Assignment 1 Report

COMP90015 - Distributed Systems (Semester 1, 2020)

Harfiyanto Santoso - 772503

harfiyantos@student.unimelb.edu.au

1 Introduction

The aim of this project is to design a Multi-threaded server based on client-server architecture using Java. The two fundamental technologies introduced so far in the subject, socket and thread, are used to facilitate the connections between the server and multiple clients (each with its own thread). This report would describe the context of the problem, along with the structure of the program and analysis of the work done.

2 Problem Context

There are few specifications that have to be followed:

1. The system needs to explicitly use sockets and threads.
2. All communications should be reliable and take place via sockets (either TCP or UDP).
3. A message exchange protocol should be designed to store and read the data
4. Ensure that a single (multi-threaded) server can handle and perform requests from multiple clients concurrently.
5. Both the server and client side errors need to be managed (exception handling). These includes:
   (a) Input from the console for what concerns the parameters passed as command line.
   (b) Network communication (address not reachable, bad data...).
   (c) I/O to and from disk (cannot find the dictionary file, error reading the file, etc...).

3 Architecture and Design Decisions

Following design decisions were made:

1. TCP would be used as the communication protocol for its reliability.
2. The multi-threaded server implements thread-per-connection architecture. This is because the system is not highly-concurrent so a simple architecture would suffice.

4 Classes

The application consists of three classes: ServerWindow, ClientWindow and Dictionary.

4.1 ServerWindow

This class is handles the GUI for the Server. It waits for client socket requests, approves them and creates a corresponding thread (Dictionary class) for each user.

4.2 ClientWindow

This class handles the Client GUI as well as sending requests (querying the definition of a word, adding a word, removing a word) to the dictionary server

4.3 Dictionary

Each instance of this class acts as a thread and is assigned to a particular client to process its requests. It is responsible for reading the dictionary file and making relevant updates.

5 UML Diagram

The following is the simple diagram of the relationship between classes. There is only one server object and the same number of ClientWindow and Dictionary objects. This is because the server assign each client with a thread (Dictionary object) to perform its requests.

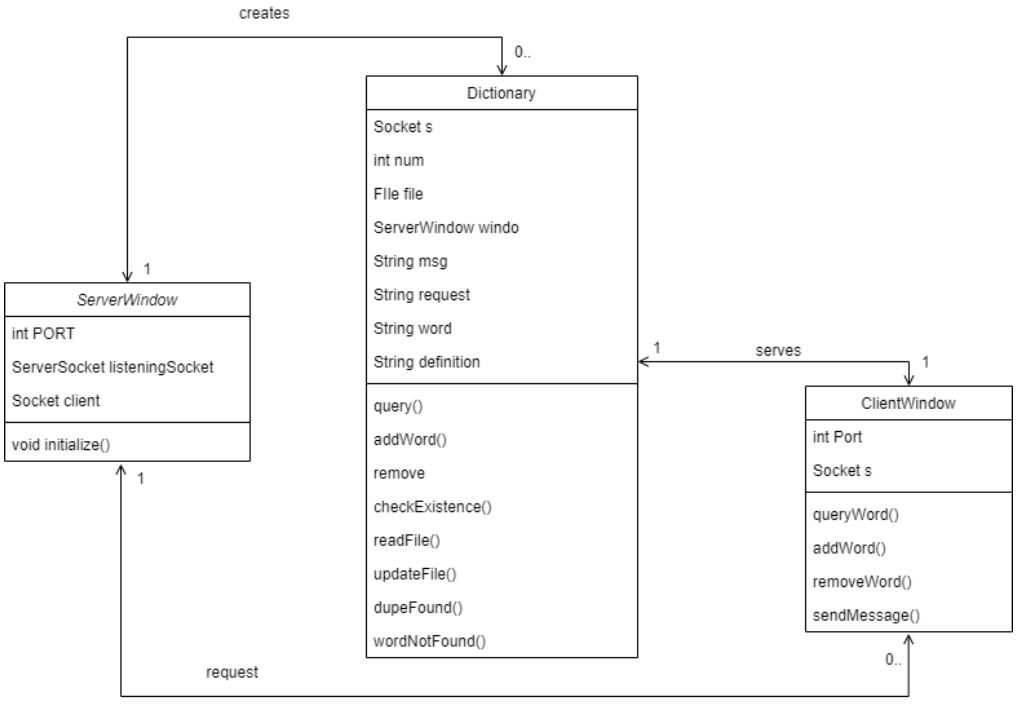

Figure 1: UML Diagram of the System

6 Graphical User Interface (GUI)

6.1 Server

The GUI for the server-side is fairly simple, consisting only a text field that acts as a log and a label to indicate the port number that the server is connected to. The log is updated whenever the client is connected, sent a request or disconnected.

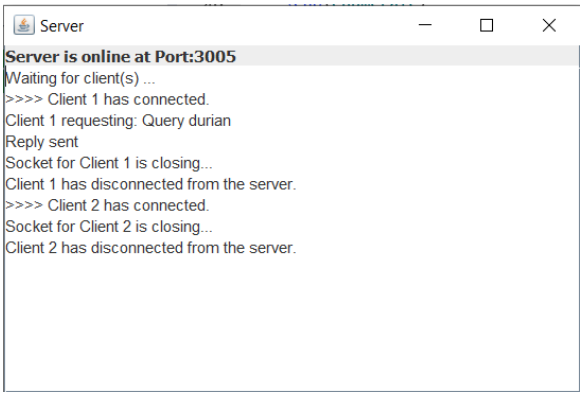

Figure 2: Server GUI

6.2 Client

The GUI for the client-side consists of two text fields and three buttons. The three buttons corresponds to the request sent to the Dictionary (Query, Add, Remove). The text field at the top is the input for the word being queried, added or removed and the field at the bottom is the input for definition(s) of the word being added.

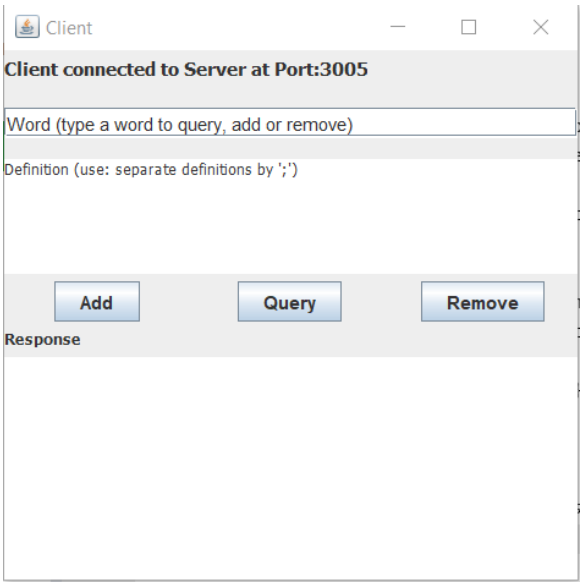

Figure 3: Server GUI

7 Interaction Diagram

The following is the interactions between the classes in a simple case of two clients trying to connect and request from the server. The first client sent the socket request to the server first, followed by the second client. The server responds by creating a thread for each client that can then process their request. All this happens concurrently thanks to the multi-thread architecture of the system.

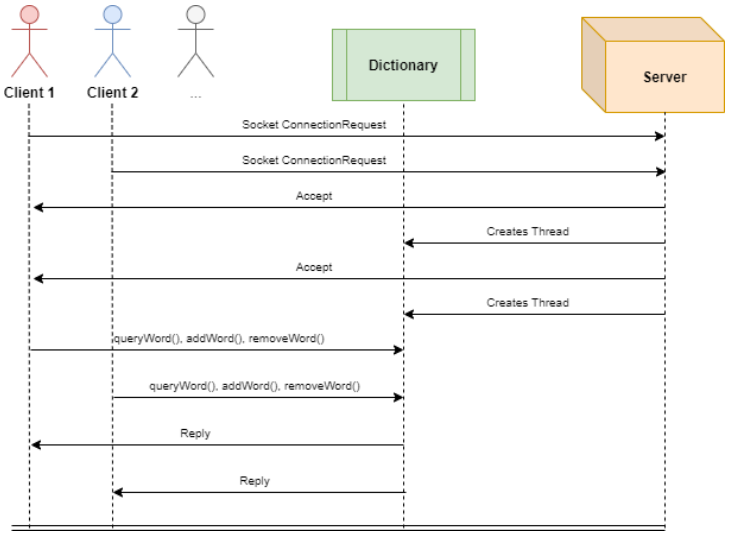

Figure 4: Example Interaction Between Classes

8 Synchronous Access to Shared Resources

The threads operate on a single JSON file so the update from a client can be seen by another client. To show this the following set of requests were executed between two clients and a sever. The dictionary did not contain the word 'mango' initially so both queries from the two clients yield the warning message.

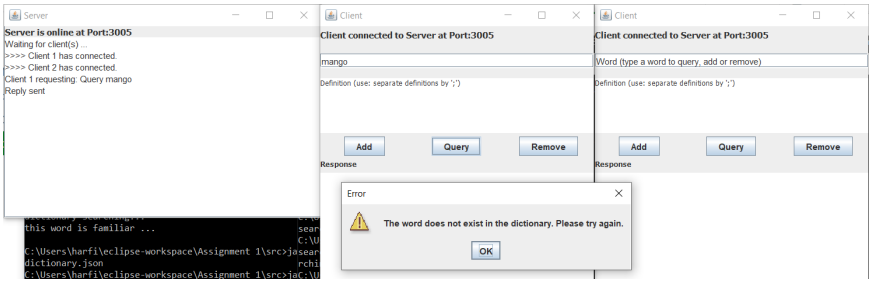

Figure 5: Client 1 Query 'mango'

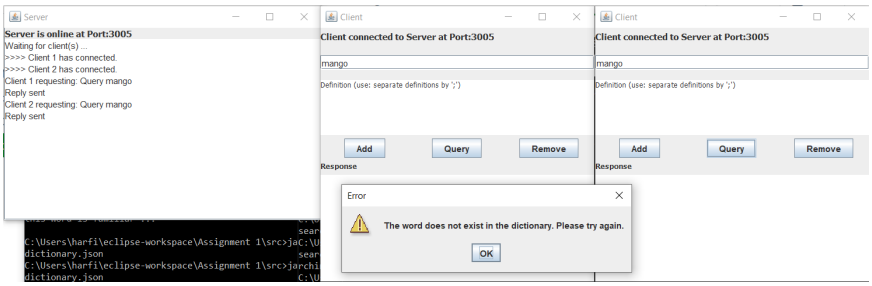

Figure 6: Client 2 Query 'mango'

Client 1 then adds the word 'mango' into the dictionary using the add button. This time, when Client 2 query 'mango' the second time, the definitions appear on the response field.

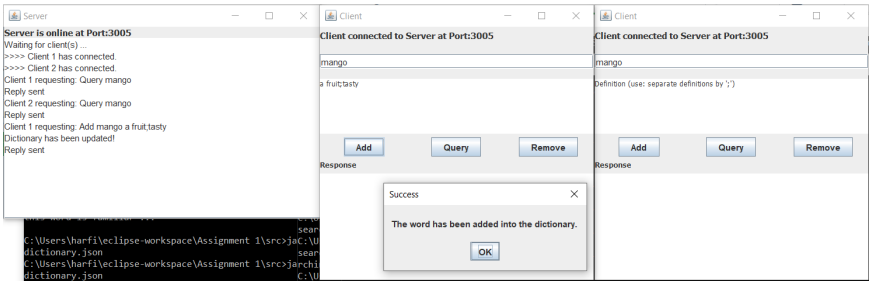

Figure 7: Client 1 Add 'mango' and its definitions

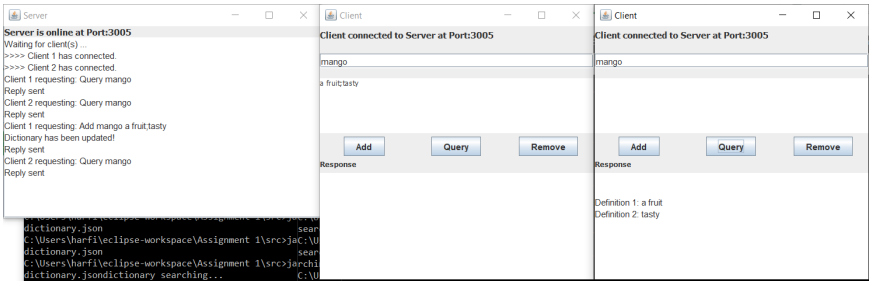

Figure 8: Client 2 Second Query 'mango'

The opposite also applies, when Client 1 removes the word 'mango' from the dictionary using the remove button, Client 2 could no longer find the word in the dictionary.

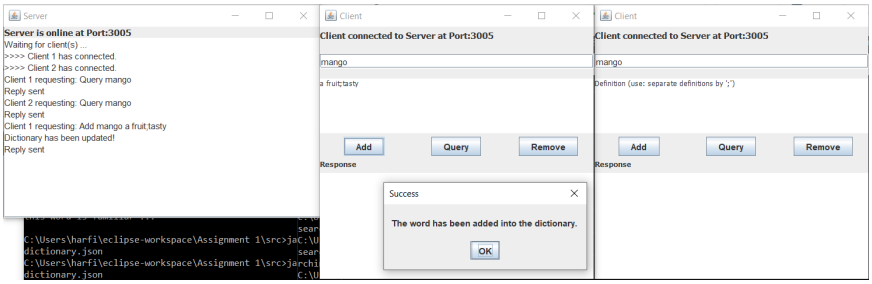

Figure 9: Client 1 Delete 'mango'

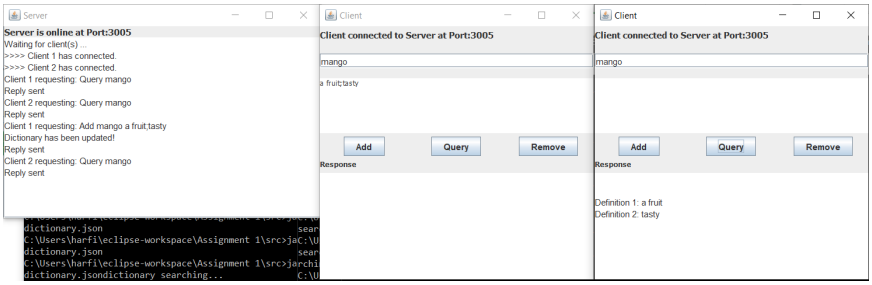

Figure 10: Client 2 Third Query 'mango'

This implementation also allows the synchronous access across the clients. This is done through by synchronizing the access to the shared resources by different threads using synchronized methods (e.g. private synchronized void addWord()). This ensures concurrency in the system.

9 Analysis and Conclusion

In this implementation, the server can fulfill the requests from multiple clients concurrently. It has also meet all the problem specifications listed in Section 2. However, there are some rooms for improvement in the design decision and assumptions made. Firstly, the thread-per-connection architecture was chosen because of its relatively simple implementation and the assumption that the operation of this application does not demand high concurrency. However, this means that once the number of clients increase significantly, the number of threads needed increases and takes up more and more resource from the server. Hence, this implementation is relatively not scalable. This issue can be improved by using a Worker pool architecture instead to limit the number of threads available for the clients.

Another improvement that can be made is to enhance the functionality of the Dictionary class or add more features. For example, the current implementation has a limitation where it cannot add new definition(s) to a word that is already in the dictionary. Instead, the client needs to manually remove the word and add the same word back again with the extra definitions. In this case, extending the functionality of the Add button to account can solve the issue.