COMP90015 - Assignment 1 Report

Nodens F. Koren (1060811)

15th April, 2021

1 Introduction

In this project, we implemented a multi-threaded dictionary server which is able to process requests from multiple clients simultaneously. The server allows four basic operations:

- look-up a word (*search*)
- add a new word and its corresponding definition to the dictionary (*add*)
- remove an existing word and its corresponding definition from the dictionary (*remove*)
- update the definition of a word (*update*).

Additionally, we implemented two advanced functions which allow the user to perform quick search from the history, and allow the server to respond to minor spelling mistakes. The paper is arranged in the following order: first we will discuss about the client-server model and its implementation details, then we will give a discussion on the the thread model adopted. Finally, we will talk about how we handle errors/exceptions and our advanced functionalities.

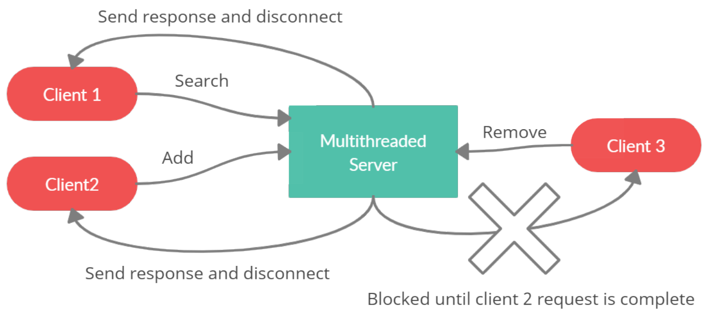

Send response and disconnect
Client 1
Search
Multithreaded Server
Remove
Client 3
Add
Client2
Send response and disconnect
Blocked until client 2 request is complete

Figure 1: Brief flow chart of the client-server dictionary application. Note that we assume client 2's request is accepted earlier than client 3's.

2 Detailed Implementations

We follow a simple client-server model which allows every single client to communicate with the server, but not directly with each other (i.e. not peer-to-peer). We believe this is how most popular dictionary servers are implemented. The users are free to update the definitions provided by another user.

The user will be connected to the server each time it sends out a *search*, *add*, *remove*, or *update* request, and will soon be disconnected once the feedback from the server has been delivered. We believe such an implementation will minimize the computational resources occupied by each client and thus maximize the total number of processed requests over time.

2.1 Detailed Implementations

Our implementation only has two components - one for client and another for server.

2.1.1 Client

The user needs to initially specify the connection address and the connection port when launching the client application. Once the client has been launched, the user can specify the word s/he wants to perform an action on, and the user will specify the action s/he wants to perform. If the desired operation is to *add*, *remove*, or *update*, the user will need to input a definition for the word as well. When the user is ready to submit the request, s/he will click the "submit" button. It is only when the "submit" button is clicked will the client actually sends out a connection request to the server.

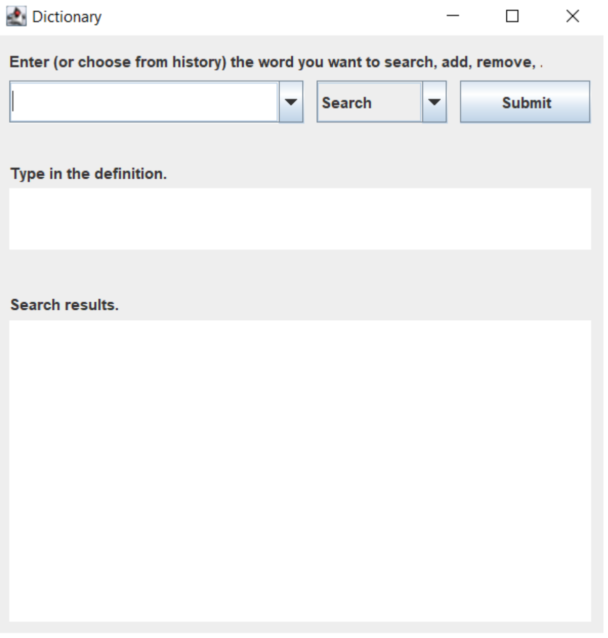

Dictionary
Enter (or choose from history) the word you want to search, add, remove, .
Search
Submit
Type in the definition.
Search results.

Figure 2: The client GUI.

In our implementation, we send out the message as an array of string object, which should contain the following components: 1. the action performed, 2. the word being queried, and 3. the definition. If the server is active, the client will receive a feedback from the server. This feedback may be the definition of the word (when a word is found in the dictionary), a success/failure message, or an error message from the server. Any feedback from the server will be displayed in the non-editable textbox, and instead of using the readLine() method, we iteratively read in all lines available from the input buffer to deal with messages that contain the new line character.

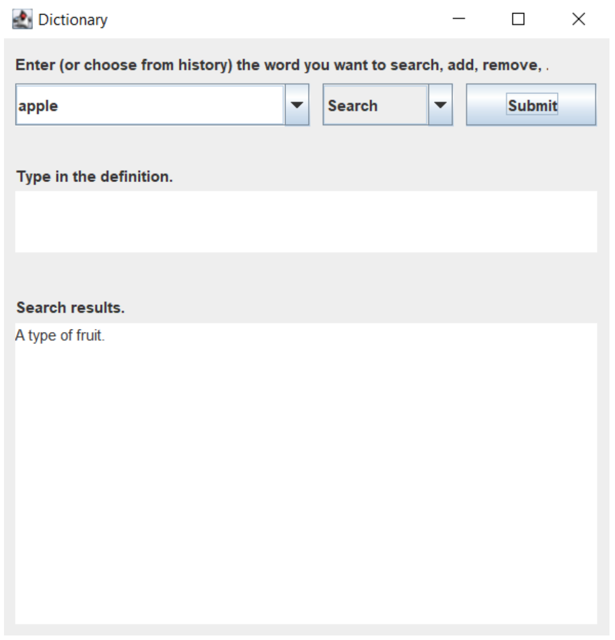
Note that, the connection to the server will be disconnected right away when the message from the server has been delivered. The client will not be connected to the server until the next time the user click the "submit" button.

2.1.2 Server

For transmission reliability, we adopted the Transmission Control Protocol (TCP) socket to manage connections between clients and the server. The server manager will need to specify the directory of the dictionary and the port s/he would like to open when launching the server application. Once launched, the port will be opened until the server application is terminated. Our system will create a new dictionary file in the specified directory if the dictionary file is not found in the specified directory. The server will then load all entries of the dictionary into a hash map data structure (a key-value pair).

When an incoming connection request is approved by the socket, the server will create a new thread for this task, which will then parse the message from the client socket. It first checks the first component of the client message (i.e. the requested action). If the requested action is *search*, the server will discard the third component (i.e. the definition) of the client message regardless of what the user has inputted. On the other hand, if the requested action is *add*, *remove*, or *update*, the third component of the client message will be used in conjunct with the second component (i.e. the word).

Search. If the specified action is *search*, the server will first check if the specified word is in the dictionary. If so, the server will retrieve the definition of the word from the hash map and send it back to the client. On the other hand, if the specified word is not in the dictionary, the server will look up alternative words (spelling correction). If there exist an alternative word that is highly similar to the specified word, the server will append the message "Do you mean -" along with the definition of the alternative word to the word-not-found failure message. If no alternative word is found, the server will simply respond the word-not-found failure message. The detail algorithm for finding an alternative word will be discussed in the advanced functionality section.


Dictionary
Enter (or choose from history) the word you want to search, add, remove, .
apple
Search
Submit
Type in the definition.
Search results.
A type of fruit.

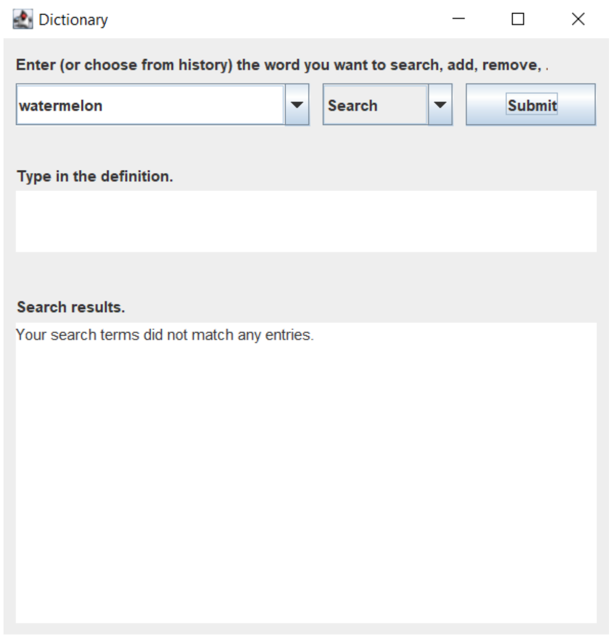
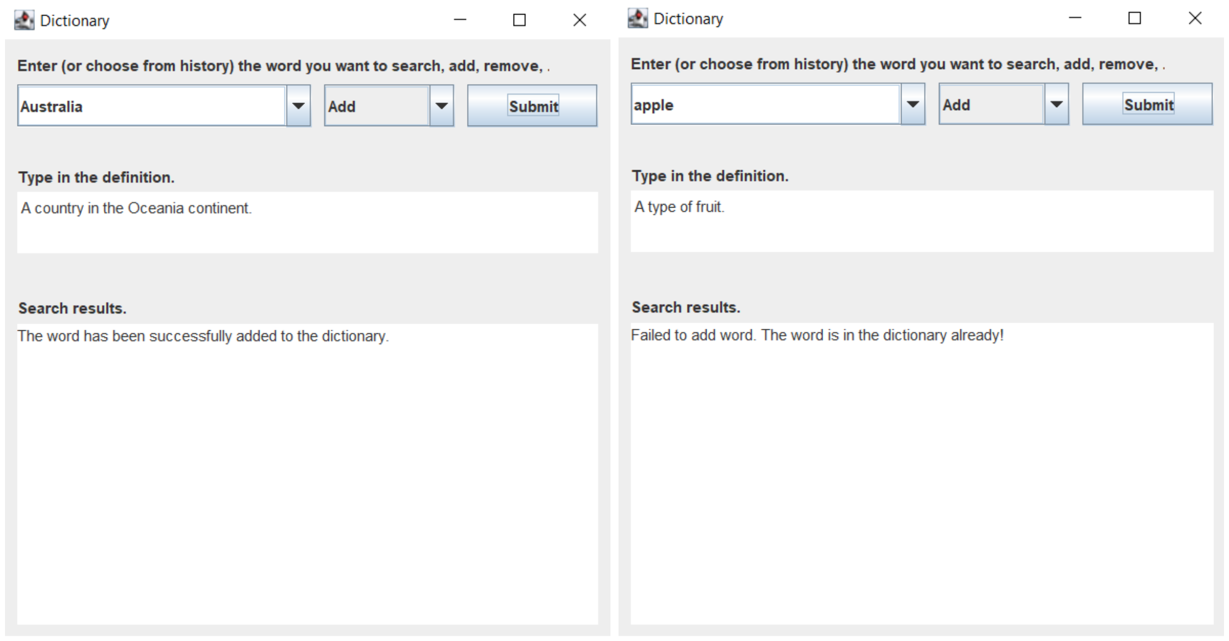

Dictionary
Enter (or choose from history) the word you want to search, add, remove, .
watermelon
Search
Submit
Type in the definition.
Search results.
Your search terms did not match any entries.

Figure 3: Successful and failed search.

We allow the *search* operation to be performed at any time as read-write and write-read conflicts can be easily fixed by repeating the read action and read-read will not lead to a conflict. We reckon that under this setting, the loss from such conflicts would be small unlike bank transactions. In particular, as the dictionary grows larger, the conflict rate would become smaller.

Add, Remove, and Update. We discuss these three actions together because they all have similar attributes. They are all write operations, and we do not want a write-write conflict to happen. Hence, we explicitly require these three operations to be *synchronized* using the Java keyword. Consequently, only one user is allowed to make changes to the dictionary at each moment.

Similar to the *search* operation, the will first check if the specified word is in the dictionary. If not, the server will not allow the client to *remove* or *update* the dictionary. Instead, the server will send a word-not-found failure message to the client without making any changes. For the *add* operation, if the specified word is


Dictionary

Enter (or choose from history) the word you want to search, add, remove, .

Australia | Add | Submit

Type in the definition.

A country in the Oceania continent.

Search results.

The word has been successfully added to the dictionary.

Dictionary

Enter (or choose from history) the word you want to search, add, remove, .

apple | Add | Submit

Type in the definition.

A type of fruit.

Search results.

Failed to add word. The word is in the dictionary already!

Figure 4: Successful and failed add.

not in the dictionary, the server will create a new entry in the dictionary hash map with the key being the word and the value being the definition. No changes will be made if the word exists in the dictionary and an error message will be sent to the client who made the request.

The server will close the connection with the client socket and save the updated hash map to the dictionary file once the request is completed.

3 Excellence Elements

3.1 Threading

We utilized the *thread-per-request* architecture. Technically, our implementation of the client-server framework does not allow the client to be continuously connected over time. Due to the reason, each connection is made only when a request is made. Thus utilizing the *thread-per-connection* architecture wouldn't be very different from using the *thread-per-request* architecture as no client will be blocking the server anyway.

3.2 Error Handling

We catch the following standard errors in the assignment specification:

- Incorrect parameters error: If the user enters a number of parameters that are not equal to 2, we remind the user about the standard format of the command line input.

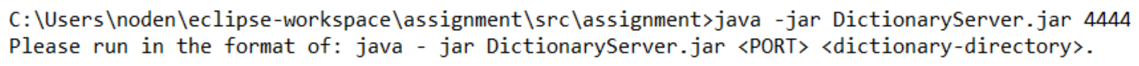
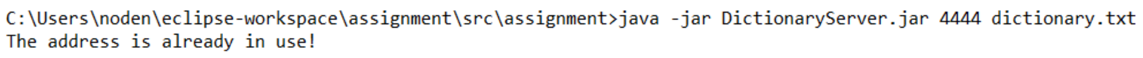

```
C:\Users\noden\eclipse-workspace\assignment\src\assignment>java -jar DictionaryServer.jar 4444
Please run in the format of: java - jar DictionaryServer.jar <PORT> <dictionary-directory>.
```

Figure 5: Responding to incorrect arguments.

- Network communication error: If the port is already in use or the input address is not reachable, we make the program notify the user about the situation and stop the program from running further.



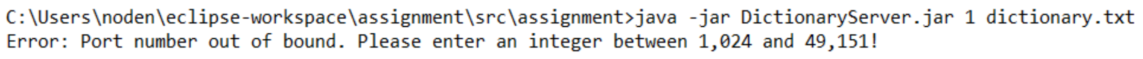
```
C:\Users\noden\eclipse-workspace\assignment\src\assignment>java -jar DictionaryServer.jar 4444 dictionary.txt
The address is already in use!
```

Figure 6: Responding to network error.

- I/O error: We do not consider non-existent dictionary path to be an error. Instead, we create a new dictionary file with a trivial entry to avoid dereferencing of NULL pointer. On the other hand, if there is a problem reading or writing a file, we make the program throw an I/O exception.

Additionally, we catch the following errors:

- Port index out of bound: If the user enters a port number outside of the range (i.e. 1,024 to 40,151), we notify the user that the inputted port number is invalid and remind s/he the valid range of port number. Doing so will prevent the user from accidentally using any predefined ports or any invalid ports.



```
C:\Users\noden\eclipse-workspace\assignment\src\assignment>java -jar DictionaryServer.jar 1 dictionary.txt
Error: Port number out of bound. Please enter an integer between 1,024 and 49,151!
```

Figure 7: Responding to invalid port number.

- Corrupt message error (server): Even though our implementation, by default would not cause any invalid operation to take place, we prevent negative consequences from corrupt data. We do so by checking the format of every input message from clients to make sure it is of the format *operation-word-definition*; fewer or more information is not allowed. We also check that the operation is one of the standard operations, but we have no way to check that the received word or definition is correct. If the received message fails to satisfy any of the conditions listed above, we notify the client about the message being corrupted.

4 Creativity Elements

4.1 Quick Search from History

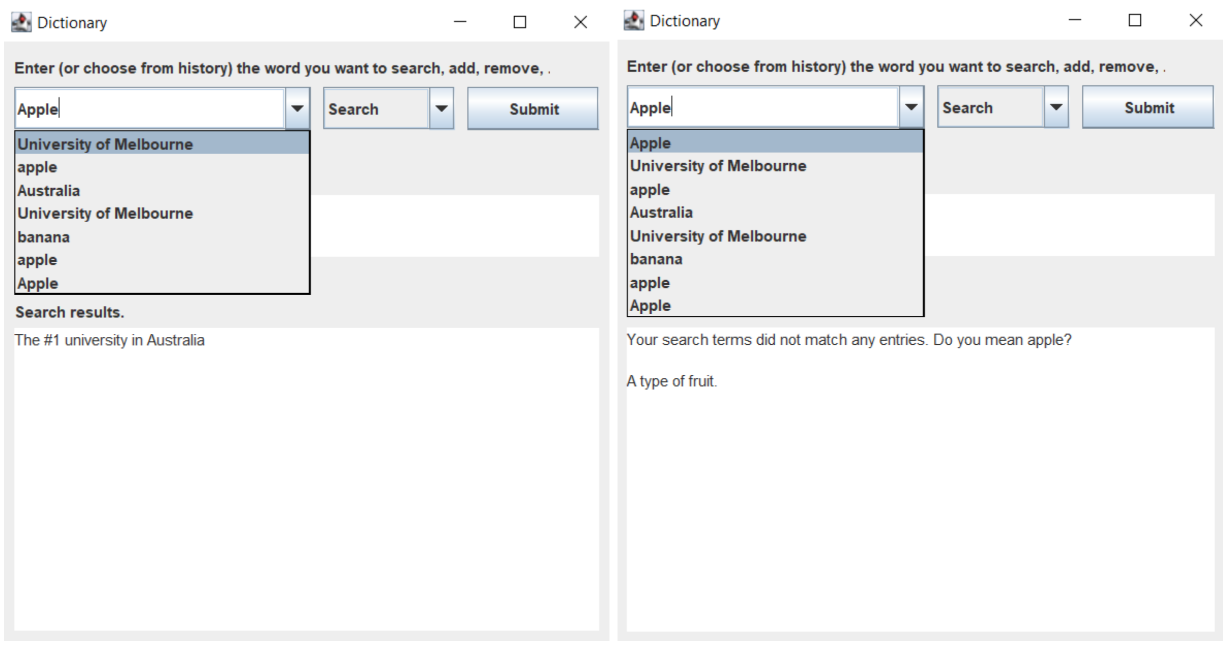
We implemented a queue-like structure on the client end to keep track of the client's search (or add, remove, update) history. The client has the option to choose from a scroll-down list to choose the word s/he would like to work on instead of typing the whole word all over again. The scroll-down list will show the search history in the last-search first-show order.

4.2 Spelling Correction

Misspelling or typo is a commonly happened mistake. I implemented a brief spelling correction algorithm on the server end such that the server may still provide the desired response to the client even if the user made a spelling mistake. The algorithm is based on the notion of edit-distance, and we set the tolerable difference in edit-distance to be 1. If the server program cannot find the exact word matching the client input, it will attempt to find a word that is within 1 edit-distance away from the input word, and append its definition to the word-not-found message. If there is no such word, the server will simply return the word-not-found message. For instance, if the client wants to look up the definition of the word "apple" but accidentally typed in "appple", the server will return the correct definition of "apple" instead of asking the client to make a separate new query. This will also allow case-insensitive words to be located correctly.

5 Conclusion

In this project, we implemented a dictionary application which can process requests from multiple clients using the thread-per-request client-server architecture. We ensured the application is fully functional and safe. We aim to further improve the performance of our dictionary application in the future from several

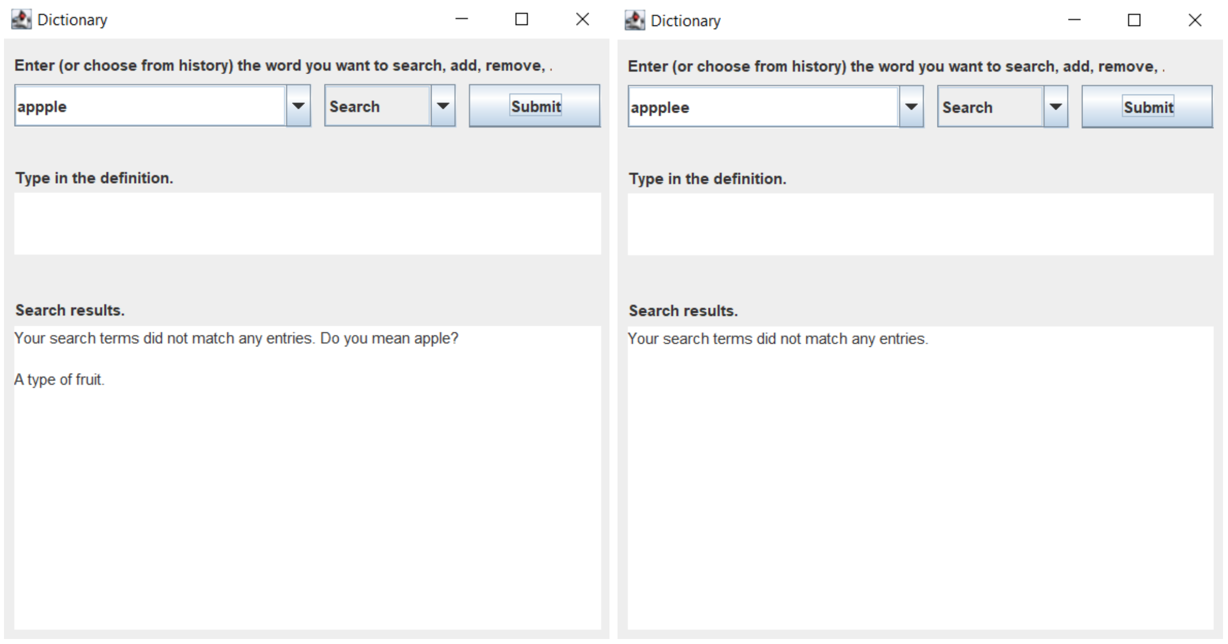


The #1 university in Australia

Your search terms did not match any entries. Do you mean apple?

A type of fruit.

Figure 8: History functionality.


Dictionary
Enter (or choose from history) the word you want to search, add, remove, .
appple
Search
Submit
Type in the definition.
Search results.
Dictionary
Enter (or choose from history) the word you want to search, add, remove, .
appplee
Search
Submit
Type in the definition.
Search results.

Your search terms did not match any entries. Do you mean apple?

A type of fruit.

Your search terms did not match any entries.

Figure 9: Spelling correction.

aspects. For example, we can use a better data structure and some data-driven ways to implement spelling correction which will cover more types of spelling error in a more timely manner. We can also build a GUI for the server to control the connection/disconnection of multiple ports simultaneously without the need to launch a separate server program or a complete shut down of the server program. We also aim to implement Google-search like auto-complete functionality with minimum use of computational resources.