Homework 2

Networkprogramming ID1212

Diaco Uthman diaco@kth.se

12-06-2018

1 Introduction

The goal of this homework was to develop a Client-Server distributed application using Java with Non-blocking sockets. 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

- **That only non-blocking sockets are used.**

In this task, the Netty framework was implemented. Netty is a Non-Blocking framework, that makes sure that the communication is non-blocking. The classes that handle the setup and connection from both sides are, from the client side *ServerConnection.java*. From the server-side the class that handles the requests, is *ProcessingHandler.java*. To send the data

over the connection, two classes have been written. The *RequestData.java* and *ResponseData.java* classes. These are encoded and decoded when sent over the connection.

```
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.**

In theory, this framework should make it possible for multiple clients, to send their requests in a non-blocking fashion. The requests should be handled straight away, and then sent to threads that handle any I/O executions. Using two and three users at the same time, seemed to be working with this implementation. All the users had a responsive interface, and did not need to wait for other requests. The users could quit the program at any time, without needing to wait.

- **That the program still works as expected. You show this by including screenshots of the user interface**

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

This task became a lot easier once the Netty framework was discovered and implemented. Making all the steps from scratch, like in HW1, took a lot more time. Learning about a new framework is also never a bad idea. They are after all made to be used. Because the business logics of the application was identic, they could pretty much be copied from HW1. Also, the View layer was almost entirely copied from HW1. This was very good, because then the only challenge in this assignment, was to understand the concept of non-blocking sockets.

From the beginning however, an attempt was made to change some parts of HW1, to make the sockets non-blocking. This was not a good idea, because the application was big and had dependencies such as requests from the View layer, going to the Net layer on the client side. These requests created some problems, and in the end therefore I decided to start from scratch and import the relevant parts, one at a time. Then, once the Netty solution came along, I decided to go with that approach.

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.