**SUMMARY**

**OVERVIEW**

The aim of the project is to simulate a message posting system using sockets. The server will maintain the list of clients and the messages sent to/by them. The server will maintain messages posted by clients, which clients can retrieve and view. In communication, client will accept a machine name and port number to connect to the server, prompt for and send the user’s name, present a menu with 7 choices to user, interact with the server to support the menu choices and ask the user for the next choice or exit. The server allows a max of 100 clients/users and stores the messages of the users until they retrieve their emails/messages.

**IMPLEMENTATION:**

We used Arraylists for known and logged in users and hashmap to store the user names and their respective messages.

There is one class in this program, “SocketClient”. When executing this program, Client will connect to server through “listenSocket”, by giving IP address and port number .user will input a name, then method “buildMenu” will be called to display menu, user will input one choice. Each choice will be pointed to the corresponding if() case, meanwhile, the choice will also be sent to server and get reply from server, then display to user.

Once a client is connected to the server, it creates a thread for it. The server takes in the name of the client and creates a hashmap entry for the client, adds him to the logged in and known users list if he a new user.If the user tries to log in with a name that is already logged in, then it throws an error saying “ The user is already logged in. Try with another name” .

The server takes the input from the client to perform the chosen operation and does the required operation.

If a user sends a message to an unknown user, the server then creates a hashmap entry for the client and stores the message. Once the un-known user logs in, it can see the messages sent to it.

When a user requests for its messages, the server displays all its messages and deletes the messages once they are read.

We used a semaphore to provide mutual exclusion to the known users, logged in users and the hashmap so as to avoid conflict when more than one thread is trying to make changes to it.

**PERSONAL EXPERIENCE:**

WEI SONG:

Difficult:

1. How to create Client

I reviewed the slides of sockets again, and read “networking and threads” in HeadFirst Java, got to know basic principle and theory about sockets, then I tried to understand and execute samples given by professor. Based on the program, I started to create my own Client, and add functions to it.

1. Multiple users trying to connect using same name

While running Client program and Server program together, I found that when multiple users trying to be connected to server using the same name, server will crash, and didn’t return the expected result. After discussion with Kavya, I added an if(line!=null) condition before letting user making choices, and then add another case that when server doesn’t return anything, the client will exit. Thus I solved this problem.

1. How to show names and msgs

When Server sending multiple user i.e. known user to Client, it was sent by a String, I didn’t know how to print out user names one by one. I searched information online, and learnt how to use String.split to split contents in one string by some certain symbols.

What I learned

Through this project I learnt how to send and receive data over a network. Although I wrote Client part of this project, by discussion with my partner, I also learnt how to create server and server threads.

**PERSONAL EXPERIENCE:**

*ARAVAPALLI KAVYA*

I worked on the server part of the project.

As I was new to socket programming, I learnt the underlying concepts of socket communication from the example professor gave us. I tried exchanging a couple of messages more between the client and the server so as to understand it better.Then I started working on the methods to send, store and retrieve messages for each client.

The major issue I had was, choosing a proper data structure to store the name of each client and their respective messages. Initially I started with a 2-D array, then went on to an arrayList of array lists in which each arraylist stores the name of the user and then its messages, but was not able to retrieve the messages efficiently. Finally I implemented the mailbox using hashmaps (client name as the key and the messages as array list) as it was easy to search using keys.

One interesting programming mistake I did was to put a “ ; ” after the for statement in retrivemsgs method, which lead the for loop to ignore all its body and just increment until the limit and exit. It took me half a day to spot the mistake.

Another difficult aspect of this project was to correctly map out all the possible interaction between the client and the server.