Computer Networks Spring 2019 – Lab 1 – Chat Program

Assigned: 3/29/19

Due: 4/5/19, 11:59pm

In this lab, you will create two programs: a chat client and a chat server, each with some basic functionality. You have a choice for what features to add beyond the basic functionality. You can receive extra credit *up to* 10%.

1. Chat Server basic functionality – 25%
    a. Allow user to choose a port on which to listen
        i. In the OMH computer lab, the port should be in the range 43500-43505
    b. Accept incoming connections from client programs
        i. The baselines is to accept only two connections. See 3a below.
        ii. You may follow this process for simplicity:
            1. Start server program, which listens for incoming connections
            2. Start client #1, which connects to server
            3. Server creates socket for client #1, stores in list
                a. Set socket to be *non-blocking*
                b. If Client #1 starts sending messages at this point, behavior is undefined
            4. Start client #2, which connects to server
            5. Server creates socket for client #2, stores in list
                a. Set socket to be *non-blocking*
            6. Server goes on to loop forever, receiving messages from either client, sending to the other one.
    c. Pass chat messages between clients.
        i. If you choose 3b, broadcast messages to all clients.
    d. You may assume messages are under 100 bytes, for simplicity

2. Chat Client basic functionality – 35%
    a. Connect to a server by IP address and port
    b. Accept user input at the command prompt
    c. Send the user's message to the server. For simplicity, you may use the following order
        i. After clients are connected to the server (see above)…
        ii. Client #1 accepts user input while Client #2 waits for a message from the server.
        iii. Client #1 sends message after user enters it. Client #1 waits for response.
        iv. Client #2, after receiving and displaying message from Client #1, accepts user input to send, sends it, and waits for a response.
        v. In this way, Clients #1-2 trade off sending/receiving messages.

3. Additional features
    a. Process server IP, port, etc. using command line arguments instead of user input. 5%
    b. Allow clients to send/receive messages at *any time* in *any order*. 10%
    c. Design the programs to work with arbitrary number of clients. 10%

d. Allow clients the option of specifying a server *hostname* instead of an IP address. 5%
    i. For testing simplicity, you can use `localhost` – I don't expect you to obtain a domain name to test this.
e. Support multiple chat rooms, including a command for a client to join a specific room, and to change rooms. 15%
f. Add a GUI to the client program – 15%
g. Add / commands, e.g., "/leave" to exit the program. 1% per command, up to 10%.
    i. Some other suggested commands: "/shutdown" to close the server; "/pm X" to send a personal message to a single client (X); "/welcome X" set a welcome message (X) the server broadcasts when new clients join.
h. Add one or more server-side bots for clients to interact with. 5% each.
    i. For example, a bot may respond to "!bot tell me a joke" with canned joke responses. The bot can be built into the server functionality, it does not have to be a separate program/process.
i. Make the client/server work with arbitrary length messages, and ensure the entire message is sent and received – 10%

4. README – 5%
    a. Include instructions for running your client and server, what platform you tested it on, and describe all features implemented and how to use them.

5. Note the following:
    a. Programming language choice: up to you.
    b. Platform: I need to be able to test in Linux (e.g., luke) or in Windows 10. If using Python, it should work fine across platforms. You may test on your personal computers, on `luke.cs.spu.edu`, or on computers in OMH (ports 43500-43505).
    c. You may work individually or in pairs for this assignment. (Pairs is especially helpful when testing multiple chat clients.)
    d. Your code must be commented following good programming practices. In addition to appropriate use of comments throughout your code (e.g., within functions), you must:
        i. Include comments at the top of each file explaining the contents of the file,
        ii. Comment all functions, classes, etc. with explanatory headers (e.g., function parameters, returns, effects),
        iii. Failure to comment appropriately will be -15%.

Turn in: your client and server programs and README.