COMP90015 - Distributed Systems (Semester 2, 2018)

Assignment 1
Multi-threaded Dictionary Server

Shubham Bawa
936127
sbawa

Tutor's name: Lakshmi Jagathamma Mohan

Introduction

The project was about building a client-server architecture to implement a Multi-threaded Dictionary Server that allowed concurrent clients to add words with meaning(s), search for the meaning(s) of a word and delete existing words from the dictionary. The architecture makes use of TCP Sockets and Threads at the lowest level of abstraction for the purpose of network communication and concurrency.

System Architecture

Since the system has a Client-Server architecture, it enables multiple clients to connect to a single Multi-threaded server concurrently and perform operations such as searching for a word's meaning (Query), adding a word and its meaning to the dictionary (Add) and removing an existing word from the dictionary (Remove). The server performs the necessary tasks for all the operations and displays the word and their meanings (Load operation) that are present in the dictionary in the form of a graphical user interface. The dictionary stores the words and their corresponding meanings in the .txt format. The communication between the Client and the Server follows the JSON data format and uses TCP Sockets for making the connection between them.

The following is a representation of the System architecture that has been designed:

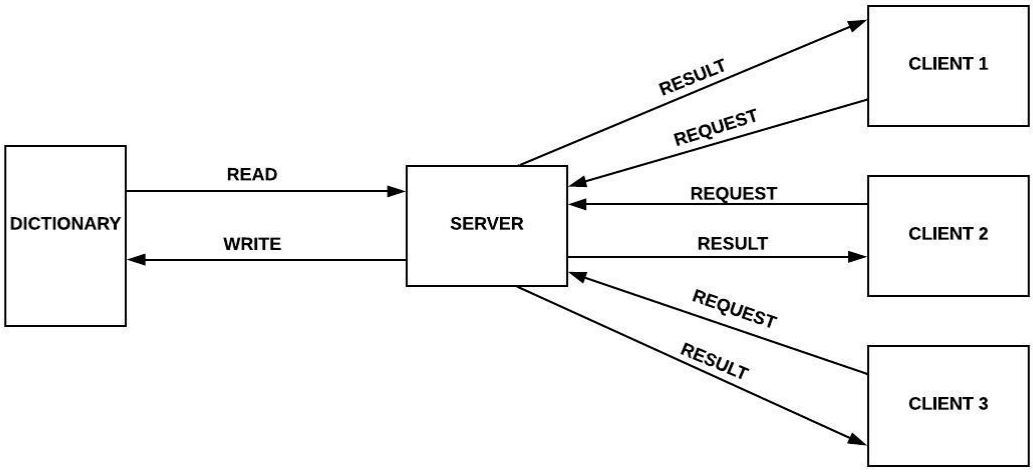

Figure 1. System architecture of Multi-Threaded Dictionary Server

The various components of the architecture are:

1. **Multiple clients**: They send a request to server (Query, add or remove).
2. **Server:** Receives the request, processes it, writes changes to the dictionary and replies back with the result.
3. **Dictionary:** Saves the words and their corresponding meaning.

Java classes design

The design of the various classes used for the project can be understood with the help of the following diagram

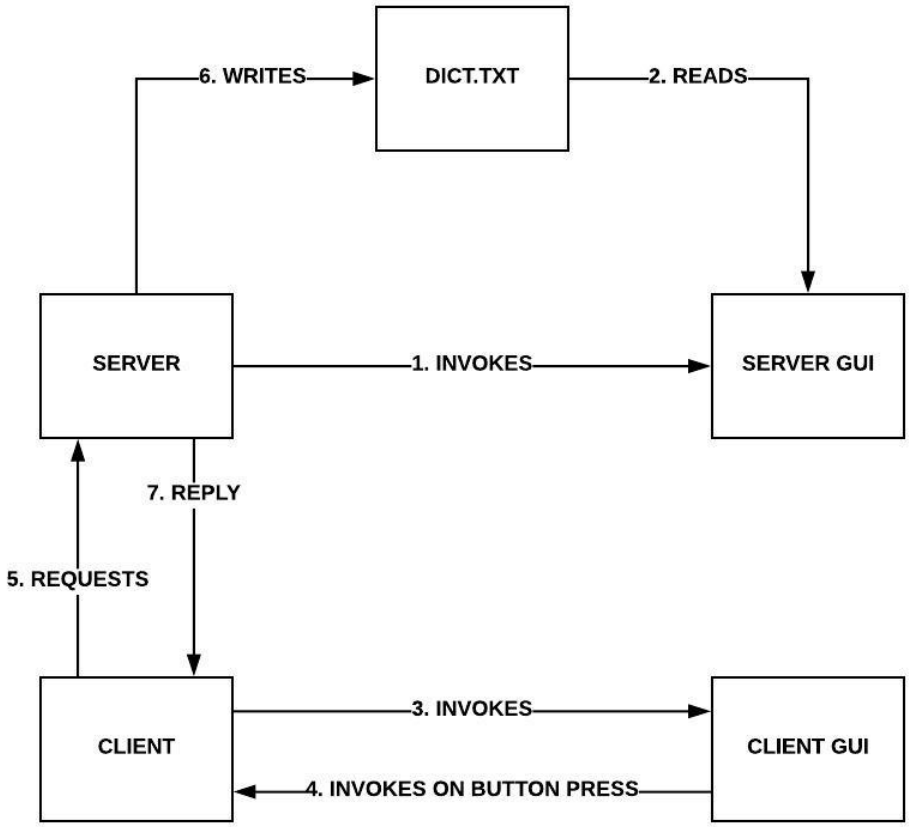

Figure 2. Class design

- **Server-side design**

The classes and files used on the server side are:

1. **Server.java:** Invokes the Server's graphical user interface and creates a socket to listen to a port for incoming connections, when it gets a request, it establishes a connection with the client and creates a new thread to take requests such as Query the meaning of a word, add a word or removing a word from the dictionary. Then calls the appropriate function to handle the request, writes the changes to "Dict.txt" and returns the results or error messages, if any, to the client.

2. **ServerGUI.java:** Creates a graphical user interface and reads the "Dict.txt" file when a "Load" button is pressed and displays the words and their corresponding meanings. It also displays errors messages.

3. **Dict.txt:** Stores all the words and their corresponding meaning(s), separated by ":" as "Word:Meaning" in the text format. For example, "Mobile:able to move or be moved freely or easily".

- **Client-side design**

The classes used on the client side are:

1. **Client.java:** Invokes the Client's graphical user interface and calls the function based on action listeners in "ClientGUI.java". Then makes a request to the server based on the type of operation user requested (Query, add or remove), receives the result or error message from the server and gives them to "ClientGUI.java" to display them to the user.

2. **ClientGUI.java:** Creates a graphical user interface for the Client, which has "Query", "Add" and "Remove" button for querying the meaning of the given word, adding the given word and its given meaning and removing a given word from the dictionary respectively. It also has a "Result" text area which displays the result of the client's operations, which can be success, failure or error.

Message exchange

The message exchanges between the client & the server are done using JSON data format.

- **Client to server**

A client sends a request to query, add, or remove, which has been implemented in the program as follows

1. **Query**

```
JSONObject data = new JSONObject();
String type = "query";
data.put("queryType", type);
data.put("userWord",word);
data.put("meaning", meaning);
```

Here, the "queryType" is "query", and the "userWord" is the word given by the client. The meaning is empty.

2. **Add**

```
JSONObject data = new JSONObject();
String type = "add";
data.put("queryType", type);
data.put("userWord",word);
data.put("meaning", meaning);
```

Here, the "queryType" is "add", and the "userWord" and "meaning" are given by the client.

3. **Remove**

```
JSONObject data = new JSONObject();
String type = "query";
data.put("queryType", type);
data.put("userWord",word);
data.put("meaning", meaning);
```

Here, the "queryType" is "remove", and the "userWord" is the word given by the client. The meaning is empty.

- **Server to client**

After receiving the client's request for query, add or remove, the server responds back with the result, which has been implemented in the program as follows.

```
JSONObject toClient = new JSONObject();
toClient.put("display", display);
```

Here, the "display" is the message that has to be displayed on the client's graphical user interface. This could be the meaning of a word for "Query" operation, message saying "Word added to dictionary" for "Add" operation, message saying "Word deleted" for "Remove" operation or could be a message reporting an error that might have occurred for any operation.

Analysis

The architecture of the project involved us creating a Client-Server model with data storage, wherein, the Server side of operations such as creating a graphical user interface, receiving data from the client, processing it, writing the data to a text file and sending back the results to the client while also reporting errors, if any, were indicated clearly. Also, the client side of operations such as creating a graphical user interface, taking user's input, sending them to the server, getting results from the server, reporting errors to the user, if any, were indicated clearly as well.

The overall system provided reliable and connection-oriented message exchanges between the client and server with the help of TCP sockets. It also provided concurrency, wherein, two or more clients could easily perform the "Query", "Add" or "Remove" operations simultaneously and the system will be able to handle them without any trouble. The system largely depended on the JSON data format for exchanging messages between client and the server such as request from the client to the server and replies from the server to the client.

The design choice of using TCP over UDP was made on the basis of reliability. TCP is considered to be a reliable and connection-oriented protocol and since I had some experience with using TCP and also the project had to have reliable communication between the components, it made it obvious to choose TCP. As for UDP, being unreliable and connection-less, a separate infrastructure would have to implemented to offer reliability, making it difficult to implement.

The reason behind using JSON data format for message exchanges was that this format provided easy transfer of the messages over the TCP Sockets as it utilized less CPU resources in comparison to the XML data format and was easy to implement as well. Hence, making it easy for the programmer to read and write the data and for the machine, to send and receive data.

Conclusion

In conclusion, the required Multi-threaded Dictionary Server system was implemented successfully that could create graphical user interfaces, query the meaning of a word, add a word and its meaning to the dictionary, remove a word and its meaning from the dictionary, store data, report errors on the server side and the client side, handle concurrency and provide reliable communication between the client and the server.