Chat Server
Due: Thursday, March 4 (11:59 PM)
CSC340: Networking and Distributed Processing

Your task is to create a chat application that follows the basic client-server model. That is, there should be two separate programs: a chat server and a chat client. I am not stipulating the exact format of the GUI, which is left up to the developers to design. However, the following functionality *must* be supported:

1. The chat server should be able to support at least 10 simultaneous connections.
2. The chat server should have a default port of 1518.[1]
3. The chat server should support multiple rooms indicated by an alphanumeric string. The default room is the main room labeled `0`.
4. The chat server should require every client connected to provide a name (which does not need to be authenticated - no need to worry about logins).
5. The chat server should broadcast all messages sent from any client C to all clients that are in the same room as C (including sending to C) and include the name of the client in the message.
6. The chat server should recognize when a client leaves or enters a room and transmit a message to all clients (in the room) when such changes happen. (E.g. "The Boss has entered the room.")
7. The chat server should keep a log of all transmissions to/from it. (This can help for debugging and verification of transmissions.) The format of the log is up to the developers but the data should be readable and informative.
8. The chat client should allow a user to connect to a remote server at a specified port and with a user-specified name. (For example, this could be done on a command line or via some menu option).
9. The chat client should support two views (for example, separate windows, separate frames, split screen). The first (room) view should be a real-time display of all chat room messages sent from the chat server to the chat client. The second (input) view should be a means for the user to enter in messages and transmit them to the server. Note, this can be handled with threading so that the conversation is updated continuously and not blocked waiting for the user to enter in a message.
10. The room view should show at least the last 10 lines of conversation. It does not need to be scrollable, though that certainly helps in reading conversations.
11. The input view does not need to transmit messages to the server one character at a time but only once a user hits enter or clicks a send button. Otherwise, you would have to deal with editing of messages.
12. Most importantly, just as there are more than one web server and web browser in the world, there will be multiple chat servers and chat clients. Your chat server and chat client must work seamlessly with the other chat servers and chat clients. That is, your chat client should be able to communicate with any of the other chat servers and vice versa. This will necessitate every group following a specific protocol correctly. To make it easier, the protocol is specified below.

[1]Port 518 is historically the port assigned to a protocol called ntalk.

Submission

As teamwork is an important part of real-world development, you are required to work in teams of approximately 4-6 people on this assignment. Teams can be self-selected, but for those individuals that aren't part of a team (of at least 4 developers), I will form or find a team for you. When you have completed your project, submit all code in a zipped file on Blackboard. In addition, a short README file should be submitted. See below for details.

The day after this assignment is due we will try demoing the applications live. Given that many teams might be remote and hindered by a firewall or two, we might not be able to perform this perfectly, but we'll give it a shot anyway. First, each team will demo their application using their chat server and their chat clients. After each team has shown that, we shall then have cross-server demonstrations. One team will start up their server and then each team will connect one of their clients to that server and commence a conversation. I will also have a basic chat server and client to test the protcol.

README

The README file should include the following:

- The names of everyone on the team. The names should also appear in the commenting for proper credit to be given to the work done.
- Instructions on how to compile and run both the server and client code.
- A breakdown of the work that each member of the team did, what parts were contributed by that individual.
- A description of how the tasks were divided out among the members.

Protocol

All commands are case-sensitive. Here are the requirements for the protocol from the client to the server.

- `ENTER` *`NAME`*: This is sent upon initial connection from the client to the server. The server will register the client in the system with the name `NAME` which is an alphanumeric sequence of up to 16 characters. Note this means that only the characters A-Za-z0-9 are allowed, all others will produce unspecified results and should be deemed a violation of protocol.[2] The server will place the client in room `0` and send an entering message to all clients in that room. The server will also transmit an `ACK` back to the client. (See `ACK ENTER` below).
- `EXIT`: The client exits the chat completely. This should initiate the server sending an exit message to all clients in this client's current room and to remove the client from the server.
- `JOIN` *`ROOM`*: Join room labeled `ROOM` (some string of up to 16 alphanumeric characters). The server will move the client from its current room to room `ROOM`. The initial room when first joining is room `0`. This should initiate the server sending an exit message to all clients in the old room and an entering message to all clients in the new room. The server will also transmit an `ACK` back to the client. (See `ACK JOIN` below).
- `TRANSMIT` *`MESSAGE`*: Transmit the message to all clients in the same room as this client. (See `NEWMESSAGE` below.) Message can be any sequence of up to 1024 characters and up to a newline. For example, `TRANSMIT Hello, this is a message to send.`

Here are the requirements for the protocol from the server to the clients.

[2]For simplicity, we will not block duplicate names although this can lead to some serious confusion.

- `ACK JOIN` *`ROOM`*: This is an acknowledgement sent back to the client that the requested room `ROOM` was joined. It is used mainly to help the client ensure that the request was granted by the server.
- `ACK ENTER` *`NAME`*: This is an acknowledgement sent back to the client that the client has been registered with the given name.
- `NEWMESSAGE` *`NAME MESSAGE`*: This is sent to a client to indicate that a new message `MESSAGE` has arrived from the given client `NAME`. The server transmits this to every client in the room of the sending client with the transmitted message. (See `TRANSMIT` above.)
- `ENTERING` *`NAME`*: This is sent to all clients in a room when client `NAME` has entered. The client entering should also receive this message.
- `EXITING` *`NAME`*: This is sent to all clients in a room when client `NAME` has exited. The client exiting also receives this message.

Team Evaluation

In addition, after the deadline, you will evaluate the performance of all of your team members (include self). Completing the evaluation will be 20 points but the specific evaluations will not affect your teammates' or your scores. We will use it for the following purposes:

- To help determine if team membership needs to change before the main final project
- To help give feedback to your teammates for future growth
- To help understand the general criteria used in measuring team performance for the final project when it will affect your grade

Scoring

I will use the following rubric in grading this assignment:

- **(10 points)**: Chat server supports multiple connections to specified port including default port 1518.
- **(5 points)**: Chat server properly recognizes client names.
- **(5 points)**: Chat server broadcasts entering and leaving messages.
- **(5 points)**: Chat server receives client messages.
- **(5 points)**: Chat server broadcasts client messages.
- **(5 points)**: Chat server distinguishes between rooms.
- **(5 points)**: Chat server supports a good (debugging) log.
- **(5 points)**: Chat client successfully connects to chat server.
- **(5 points)**: Chat client supports specifying communication host and (default) port names.
- **(5 points)**: Chat client supports user-specified name.
- **(5 points)**: Chat client supports changing rooms.
- **(5 points)**: Chat client indicates messages received from server in some form.
- **(5 points)**: Chat client properly transmits messages to server in some form.

- **(5 points)**: Chat client GUI has adequate room view (for incoming conversation) with properly labeled real-time messages indicating message and sender name.
- **(5 points)**: Chat client GUI has adequate input view (for transmitting conversation to server).
- **(5 points)**: Chat server supports other application chat clients by properly following specified protocol.
- **(5 points)**: Chat client supports other applicaton chat servers by properly following specified protcol.
- **(5 points)**: Chat client and server are robust to communication errors (failed or dropped connections, recognizing ACKs, etc.)
- **(10 points)**: Style: code is well documented and easy to follow.
- **(10 points)**: Documentation: README text is present and provides the required content.
- **(10 points)**: Preparedness: Code is ready to demonstrate in class on Demo day (Friday, March 5).