**Lab 6 – Creating a Chat Server**

In this lab you will learn how to write a multi-threaded TCP server, which will allow multiple clients to connect and chat with each other.

**Clients**

Normally a chat client is a single program that has one area for typing and sending messages and another area for displaying the users’ chatter.

The intent of this project is to focus on the server, so instead of writing our own clients we will just use telnet.

To support having different windows for typing and for reading the chatter, clients will either function as a sender or an observer. Senders can send messages and commands, but will not receive any messages from the server. Observers will receive notifications and chatter, but will not send any messages or commands unless they wish to disconnect.

Clients will establish their role by sending either an‘s’ or an ‘o’ when they first connect.

**Senders**

Each sender will have a nickname. At first the sender’s nickname will be **<ip address>:<port>**. The sender can change nicknames by sending the message **/nick <nickname>**.

The sender can quit by sending an empty line.

Any line the sender sends that is not to change nickname or to quit will be sent to all the observers with

**<nickname>:**  prepended to the line.

**Client Sockets on the Server**

When a client first connects to the server, its role has not yet been established. We can think of this as a third role named undecided. Thus each client socket will be in one of 3 states: undecided, sender, or observer.

The server should send a message to an undecided socket telling the user that he or she needs to send an ‘s’ or an ‘o’. The server will then **recv()**, waiting for a response from the client.

If the socket receives a message other than ‘s’ or ‘o’, it should send another message asking for a proper response. After 3 incorrect messages, close the socket.

Once an ‘s’ or an ‘o’ has been received, pass the socket and any other important data to a function which will handle the socket for the rest of its lifetime.

**Multi-threaded Design**

In order to support multiple clients gracefully, each client’s socket will be handled by a different thread. Each thread should be created immediately after the TCP connection is made.

One additional thread will handle pushing messages to all the observers.

Let’s discuss more about the design in class.

**Running the Chat Server**

In command line type: python3 chatServer.py <IPaddr> <port#>

**Running a Chat Client**

In command line type: telnet <IPaddrOfHostRunningTheServer> <port#OfServer>