Homework 1

Networkprogramming ID1212

Diaco Uthman diaco@kth.se

10/12-2017

1 Introduction

The goal of this homework was to develop a Client-Server distributed application using Java. The application chosen was the Hangman game.

2 Literature Study

The literature study consisted of watching the tutorial videos provided on Canvas, along with discussion with other students, in how to solve the problems that occurred.

3 Method

To solve this project, NetBeans IDE was used mainly. Although when trying out different examples, Notepad++ was used.

Before starting the project, it was important to have an idea of the design. The videos that were given, along with looking at the source files of the provided codes, gave a good understanding of what was to be done.

To make sure that the application fulfils the requirements that were given, every requirement was thoroughly tested and approved, using a checklist to sign off every requirement.

4 Result

The source codes have been submitted on GitHub in the repository given in the link below:

Figure 1: https://github.com/FlyHighXX/ID1212-Networkprogramming

Link to the source codes of the application.

Some of the requirements will be shown and proven below

- **Client and server must communicate by sending messages over a TCP connection using blocking TCP sockets.**

To fulfil this requirement, a set of sockets were created both on client and server side. These sockets communicate with each other to send the required data between them. On the client side, a socket is created in the ServerConnection.java file. It is only created once

the client requests to establish a connection to the server. Otherwise there is no need for a socket to be created. Once the client tries to connect, the server will accept the connection if everything is entered correctly (IP-address and port). Once this is done, the server will generate a client socket, that will be used for communication with that specific client.

- **The client must not store any data. All data entered by the user must be sent to the server for processing, and all data displayed to the user must be received from the server.**

All data that is used in the client view, are received by the server-side classes. To achieve this, it was important to make sure that any game-related data was stored on the server. This could of course be done in many ways, for example in a database, or as variables in a Java class. Because of the simple nature of the game, no database was implemented. The relevant data of the game, such as *score*, *current word* and *remaining attempts*, were stored in the Hangman.java class on the server-side. Each time the client wants to see these data, it must make a call to receive them. An example of this is shown in Figure 1.

```
start
---- HANGMAN MENU ----
---- Write one of the following commands, followed by required arguments ----
---- connect, start, guess, gameinfo, quit ----
```

Figure 1: The user enters the command start, which will start a new game. Then the word will be received, and the user can start guessing letters.

- **The client must have a responsive user interface, which means it must be multithreaded. The user must be able to give commands, for example to quit the program, even if the client is waiting for a message from the server.**

To achieve this requirement, it was important to make a non-blocking client-side. Because, with this implementation it would be possible to make multiple requests to the server, and await the answers without having to stop the execution. To do this, the CompletableFuture() class was used in the Controller.java file. Namely the runAsync() method was used, to make the calls asynchronized, meaning they can be called at any time without depending on each other. This implementation makes sure that the user interface is responsive, and multithreaded.

- **The server must be able to handle multiple clients playing concurrently, which means it must be multithreaded.**

To fulfil this goal, an implementation was done in the server. Once a user connects, and the client socket is generated, a new thread will be created. This thread will handle all the communication with that specific client. By doing this, the server will have its own separate thread, and thus we will have a multithreaded implementation of the program.

- **The user interface must be informative. The current state of the program must be clear to the user, and the user must understand what to do next.**

All the user interface is in the command line, and no graphical interface was implemented. If this project was to be developed further, then a graphical interface would be a much more user-friendly solution to use.

When the client opens the program, a menu will be shown. The menu is shown in Figure 2. Of course, the menu could be developed much more, but this is sufficient for a user with knowledge about the source code. If the user has no idea, then a much more developed interface must be created. Further, when the new game is started, after connection has been established, then Figure 3 shows the word.

```
---- HANGMAN MENU ----
---- Write one of the following commands, followed by required arguments ----
---- connect, start, guess, gameinfo, quit ----
```

Figure 2: The first information that is shown upon starting the client-side.

```
---- HANGMAN MENU ----
---- Write one of the following commands, followed by required arguments ----
---- connect, start, guess, gameinfo, quit ----

_ _ _ _ _ _ _ _ _ _
guess a
```

Figure 3: Shows the menu, the word that has been chosen for the current game, and the user enters that he wants to guess the letter a.

The command *gameinfo* will provide the client with information about the session, such as the current score, the remaining attempts in the current game. If there is no current game, then the user will be notified that there is no current game running. This is shown in Figure 4.

```
gameinfo
---- HANGMAN MENU ----
---- Write one of the following commands, followed by required arguments ----
---- connect, start, guess, gameinfo, quit ----
Current score: 0
Current remaining attempts: 11
```

Figure 4: The user specifies that he wants to see the *gameinfo*. The menu is shown for the next command, along with the game information. In this case, the users score is 0, and the user has 11 attempts left of the current word.

5 Discussion

All mandatory tasks have been completed. A lot of the time was spent trying to make a graphical interface, but in the end the command line interface was used. Therefore, it may not be the most developed interface that is shown in the report. In the future, it would be better to develop an interface that is easy to make, and then focus on improving with graphical interfaces and such, later.

Furthermore, when it comes to the implementation of the system, it would be wiser to focus on the server side first, and then implement the client. In my case, I started with the

client, and then tried to develop and adapt the server to work with the client. If I had started with the server and then implemented the client, there would be a better understanding of how the client should communicate with the server, before trying to write the code for it.

6 Comments About the Course

Because I started so late with the assignments, they didn't go as my plan, and hence needed to start over many times before actually completing the assignment. It would have been wiser to start earlier of course.