Group 14 Project 1: Client-Server Chat Program

Dylan Callaghan: 21831599@sun.ac.za
Stephen Cochrane: 21748209@sun.ac.za

August 14, 2020

1 Introduction

The goal of the project was to create a simple chat service, with multiple client GUI's that connects to a centralized server. The chat service required clients to give a unique nickname on startup, after which they would be directed to a global chat on which they could broadcast messages to all users at once. Clients could also message privately in the form of a whisper, with which messages are only sent to the specified user. All relevant messages sent, even ones sent by the same client, should be displayed on the client's GUI. A list of online users is displayed once a client is run, and updates in real time. Both the server and any client disconnecting at any time should be gracefull, and not disturb any other parts of the service.

2 Unimplemented Features

From the set of required features in the above introduction, we have implemented all of them.

3 Additional features

- If a client cannot establish a connection to the server, or loses its current connection (i.e. Server stopping). Instead of closing, the client continuosly tries to reconnect to the server, and on connection, seamlessly re-establishes the connection.
- Clients have the added feature of being able to mute other users. The feature is one of the available commands, along with whispering and logging out. These commands are typed in the chat and so they are easy to use.
- Apon server shutdown, a countdown is broadcast to all online users, notifying them. At the end of the countdown, clients are given the option to remain open, or to stop as well. Any user trying to connect to the server during this closing sequence is refused.

4 Description of files

4.1 Client

- Client Class:
  The client class is responsible for connecting to the server, and maintaining all input and output from the server. This class therefore holds the input and output stream, and assigns these to the relavent threads on startup. Any server feedback, including server shutdowns, are handled in this class.

- GUI Class:
  The GUI class is responsible for all graphical display, and output to the user. It handles displaying messages, and currently connected users, as well as allowing users to type and send new messages (which are themselves also displayed). This class makes use of locks and data structures to ensure messages and other elements painted to the GUI do not create race conditions when done concurrently.

4.2 Server

- Server Class:
  The server class keeps track of all clients connected. It is responsible for establishing new connections, and then creating and passing these on to a slave class. The server has a broadcast function, which allows it (and other slaves) to send messsages to all currently connected clients. It also stores administration information about users connected etc., and passes this on to all clients.

- Slave Class:
  A new slave is constructed for every client that connects to the server. The slave makes use of the server's broadcast function to send messages to all currently connected clients. Whisper's are handled by the slave by sending only to the target of the whisper, instead of broadcasting to everyone.

5 Program description

The service comprises of two distinct parts, namely the server and the client. There is only one server for a given instance of the service, that handles multiple clients at a time. How the server does this is by having a slave thread handle each client seperately, and managing the slave threads instead of the clients themselves. The clients make use of sender and reciever threads to contact the server, maintaining concurrency and allowing for server interaction.

The program flow starts with a running server and at least one running client. The client(s) send messages via their sender thread to the server. These are then either broadcasted to all users, or sent to specific ones accordingly.

The clients then recieve these messages using their reciever thread, and display them accordingly. And administration messages sent by the server are recieved by clients in the same way, but handled differently, allowing for easy implementation.
Any part of the service disconnecting only breaks the chain of flow in the service and does not affect the components it is connected to. This means that a client, that is connected to the server, disconnecting, does not affect the server, but rather just notifies it. Likewise, the server disconnecting/stopping, only stops messages from being sent using the server, and merely notifies the client.

6 Experiments

6.1 Server Devolopment

When devolping the server, we did not have a client, so the sockets where tested via the CLI tool "telnet", when connecting to the server, we would run

```
$ telnet <hostname> <port>
```

This tool allowed me to test the communication between the server and the would be client, to ensure the correct control signals where sent, and that the messages would "flow" correctly. At the start of the project, the Server would simply echo messages sent to it, and slowly began to evolve over time,

Early on the development of the server, we made the decision to have a single master thread, handling all new connections, and once a connection was established, it would be forked to a slave thread, to handle actual passing of the messages, we decided on this approach, as it allowed the server to accept new connections, whilst other users are already connected, we also decided on using multiple threads, apposed to a queue based system, as the threads wont actually be doing a large amount of work, so we can afford having mant threads.

We also realised that to have non-trivial features, we would have to package our messages in a structured format, as "raw" text wont aid in information such as who sent the message etc, and so, messages sent by the server had the form

```
server:command:extra
```

With "server" indicating the sender is the server (and has priviliged commands), "command" being a the specified command, and "extra" being any extra information needed for the command, an example would be, "server:message:Hello, World!" which would broadcast the text "Hello, World!" to all connected clients, this is one of the commands available, others include, sending connected users (either as a group, or as a single username) and sending users who disconnected.

Towards the end of the project, a lot of testing was done on clients connect, and disconnecting, to ensure that the server would remain stable, and that other clients would not be effected, one of the "hardest" areas was, when a client attempts to connect whilst the server begins its shutdown sequence, but with correct testing and locks, depending on when the client connects, the client would either successfully connect (after supplying name), or, would be told the server is offline, and would begin the attempt to re-connect, or if the client attempts to connect right as the server goes down, it would be told the server cannot be reached, and would exit.

6.2 Gui Experiments

Starting out with the project, the initial idea was to do a very simplistic GUI with more focus on the functionality. Although this focus stayed throughtout the design, the GUI changed from very simplistic to more sophisticated through experimentation. The initial experiment was just to see if we could do a GUI with text printed out for the messages. Once we had this working, we decided to move onto trying GUI layouts.

The goal was to use a layout that allowed for the different components of the finished service that is available now. To start, a box layout seemed most fitting. This however turned out to be more troublesome for the intended purpose due to the very specific parameters for it. After looking over the flow layout, the choice was made to use GridBag Layout for it's flexibility and suitability for this project.

The GUI was all coded manually, and so writting this took a series of experimentations with different values and parameters to get right. The three-pane application was finally acheived through the GridBag layout, using horizontal and vertical weights, and hiding the borders between the panes.

Finally, the scrollable pane used for messages which was initially experimented with using a normal JPane, was revised to include a JScrollPane, to accomodate for this.

7 Issues encountered

At first we expirienced some concurrency issues with the server, when multiple users connected to the server at once. These were quickly solved using locks and java concurrency structures. The major issues after that were to do with client and server disconnection. These issues were centered around clients or the server crashing on errors when the other disconnected. These were all solved in the final submisssion however.

8 Compilation

A full description of compilation for this project is described in the README file in the gitlab repository. A short summary is given below.

8.1 Compiling

Simply run,

```
$ make
```

9 Execution

A full description for execution of this project is described in the README file in the gitlab repository.

9.1 Running the server

An optional port can be given as an argument, if none is provided, the defualt port: 8199 is used.

```
$ java Server <port>
```

9.2 Running a client

An optional port and host can be supplied, if none are provided, the defualt port: 8199, host: localhost are used.

```
$ java Client <port> <hostname>
```

10 Libraries used

The libraries used, by the server,

- java.net
- java.io
- java.swing and
- java.awt for the GUI