Assignment 1
SOFE4790 Distributed Systems
Fall 2016

A messaging distributed application

This assignment is meant to help you become familiar and comfortable with interprocess communication over the network using sockets and threads. You can use Java since you already have experience with it from the lab. However, you are free to use any programming language of your preference.

You will be implementing a simple messaging distributed application. Users will connect to a server to join, and afterwards they will be able to send and receive one-to-one messages from each other. To keep the implementation simple, the server does the message forwarding on behalf of the users. If user X wants to send a message m to user Y, X sends m to the server specifying the target to be Y. The server then forwards m to Y specifying the message came from X.

You will need to design a bare-bones message *protocol* for your application. For example, the initial message from a joining user could have its username information. In a message exchanged between users, the target user must be specified in some format known to both server and user. The server can determine the user who sent the message by user connection it came in through.

Users should be identified by usernames, for simplicity, a joining user could supply their own username (assuming no duplicates). The server should also maintain an internal list of user id's for indexing the connections to the users. The server should maintain an array of user connections (sockets) indexed by their id's, as well as a mapping of usernames to internal user id's, for look-ups when forwarding messages.

Server design. The server has one main thread that listens for incoming connections. Every time an incoming connection from a user is made, a new socket (user connection) is created and a new thread is created and started for that user connection. The server object and the socket just created are passed to this user thread. The server also adds this new connection thread to its array of user threads. The user id (could simply use incrementing count of joining users) and the username pair is added to the map maintained by the server. The username could be included in the initial connection request message from the user.

The main server class should have a method, called for example *forward(...)* that the user threads would call when they receive messages on their sockets to be forwarded to other users. In this method, the target username is looked up and mapped to its user id, then the corresponding user thread is called and the message passed to it. Therefore the user thread object should have a method, say *send(...)*, that can be called to send messages through its socket.

Every user thread is responsible for receiving messages from the corresponding user. The receive on the socket is blocking, but this won't affect the operations of the main server thread and other user threads. A user thread object should have at least instance variables for socket, user id, name, etc.

To keep it simple, you do not need to worry about users leaving and any clean-up of user information in the server class.

Client/user design. Each user has a main thread that initiates a connection request to the server. In the connection request, the user name can be included. After the connection has been made, a new thread needs to be created, because each user will need two threads (one of which can be the main thread): One for reading messages from `stdin` typed in by user, and sending these messages through the socket. The other thread will be responsible for waiting for and receiving messages coming in from the socket (i.e., from the server), and displaying them on screen `stdout`.

Submission: You should submit your programs and a `readme` file with instructions on how to run your application. You should include screenshots of your application working.