IT 347 Project 1: Socket Programming

Objective:

1. Gain experience with socket programming.
2. Gain experience with client and server programs.
3. Understand what goes into implementing protocols.
4. Gain experience with HTTP and HTML client-server patterns.

Instructions:

You will create client and server applications to talk with each other as well as the clients and servers of other class members. Project 1 is a very simple command driven client and server. Project 2 gets a bit more exciting.

This can be written in any programming language. In fact, since every class member is working off the same protocol, you should be able to interface with a client or server written in a different language without a problem. Recommended languages include Python, C#, C/C++, Ruby, Java, Perl, etc. Pick one you are familiar with (if you are still unsure, go with Python 2.7).

There are many examples with similar code on the web. It is recommended to use these as examples, but none of them implement the entire project. Just like IT 210 or CS 142, you need to understand how all parts of your code work you may lose points if you cannot adequately explain how pieces of your code work.

Specifications

Server

1. Open a listener on TCP port 9020.
2. Implement the server such that (all client commands are case sensitive):

Implemented?	Requirement
☐	Client open receives a response of "Welcome to <yourname>'s Chat room<cr><lf>"
☐	Client request "help<cr><lf>" receives a response of a list of the commands and their syntax.
☐	Client request "test: *words<cr><lf>*" receives a response of "words<cr><lf>".
☐	Client request "name: <chatname><cr><lf>"" receives a response of "OK<cr><lf>".

- ☐ Client request "get<cr><lf>" receives a response of the entire contents of the chat buffer.
- ☐ Client request "push: <stuff><cr><lf>" receives a response of "OK<cr><lf>". The result is that "<chatname>: <stuff>" is added as a new line to the chat buffer.
- ☐ Client request "getrange <startline> <endline><cr><lf>" receives a response of lines <startline> through <endline> from the chat buffer. getrange assumes a 0 based bufffer. Your client should return lines <startline> <endline-1 >
- ☐ Client request "SOME UNRECOGNIZED COMMAND<cr><lf>" receives a response "Error: unrecognized command: SOME UNRECOGNIZED COMMAND <cr><lf>".
- ☐ Client request "adios<cr><lf>" will quit the current connection.

3. Additional Constraints

- The appearance of a a <cr><lf> signifies the end of a request or a response.
- If you extend the protocol as part of your Extra features, you must support the original protocol in a backwards compatible manner.
- Client commands can be issued in any order.
- If a name hasn't been set use "unknown"
- Don't insert extra tabbing/formatting unless it is enabled via a protocol extension.

Client

1. Run the client by typing: <yourclientprogram> < ipaddress> <port>
2. Introduce yourself to the server (name: <name>)
3. use help
4. use test
5. use name
6. use get
7. use getrange
8. use adios
9. All the commands and responses are written out to the screen for checking later.

Grading

1. (10) 1-9 demonstrated with own client.
2. (15) 1-9 demonstrated 2 other student's client/servers both in a different language. Describe in your write-up your experiences interoperating with others' servers and clients. What problems did you have? What did you have to do to overcome incompatibilities between students' implementations?
3. (20) Pass the robust implementation test. prj1_test.rb
4. (20) Project write-up and code package
    - o (15) Code structure and readability
    - o Project description and architecture (how are things structured and why?)

    - Analysis of implementation issues (what were your bugs and how did you find them?).
    - Grammar and readability.
    - Useability of the lab package (ease of access to parts, ie html links to source etc)
5. (15) Fix your client broken by someone else's test case code (ruby test script provided) and include in the write up what was broken, how you fixed it, and what you learned.
6. (5) Add a command that upgrades your protocol to an enhanced protocol with a custom advanced command that interoperates with someone else in the class.

Useful Tips

Socket timeout can be avoided by using the following python code:

```
s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
```

Similar functionality should be available from the operating system in other programming languages as well.