration messages (RFC #2812 Sec. 3.1): NICK, USER, and QUIT.
6. Support the following Channel Operation messages (RFC #2812 Sec. 3.2): JOIN, PART, TOPIC, and NAMES.
7. PRIVMSG command (RFC #2812 Sec. 3.3.1).
8. TIME command (RFC #2812 Sec. 3.4.6).
9. Closely related to each message above are the numeric replies listed under them. These correspond to message responses (RPL_ prefix) and error replies (ERR_ prefix). Carefully read RFC #2812 Sec. 5.1 and 5.2.
• Clearly document all reply codes you’ve implemented in your report.
• Grader will randomly test numeric replies by triggering the corresponding error condition(s). Thus, it is advised to implement as many numeric replies as possible corresponding to each message.

Note:
The RFC lists several examples under each message (Sec. 3.1 and 3.2) that demonstrates each client message as well as the corresponding server response. You are expected to follow the same format/syntax in your implementations.

Logistics

1. Use configuration files to provide runtime arguments as follows:
   • The client executable receives two runtime arguments through client.conf.

   ```
   client.conf
   SERVER_IP=y.y.y.y
   SERVER_PORT=xxxx
   ```

   • The configuration file for the server executable should have the following format:

   ```
   server.conf
   NICK=<server_nickname>
   PASS=
   PORT=
   SERVERS=
   SOCK_ADDR=
   ```

   The only necessary server runtime argument at this point are NICK and PORT. Rest are in anticipation of P2. For now, simply read and store the unused runtime arguments in variables (for future use).

2. Concurrent multiple clients should be supported; more than one client should be capable of communicating with the server at the same time without any interference.
3. Clients should be able to change their nickname(s) (NICK), register their usernames (USER), and gracefully exit thru QUIT.
4. Clients should be able to join/create channels (JOIN), leave channels (PART), set/view/list/clear topics (TOPIC), and list all other nicknames on the server (NAMES).
5. Clients should be able to communicate with other users either directly or through channels (PRIVMSG).
6. Clients should be able to query the server for the localtime (TIME).
7. The server process is permitted to be forcefully killed if needed.
8. Print all reply codes on all interactions to the standard output.
9. 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 the grader might have to install for the grader in case your MAKEFILE doesn’t work.
   • Run your server program first.
   • Run one or more clients and register with the server.
   • Create channels. Engage with other users through IRC.
   • Exit the client(s) gracefully.
10. Here’s a very basic sample interaction. The client’s hostname and IP are 10.24.186.1/tgamage-1. For the server they are 10.24.186.0/tgamage-0. Assume the server’s nickname is babbage.


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/](https://google.github.io/styleguide/), if you don't already follow one.
- In addition to sections explicitly listed above, it is recommended to read other relevant sections of the RFC #2812 to get a better understanding of the protocol.
- Use of unauthorized/third party packages, except basic I/O, regex, socket, and/or string parsing packages, without the instructor’s explicit permission, will nullify your submission.
- The due date of this assignment is **Monday, March 27, 2023 @ 01:29:59 p.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 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 (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. There will be penalties for those who do this.
  - A short readme file with compilation instructions.
  - A makefile to automate compilation.

Have a look at [http://chi.cs.uchicago.edu/chirc/irc_examples.html](http://chi.cs.uchicago.edu/chirc/irc_examples.html) for few more sample interactions but do not use any code you find there in your implementations.

---

last updated: 03/16/23 @ 2:03pm

3
To create a compressed tarball of the directory `source`, use the following command:
```bash
tar -zcvf siue-id-p1.tgz source/
```
e.g. `tar -zcvf tgamage-p1.tgz P1/`

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 [https://beej.us/guide/bgnet/html/](https://beej.us/guide/bgnet/html/)
- The University of Chicago χ-Project [http://chi.cs.uchicago.edu/chirc/irc.html](http://chi.cs.uchicago.edu/chirc/irc.html)