Project 1: Client-Server Chat Program

Computer Science 354 – 3 August 2022

Group Members

Bernhardt Steyn – 21740178

Matthew Oosthuyse – 21784507

Project Description

This is a chat program based on a client-server model. Solutions to practical computer networks and implementing client-server applications are handled in this program. This program allows users to connect to a server and parse text messages to an open chat room or directly to another user via the whisper function.

Features Included

Server

- Multiple clients can connect to a single server.
- The server is multi-threaded so that it can handle requests from multiple clients.
- Server operations such as connect requests and disconnect requests are printed out by the server.
- The server handle's connections/disconnections without disruption of other services.
- Clients have unique nicknames; duplicates are resolved before allowing a client to be connected.
- All clients are informed of changes in the list of connected users.

Client

- A GUI is implemented for the client.
- Clients can choose a nickname.
- A list of online users is be displayed inside the GUI.
- Connection/disconnection actions of users is displayed.
- Messages from the originating user and other users is displayed.
- Whisper (private) messages between clients is implemented.
- Can receive messages/actions while typing a message.
- Clients can disconnect without disrupting the server.

Features Not Included

The solution has all the requirements implemented.

Extra Features

- A graphical user interface for the server. This is used for the logging and debugging of the server.
- When a user types “\list” it will give them a list of the commands that they can use such as the whisper and printing the online users to their chat terminal using “/w” and “\users” respectively.
- If a user has been connected to the server and I no longer online. If an online user attempts to send that user a whisper while that user is offline during the iteration of the server, the whisper will be place in a pending state and the message will send to the user if they connect to the server again.

Algorithms and Data Structures

The server is multithreaded by creating a ServerThread object each time a user tries to connect to the server. The server thread is responsible for managing the specific user who is then added to a UserList object. The UserList object manages messages between the users, and it is synchronised by using the synchronizedList method from the Collections library. The UserList is responsible for checking whether a username is valid. It also manages sending messages between users including broadcast messages. For the messages that we send they all consist of a Packet object as these allow us to use keywords which are Enums to specify what the contents of our message will be. This allows for an easier management of incoming messages as the server can easily distinguish between whispers and broadcasts. It also allows us to manage connections and disconnections as unique events so when the user receives the connection Enum it knows that it will also receive an arraylist of connected users. This works similarly for disconnections. The Packet object consists of different constructors that allow packets that consist of diverse types of variables to be sent easily using the same object. These Packets are distinguishable by the keyword that they all contain.

Experiment 1

Observation

When a user connects or disconnects, the user list is update.

Question

Does the user list for all users update when a user connects/disconnects?

Hypothesis

When a user connects/disconnects, the user list is updated and sent to all online users.

Experiment

I am doing this experiment to test my hypothesis. I am going to connect two users with a third user waiting to connect. I will then connect the third user and observe the change in the user list. Thereafter I will disconnect one of the previously connected users and observe what happens to the other users. This process is demonstrated in the screenshots below.

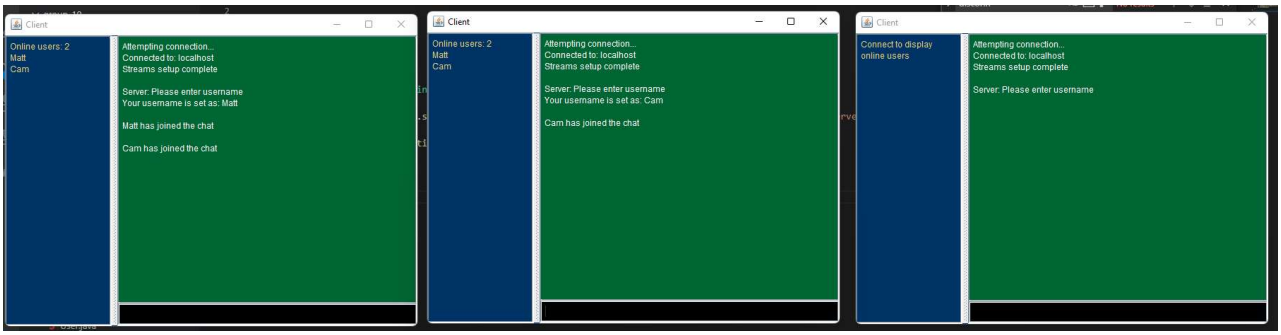

Figure 1 - Two connected clients with one client waiting

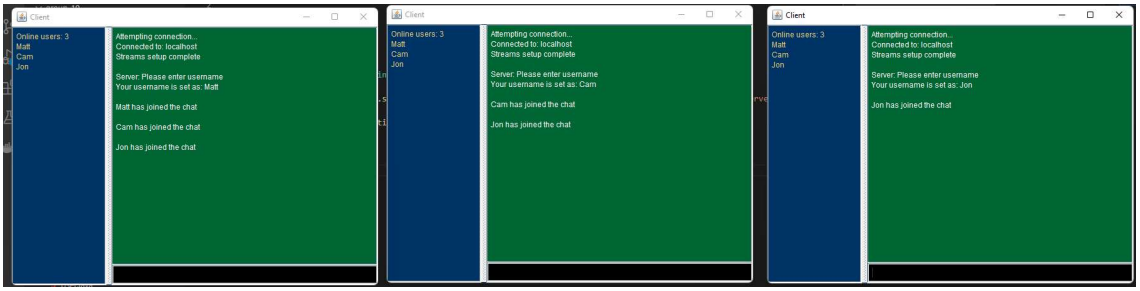

Figure 2 - Third client connecting and user list updating

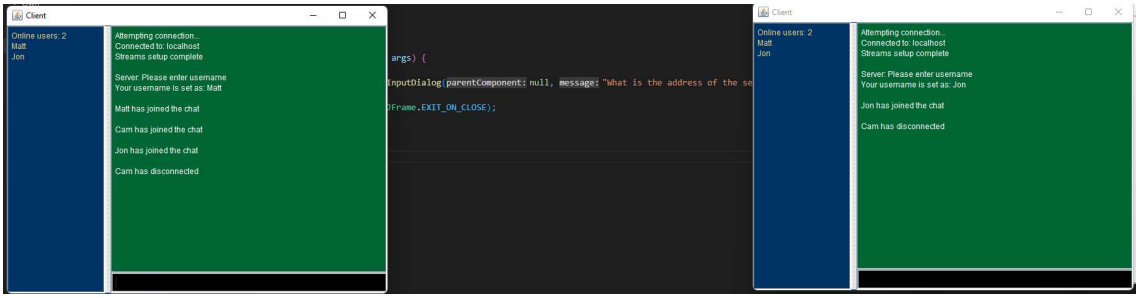

Figure 3 - Disconnecting a previously connected user

Results

From the experiment, it is observed that when the third user connected, the user list on the previously connected users was updated and that the new user received a list of all other online users. Thereafter, when a previously connected user disconnected, the user list on the remaining users was updated and received a message stating that that user has disconnected.

Conclusion

The user list for all users' updates when a user connects or disconnect.

Experiment 2

Observation

A client is stable after server termination and the server is stable after client termination.

Question

Is the client stable when the server is terminated, and the client is still connected? Is the server stable after the client is terminated and the server is still connected?

Hypothesis

When the server is terminated and a client is connected, all ports will be closed, and the client will be stable. When the client is terminated and the server is still running, all the ports for that client will be closed, and the server will be stable.

Experiment

First, the client will be tested with the server being terminated. The client will be connected to the server and have a username setup. Thereafter the server will be terminated, and the client will be observer. This process is shown in the screenshots below.

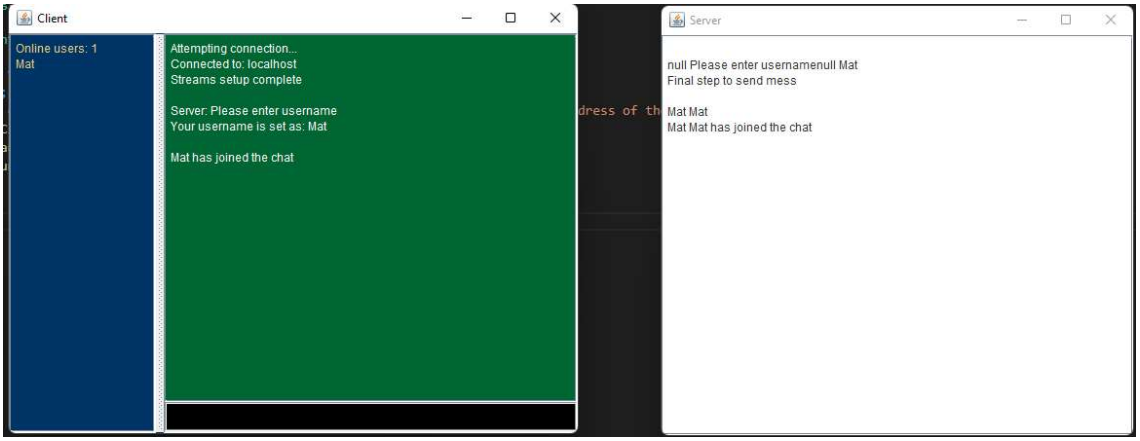

Figure 4 - Client and server connected

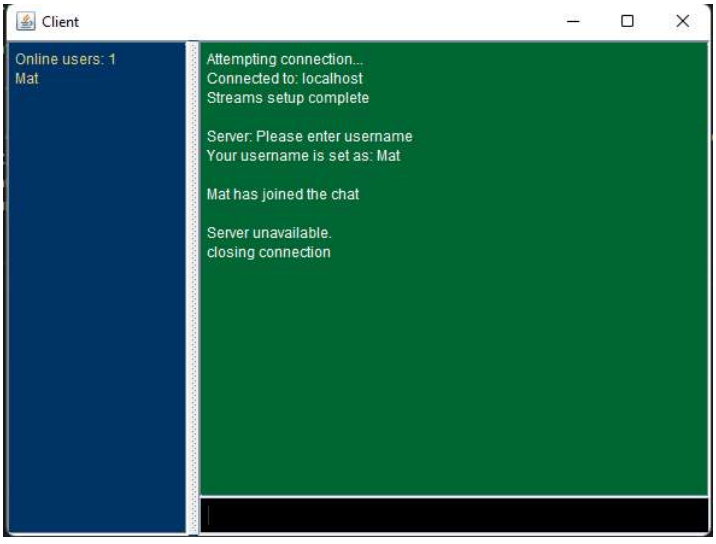

Figure 5 - Client state after server termination

Secondly, the server will be tested with the client being terminated. The client will be connected to the server and have a username setup. Thereafter the client will be terminated, and the server will be observer. This process is shown in the screenshots below.

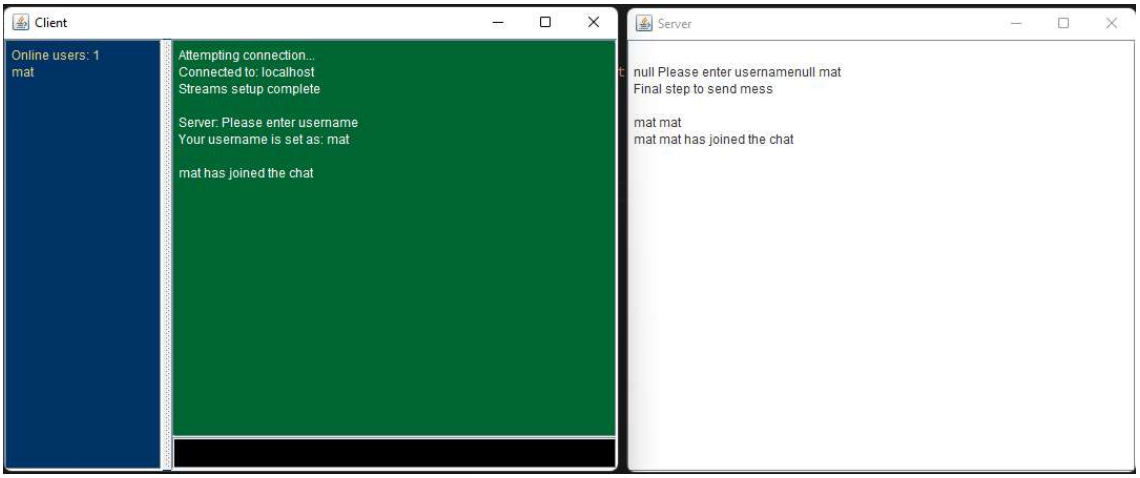

Figure 6 - Client and server connected

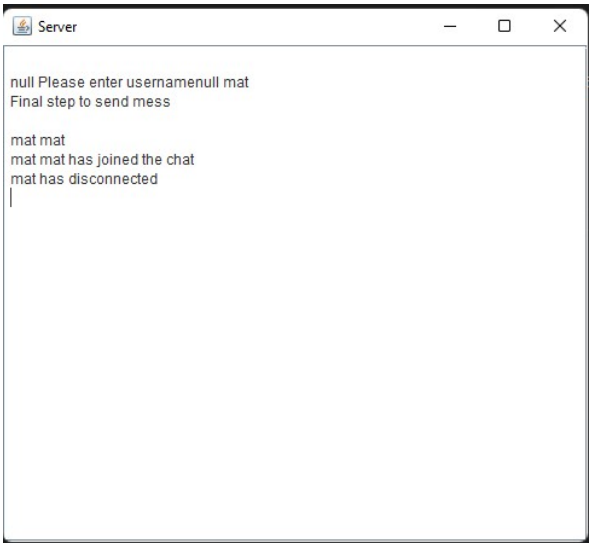

Figure 7 - Server state after client termination

Results

The first part it is observed that when the server is terminated, the client loses the connection and thereafter the ports are closed, and the client remains stable in a disconnected state. The second part, it is observer that when the client is terminated, the server disconnects the client and closes all ports related to the client. The server then remains in a stable state waiting for a client to connect.

Conclusion

From the results, it shows that when the server is terminated while a client is connected, all ports are closed, and the client is stable. It shows that for the second

part that when a client is terminated and the server is still running, all the ports for the client are closed and the server is in a stable state waiting for a client to connect.

Experiment 3

Observation

When a user attempts to whisper to a non-existing user, it will not send the message and will notify the user that the user they are trying to whisper to does not exist.

Question

Is the user able to send a whisper to a non-existing user?

Hypothesis

When the user attempts to send a whisper to a non-existing user, they will be notified that the user they are trying to whisper to does not exist.

Experiment

To test whispering to a non-existing user, a user will be connected and will attempt to send a whisper to a user called “random”. Thereafter the state of the client will be observed. These steps are shown in the screenshots below.

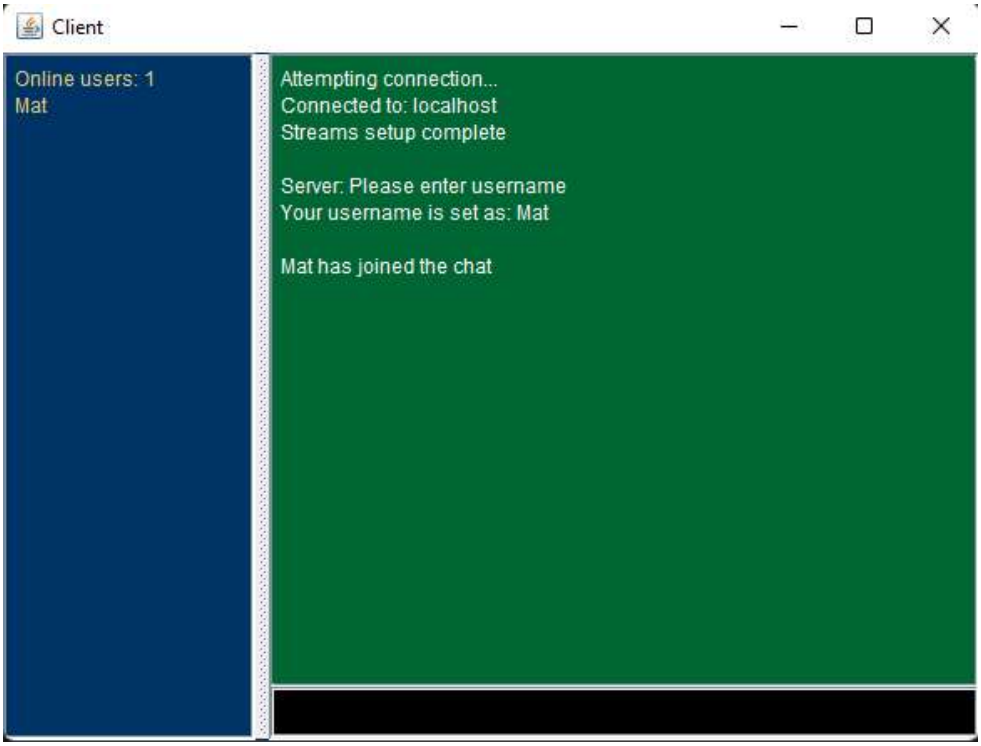

Figure 8 - A connected user

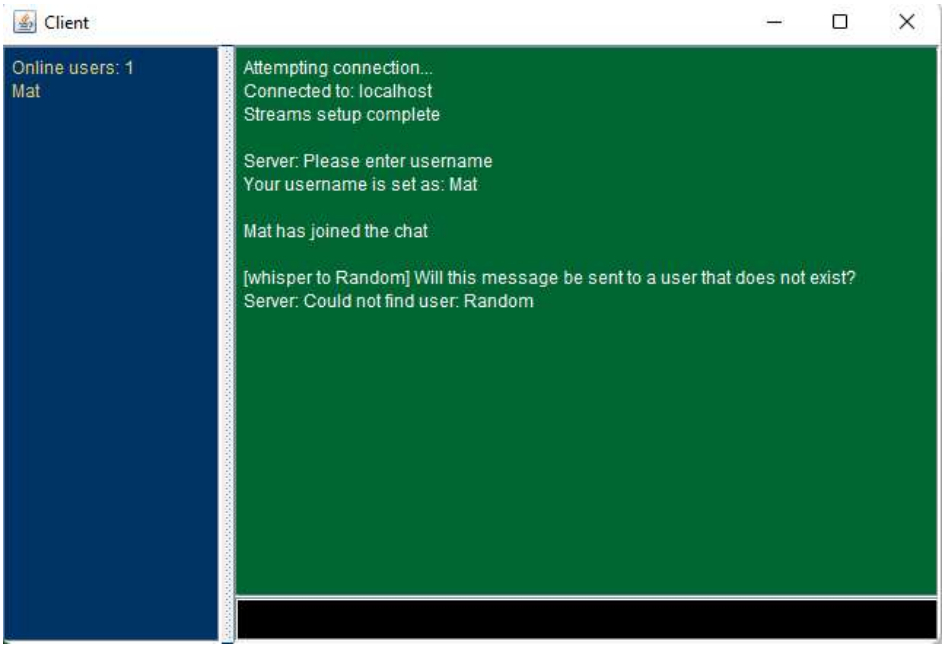

Figure 9 - User attempting to send a non-existing user a whisper

Result

From the experiment, it is observed that when a user attempts to send a non-existing user a whisper, the server returns the message stating that it could not find the "Random" user.

Conclusion

A user can not send a whisper to a non-existing user. When the user attempts to send a whisper to a non-existing user, they are notified that the user they are trying to whisper to does not exist.

How to run the Program

First, we need to compile all the java classes.
make

Once all the classes have been compiled, we need to start the server.

How to start the Server

While in the root directory of the program (group_10), run the command:

```
java Own.Server.ServerTest
```

How to start a client

While in the root directory of the program (group_10), run the command:

```
java Own.Client.ClientTest
```