CSE4226: Network Programming Lab

Assignment #2

Developing a multi-threaded server-client application using TCP Sockets

Objective: The aim of this assignment is to create a simple multi-threaded server-client application where multiple users can exchange message or text with each other with some additional features.

Detailed Features:

i. **User Registration and Login:** To join the system, a user requires creating an account with a username and password. Once account creation is done, the user can log-in or log-out anytime.

ii. **Online User Lists:** After logging on, a user can have the online user list.

iii. **Friend Request:** A user can send friend request to other users. The request can be accepted or rejected. A user can have his/her friend list.

iv. **Unicast:** A user can send message to any other online user

v. **Multicast:** A user can send message to a group of online users (for this feature send the message as **message@first-user:second-user:third-user** and the message will be delivered to the users separated by ":")

vi. **Broadcast:** A user can send message to all the online users

vii. **Create Chat Room:** A user can also create a chat room, see the list of existing chat rooms, join and leave it

viii. **Chat History:** Whenever a user logs in, he/she should get the unread messages and see his or her chat history.

Evaluation Phase – I

Date of submission:

Section	Date
A1	**June 05, 2018**
A2	**May 29, 2018**
B1	**June 06, 2018**
B2	**May 30, 2018**

Required Features: i to vi

Instructions:

- Using database is not required for Evaluation Phase – I. For storing username and password of a user use simple data structures like Array/Array list/Vector/Map.
- Do not include any Graphical User Interface (GUI)
- For server and client side programming use similar architecture as illustrated in the class. **Server and Client program must be in two different projects.** Implementation guidelines are given later in this document.
- Use standard coding convention and documentation of Java.
- **Do not copy from others (Neither code nor Report). It will lead to cancelation of the assignment and inflict severe punishment to both source and destination.**

Report Format and Guidelines:

Format:

Report must be written in LATEX and you need to submit in PDF format (you have already used LATEX for writing AI lab report)

Report Structure:

CSE4226: Network Programming Lab

Assignment #2: Developing a multi-threaded server-client application

Evaluation Phase – I Report

Student ID:

Student Name:

Lab Group:

Date of Submission:

Implementation summary:

Features	Status
(i) User Register and Login	Implemented/Not implemented/Partially implemented

Implementation Challenges:

(Referring each of the features describe in your own words the challenges you faced implementing that feature and how you overcome those with example)

Interaction Diagram:

(Show the interaction between server and client and illustrate the work flow using a diagram)

Limitation and Future Scope of Improvements:

(State the limitations of your assignment and suggest some future improvements to make the application more functional and efficient)

Discussion:

(State your own understandings, Findings and conclusion after completing this assignment)

References:

Appendix

(Source code of your assignment)

Implementation Guidelines:

Visit following links for implementation guidelines:

For server side programming:

https://www.geeksforgeeks.org/multi-threaded-chat-application-set-1/

For client side programming:

https://www.geeksforgeeks.org/multi-threaded-chat-application-set-2/

Followings are the guidelines extracted from above links but slightly modified according to our requirements. So read those carefully. If you face any difficulties feel free to contact me.

Server Side Programming

Server class:

The main server implementation is easy and similar to server of Example 2.1. The following points will help you to implement the Server class:

- The server runs an infinite loop to keep accepting incoming requests.
- When a new request comes, user checks its username and password. If its username and password matches server assigns a new thread to handle the communication

part. User can create a new account if it does not have any and server stores its username and password in a file or database.

- The sever also stores the username and the thread object corresponding to the user request into a vector/map/arraylist to keep a track of connected devices. The helper class uses this vector/map/arraylist to find the name of recipient as well as its thread object to which message is to be delivered. As this vector holds the client socket, handler class can use it to extract stream objects and successfully deliver messages to specific clients.

ClientHandler class:
An object of this class will be instantiated each time a request comes.

- This class extends Thread so that its objects assume all properties of Threads.
- The constructor of this class takes one parameter, which can uniquely identify any incoming request, i.e. **Socket**. Whenever we receive any request of client, the server creates a new thread object of this class and invokes start() method on it.
- Inside the **run()** method of this class, it extracts the InputStream and OutputStream objects from the Socket object, read the client sentence from input stream object and do the followings:

  - ✓ Whenever the handler receives any string it uses Stringtokenizer to extract different tokens from the string. For this purpose '**:**' is used as the delimiter. (According to the defined protocol. See client side programming)
  - ✓ If client's string contains **MSG:B:Hello,** it extracts the 1st token as the command that is sending message, 2nd token as the recipient and 3rd token as the client's message. It then searches for the name of recipient in the connected clients list, stored as a vector/map/arraylist in the server. If it finds the recipient's name in the clients list, it forwards the message on its output stream with the name of the sender prefixed to the message. Similarly other functions are performed depending on the command.

Client Side Programming

Till now all examples in socket programming assume that client first sends some information and then server or other clients responds to that information. In real world, this might not be the case. It is not required to send someone a message in order to be able to receive one. A client should readily receive a message whenever it is delivered to it i.e sending and receiving must be **implemented as separate activities rather than sequential**. There is a very simple solution which uses threads to achieve this functionality. In the client side implementation we will be creating two threads:

ClientWriter Class:

This thread will be used for sending the message to other clients. It takes input the message to send and the recipient to deliver to. Define a suitable protocol for client to communicate with server. For example, suppose client A wants to say "Hello" to client B, then you can

define the format **MSG:B:Hello**, where B is the name of the recipient. It then writes the message on its output stream which is connected to the handler for this client. The handler breaks the message and recipient part and delivers to particular recipient Similarly, message format can be defined for other activities. For example, if client wants to see the online user list you can use the format **Task:show user list**.

ClientReceiver Class:

A similar approach is taken for creating a thread for receiving the messages. When any client tries to write on this clients input stream, **ClientReceiver** reads that message using readLine() method.

Client Class: The remaining steps of client side programming include establishing a Socket Connection and Communicating with the server. Communication occurs with the help of the ClientReceiver and ClientWriter threads. Separate threads for reading and writing ensure simultaneous sending and receiving of messages.