[redacted] Introduction to Computer Networks

Programming Assignment #1
Due Sunday, end of Week 5, by 11:59pm
Submit the source files, Makefile, and README in a .zip file to Canvas.

Objectives:

1. Implement a client-server network application
2. Learn to use the *sockets* API
3. Use the *TCP* protocol
4. Refresh programming skills

The Program:

Design and implement a simple chat system that works for one pair of users, i.e., create two programs: a chat server and a chat client. The final version of your programs must accomplish the following tasks:

1. *chatserve* starts on host A.
2. *chatserve* on host A waits on a port (specified by command-line) for a client request.
3. *chatclient* starts on host B, specifying host A's hostname and port number on the command line.
4. *chatclient* on host B gets the user's "handle" by initial query (a one-word name, up to 10 characters). *chatclient* will display this handle as a prompt on host B, and will prepend it to all messages sent to host A. e.g., "`SteveO> Hi!!`"
5. *chatclient* on host B sends an initial message to *chatserve* on host A : PORTNUM. This causes a connection to be established between Host A and Host B. Host A and host B are now peers, and may alternate sending and receiving messages. Responses from host A should have host A's "handle" prepended.
6. Host A responds to Host B, or closes the connection with the command "\quit"
7. Host B responds to Host A, or closes the connection with the command "\quit"
8. If the connection is not closed, repeat from 6.
9. If the connection is closed, *chatserve* repeats from 2 (until a SIGINT is received).

Requirements:

- *chatserve* must be implemented in Java or Python
- *chatclient* must be implemented in C.
- Of course, your program **must be well-modularized and well-documented**.
- Your programs must run on an OSU *flip* server (for example: flip1.engr.oregonstate.edu). Specify your testing machine in the program documentation.
- Your programs should be able to send messages of up to 500 characters.
- Use the directories in which the programs are running. Don't hard-code any directories, since they might be inaccessible to the graders.
- Be sure to cite any references, and credit any collaborators.
- Provide a README.txt with **detailed** instructions on how to compile, execute, and control your program.

Notes:

- If you are implementing in *C*, read *Beej's Guide*. It has everything you need for this assignment. You will probably learn the most about socket programming by using C.
- However … if you are using *Python*, check http://docs.python.org/release/2.6.5/library/internet.html, and if you are using *Java*, check http://docs.oracle.com/javase/tutorial/networking/sockets/index.html.
- It's OK to hard-code host A's handle.
- It's OK to implement this system so that it requires the two users to take turns sending messages, i.e., when a user sends a message, s/he must wait for a response before sending the next message.
- When debugging, don't use the well-known port numbers, because these will already be in use. I'd suggest using 30020 or 30021, though other students may be using these when you're on the servers.
- If you use additional include-files or make-files, be sure to hand them in with your program. You can zip/tar all of your files together and submit just one archive file if you wish.
- You can test these programs using just one computer. Start the server, then start the client in a new window, and reference the server as *localhost*. You can then switch back and forth between the two windows. (See http://en.wikipedia.org/wiki/Localhost)
- Project #1 will be accepted up to one week late with a penalty of up to 10% per day.

Extras:

- If you implement extra credit features, be sure to describe those features in your program documentation and README.txt
- There are many possibilities for extra credit. Be sure that your program satisfies the requirements first, and then do the extra credit in separate files.
- Extra credit possibilities include (but are not limited to) …
  - Set up your programs so that either host can make first contact.
  - Make it possible for either host to send at any time (while the connection is active) without "taking turns".
  - Make your server multi-threaded.
  - Split the screen to show host B's typing in one panel, and host A's responses in the other panel.
  - Make the characters appear on the receiving host as they are being typed on the sending host (instead of waiting for the entire line to be sent).