NOTTINGHAM TRENT UNIVERSITY

College of Arts and Science

School of Science and Technology

COMP20081: Systems Software

Systems Software Report

By

N0634342: Taran Basi

N0631096: Daniel Jones

Java Programming Assignment

Table of Contents

Introduction

As part of our System Software coursework we intend to create a 'Music Social Network System' with the use of the Java programming language. With the use of NetBeans, we will make the system graphical allowing the user to interact with the system easily and we will also implement a centralised server allowing the system to have a Spotify like service to it. The system will contain similarities both with Facebook and Spotify. This means that when the user registers and signs into the system, they will be able to send friends requests, chat to their friends, post and view other posts as well as listen to songs that their friends may have imported into the system. For the system to meet these requirements we will have to include two serves and a client. One server would act as a centralised server overseeing tasks such as friend requests as well as various other features and the other server will be dealing with the chat system.

Requirements

We have created a list of minimum requirements for the system so that it to be functioning appropriately:

- User able to create new user
- User able to log into the system
- User able to upload music file upon creation of user
- User able to send a message to a selected person
- User able to read message that has been sent to you from another user
- Able to Send a friend request to different users of the system
- User can accept or refuse a friend request
- All users can see who their current friends are
- Users can post into the system
- Able to view posts posted by other users
- Users able play songs that have been uploaded by users upon creation of new user

Developing the system

Graphical User Interface

Although prior to the beginning of developing this system we had experience using NetBeans through what we had done in the labs, we did not have much experience when it came to the graphical side of the software. As this was the case when we first began to implement the graphical user interface, it was difficult, however we very quickly understood and came to terms with NetBeans and how it deals with adding components and adding the code so that the components can interact with one another. We then came up with a simple yet stylish design that will allow the users to interact with the system with ease. We began with creating a login page with the options of clicking the 'register' button in order for the user to be able create a new user. From there, we created the registration page which contains all the fields the user must fill out so that they successfully create a user. Both the register and login pages contain validation so that the user information is correct. Once the user is logged in they will be able to navigate through the main page of the system where they will be able to perform any function they wish. Such functions include; add a friend, post, listen to music etc. Finally, we designed the chat page, where the user is able to view who they are chatting to and the messages that have been sent back and forth between them and the recipient. Once we understood the basics of graphical user interface on NetBeans what followed became relatively simple.

Multiple Socket Server

The multi socket server allows for the program to perform multiple tasks simultaneously with the use of threads. This is called a multi-threaded program. A socket allows for a bidirectional communication between the server and one or multiple clients. The server contains a specific port with which the socket is associated with and the clients who are also associated with this socket port can then communicate with the server. So, for our system we begin coding our 'MultipleSocketServer.java' file by creating a new socket and initialising it so its name is equal to 's' and also stating the port number, in this case '19999'. We then connect the server socket to the port so that it will be ready to communicate with the client.

A Multi-threaded server allows for the server to respond to a request from one or more clients. It does this by creating a new thread for each request it receives from one of the clients. This allows the system to be a lot more functional when dealing with multiple users, who are the clients. So, in our multiple socket server, after we set up the port and initialise a server we then create a thread to deal with these requests from the client.

```
public class MultipleSocketServer implements Runnable{

    private Socket csocket;
    static private ArrayList<String> currentLoginList = new ArrayList<String>();
    private static AudioStream audioStream;
   // private static Player player;


    MultipleSocketServer(Socket s) {
        this.csocket = s;
    }

    public static void main(String args[]) {
        int port = 19999;


            try {
                ServerSocket socket1 = new ServerSocket(port);
                System.out.println("Socket connected");

                while (true){
                    Socket socket = socket1.accept();

                    Runnable runnable = new MultipleSocketServer(socket);
                    Thread thread = new Thread(runnable);
                    thread.start();

                    System.out.println("thread started");
                }
            }
            catch (Exception e) { }

    }
```

The rest of the 'MultipleSocketServer.java' file deals with any functionality of the system and also allows for packets to be sent to the client as well as to be received from the client. Packets are used to send pieces of data across a system. In this system that we have created, you will see that we send packets from one client to another or between client and server for various reasons. For example, in this function 'logout' we receive a packet stating that a user has logged out and so we then split the packet so we have the correct information to then remove the username of the user

who has just logged out from the 'currentLoginList'. We then send a message back to client stating that the user has logged out.

```
private void logout(String packet) {

    String[] packetArray = packet.split("~");
    String compare;
    for (int i = 0; i < currentLoginList.size(); i++) {
        compare = currentLoginList.get(i);
        if (packetArray[1].equals(compare)) {
            currentLoginList.remove(i);
        }
    }

    sendToClient("loggedOut");
}
```

Client

The client starts off by connecting to the same port as the 'MultiSocketServer.java' file allowing the client and the server to be able to communicate with one another. Also within the client we have some code which is used for chatting amongst users so that they can chat with one another. This includes setting up a new socket so that the users can send messages as well as receive messages. This is connected to the server as the sender will send a message, the message will then go to the server and from there, and the recipient will receive the message. The client can also open a connection with the chat server to be able to send and receive messages with other users.

```
try {

    if (strArray[0].equals("chat")) {

        port = 19998;

        System.out.println("Received chat");


        Socket cServer = new Socket(host, port);

        DataInputStream din = new DataInputStream(cServer.getInputStream());

        DataOutputStream dos = new DataOutputStream(cServer.getOutputStream());

        String line = null;




           dos.writeUTF(packetStr);
           dos.flush();


           message = din.readUTF();
           System.out.println("Message:" + message);


    } else {
```

Chat Client (nested class within client)

The chat client is used to send and receive messages between clients directly. It begins by calling the client function. Once the function has been called the client setups a new chat client, this is a nested class (we used a nested class as we would need to access it as part of the client class). The chat client then begins to setup a new client listener which will listen for messages. When the send button has been clicked the chat client sends the message to the chat server who then sends it to the server handler to deal with input and distribute output to other intended clients.

For example, client A will open a chat window with client B, the client file will open the socket with the chat server rather than the main server, it does this by using a separate port number that the chat server is listening on, this will then establish an open connection that will allow the client to send messages to the server, which will distribute these messages to the appropriate users.

Chat Server

The chat server is used to establish a connection between a client and the server. Once the connection has been established, the chat server creates a new instance of a user and adds it to the user array list. The chat server then creates new instance of the server handler, passing the socket, and the user array list as parameters. The chat server then starts a new thread to run this server handlers so that the server can continue to listen for incoming connections from other clients.

Server handler

The server handler receives the sockets that have been sent by the client server and also receives the user array list. It then goes on to creating an input and output DataStream which allows for the server handler to read and write messages respectively; that are being sent by the client. The server handler reads the message from the input stream and then using the user array list, writes the message to the intended recipient.

Conclusion

Having coded the system effectively and thought about each process logically we chose to implement a TCP/IP network connection. By using a TCP/IP connection it allows for an error-checked and ordered delivery of data. We have used an open connection to each of the clients on the chat server to allow for easy continuous messaging back and forth.

The project has been coded efficiently and effectively allowing for the user to navigate through the system with ease. After testing, we have resolved any bug fixes that have been found, again ensuring that the intended user has a positive experience when using the system. Due to time constraints we could have implemented some extra features to help make the system more appealing and more enjoyable for the user, however we have implemented all our fundamental features.