The University of Melbourne

School of Computing and Information Systems

COMP90015 Distributed Systems

(Semester 2, 2017)

Assignment One – Multi-threaded Dictionary Server

Shuai Wang

August 30, 2017

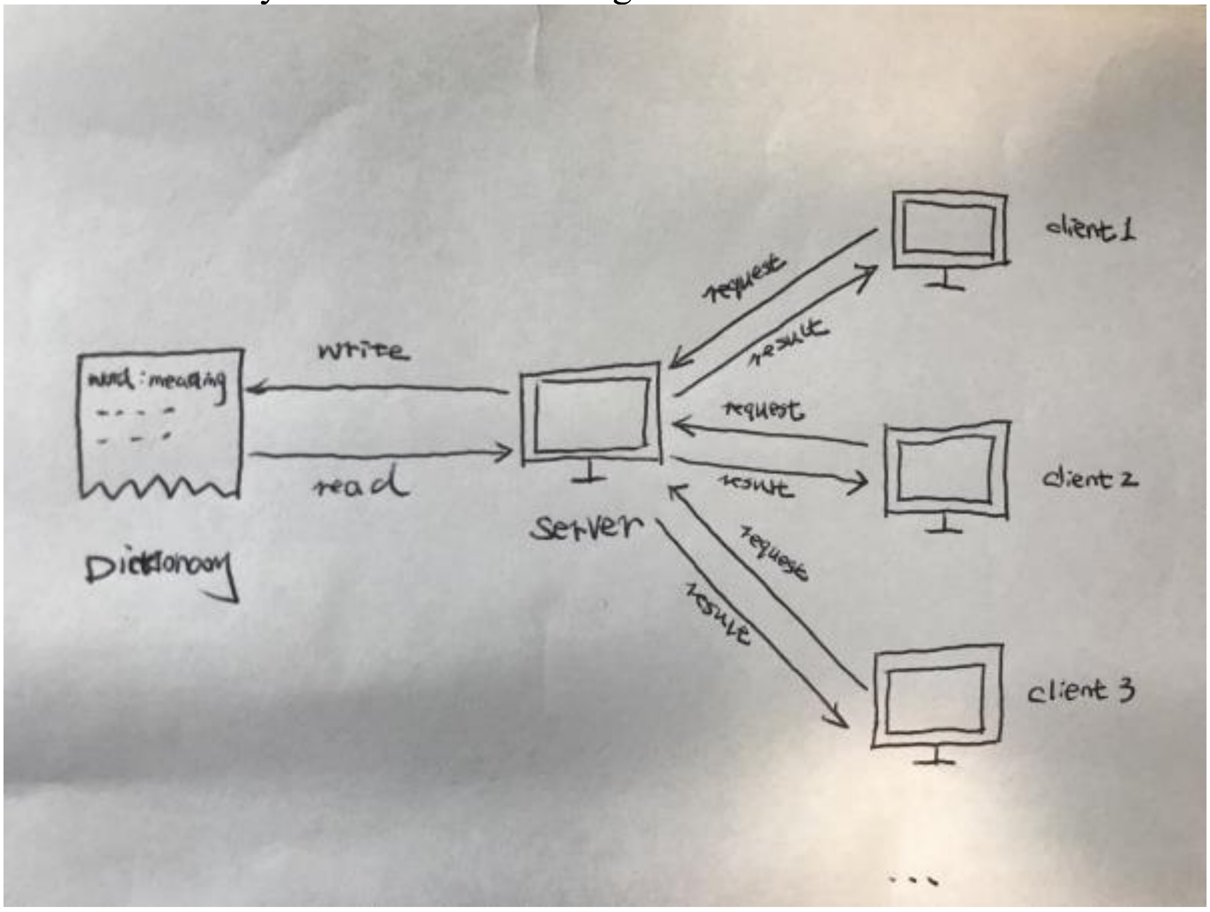
1. Problem

The report attempts to design and implement a multi-threaded dictionary server that allows multiple clients to concurrently search the meaning of a word, add a new word and remove an existing word based on a client-server architecture. Java sockets and threads are used for the lowest level of abstraction for network communication and concurrency.

2. Architecture

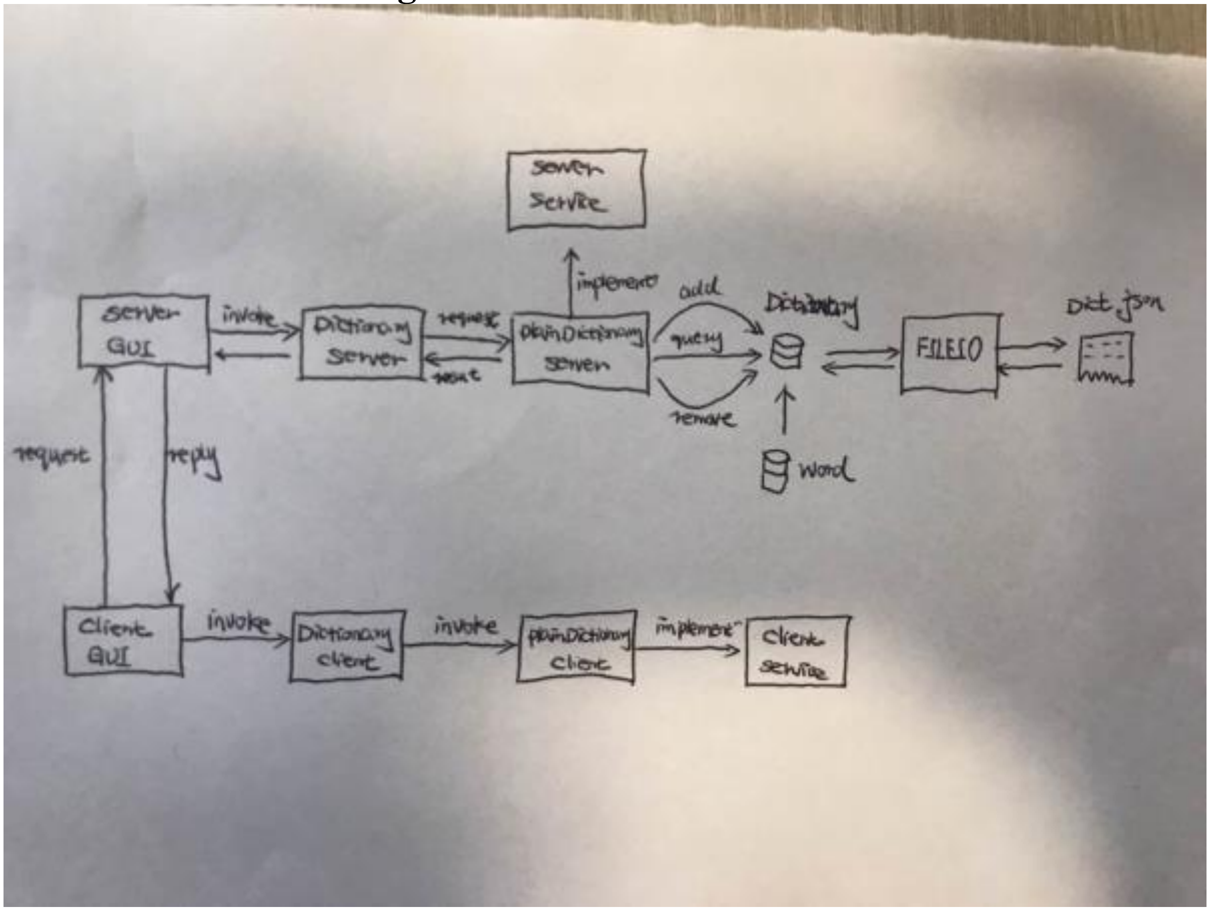
The system follows a client-server architecture in which multiple clients can connect to a single multi-threaded dictionary server and performs “Query”, “Add” and “Remove” operations concurrently, and the server can perform “Load” operation to load all words and meanings in dictionary and show on screen. The dictionary file uses JSON format to store words and corresponding meanings. The exchanged data format between client and server is also using JSON. All communications will take place via Sockets based on TCP. The server implements a thread-per-request architecture based on Java Threads.

An illustrative system architecture diagram follows:



Components in the architecture are:
Client: sends request to server.
Server: accept request, process request and reply client.
Dictionary: holds a list of words and meanings.

3. Java Classes Design



On the server side:
ServerGUI.java: generates a visual window which allows to show all words and their meanings loaded from "Dict.json" file.

DictionaryServer.java: obtains requests from clients, such as adding a new word, querying the meaning of a word and removing an existing word. Then it parses the requests and invokes corresponding methods to process. Finally, send results back to clients.

PlainDinctionaryServer.java: implements adding, querying and removing operations.

DictionaryService.java: lists all services that the server can provide. This is an intnerface.

Dictionary.java: contains all words and the number of words. This is a Java bean.

Word.java: represents a real word containing a word spell and corresponding meaning.

FileIO.java: handles reading from "Dict.json" file and writing to it.

Dict.json: contains words and meanings in Json format. For example, {"word":"teacher", "meaning": "A teacher is a person who teaches, usually as a job at a school or similar institution"}.

StartServer.java: creates a serverSocket to listen to a port, once gets a connection request, creates a new thread to handle the request. At the same time, it launches the server GUI. This is the entry of server program.

On the client side:
ClientGUI.java: generates a visual window that provide adding, querying and removing operations for a client.

ClientService.java: lists all services that the client program is required to provide. Those services are adding request, querying request and removing request. This is an interface.

PlainDinctionaryServer.java: has implemented all services client requires.

StartClient.java: launch the client program and client GUI has been shown.

Exchange data:
Request: the client send operation request to server. The request format is as below:

```
{
"operation" : "add"|"query"|"remove",
"word" : ""|"word",
"meaning" : ""|"meaning"
}
```

For example:
Adding request:

```
{
“operation” : “add”,
“word” : “teacher”,
“meaning” : “A teacher is a person who teaches, usually as a job at a school or similar institution”
}
```

Note that “word” and “meaning” cannot be empty.

Querying request:

```
{
“operation” : “query”,
“word” : “teacher”,
“meaning” : “”
}
```

Note that “word” must be provided while “meaning” can be empty.

Remove request:

```
{
        “operation” : “remove”,
        “word” : “teacher”,
        “meaning” : “”
}
```

Note that “word” must be provided while “meaning” can be empty.

Reply: server sends result back to client after processing. The data format of reply is as below:

```
{
        “result” : “processed result”
}
```

For example, a reply for “query” operation can be:

```
{
“result” : “A teacher is a person who teaches, usually as a job at a school or similar institution”
}
```

4. Analysis

The architecture of the multi-threaded dictionary server system is quite clear. It contains server side, client side and data storage. On server side, it clearly indicates what services the server can provide, how the server implements those services, how the server read from and write to a dictionary, what the GUI looks like and what Java beans the server are using. On the client side, it clearly describes the client windows, services and functionalities.

The system provides reliable communications between server and clients. It uses TCP protocol based on Socket, which is a connection-oriented protocol that provides reliable flow of data between server and client. Moreover, Socket is the most widely used programming interfaces for supporting TCP.

The system can handle concurrency issue. The client can perform adding, querying and removing operations. Concurrency will occur when multiple clients are operating. The system can handle it by Java Synchronization.

The system provides easily readable exchanging data. It uses JSON format as data-interchanging format between server and clients. Such as requests from clients, results produced by server, words and meaning in dictionary. This light-weight data exchanging format is easy for humans to read and write, it is also easy for machines to parse and generate.

5. Conclusion

In general, the multi-threaded dictionary server has successfully implemented a system that can provide some operations, such as adding a new word to the dictionary, querying a word and removing an existing word. It can handle concurrency issue and provide reliable communications between server and clients.