Overview

Your first 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, over the years, has expanded into multiple RFCs. Given our 2 week window, you are not expected to implement a fully-fledged IRC application, but rather a simplified basic IRC application. Moreover, your PR02 is expected to be an extension of PR01. Specific objectives of this programming assignment 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 previous programming concepts;
d. to learn how to implement protocols to their RFC specifications; and
e. to gain experience in developing both TCP and UDP-based applications.

Back Story

Captain Haddock started an international shipping business using the Red Rackham’s treasure that he discovered recently. While business is booming, Haddock is having all sorts of problems communicating with his captains while they are on a voyage. “Blue blistering barnacles!!”. Screamed Haddock. “These Sea-gherkin Carrier Pigeons are not reliable or efficient. I should have read this message 2 weeks ago”. 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’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. You will need 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 this project.
3. The client ↔ server interaction should happen over TCP.
4. 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.

5. 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 regex support of your implementation language for efficient parsing.

6. Support the following Connection Registration (RFC #2812 Sec. 3.1) messages: USER, NICK, and QUIT.

7. Support the following Channel Operation (RFC #2812 Sec. 3.2) messages: JOIN, PART, and TOPIC.
   - For this first assignment, consider channels as the way clients chat with each other.

8. 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. IP addresses/hostnames and port numbers should not be hard coded.
   - A typical server executable is expected to have the following execution signature: ./server <tcp-listen-port>
   - A typical client executable is expected to have the following execution signature: ./client <server-ip> <server-port>.
   
   *Your may assume that your client(s) know the server's hostname/ip address and the port; port/hostname discovery is outside the scope of this project.*

2. Create a default channel named #time at the server. Use the TOPIC message to query the server’s system time using this channel.
   - The purpose of the #time channel is to verify minimum functionality of your implementation.

3. Test that two clients can chat with each other through a (private) channel.

4. Your client and server should be able to run on two separate end systems. Bare minimum, use the zone server (zone.cs.siue.edu) to spawn two containers – one for a server and one for a client – for minimum testing before submission.

5. All clients should exit gracefully thru QUIT. server process is permitted to be forcefully killed.

6. 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.
   - Have a look at 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.
Print all reply codes on all interactions to the standard output. 

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.
- 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.

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.
- This assignment is fully intended to be implemented with just the socket package provided by preferred programming language API. Use of any other package, except basic I/O, regex, or string parsing packages, without the instructor’s explicit permission, will nullify your submission.
- The due date of this assignment is **Wednesday, September 30, 2020 @ 09:59:59 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 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.
  – 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:
```bash
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. 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
• The University of Chicago χ-Project  [http://chi.cs.uchicago.edu/chirc/irc.html](http://chi.cs.uchicago.edu/chirc/irc.html)