COMP90015 Distributed Systems Assignment 1

Name: Hongzhi Fu Student ID: 1058170

1. Introduction

1.1 Socket

In a network system, there are multiple servers and clients connecting with each other via some protocol, which involves the most two common protocols in transport layer, TCP and UDP. TCP, also called transport control protocol, is a connection-oriented protocol, provides reliable data flow, and have mechanisms to handle out-of-order and replicated data segmentation. Because of reliability of TCP, it can be applied to transfer a secret file, remote terminal access, etc. UDP is connectionless protocol that only transports several independent datagrams, which is very fast, but it does not guarantee the arrival and sequence of those packets. UDP also has many applications, such as video streaming, DNS service, etc. However, network protocols typically have a very complex structure. To address it, a network socket provides an interface that two entities can communicate with each other. With the encapsulation of object-oriented programming, we can create a simple, easy-to-manage and efficient network application. In TCP, we can communicate with a server via binding server's host name and port number that specifies which process we need to access.

1.2 Multithreading

In the real world, one server serving only one client machine is not applicable and sustainable. Instead, multiple clients will connect, send requests to a server concurrently, which requires server running multiple programs concurrently along with consistency. Multithreading is one technology that supports multiple clients accessing shared resources simultaneously. The model of multithreading server has two ports: one is for listening the upcoming client that requests for connection, another is for data transfer or providing other services. Initially, the server creates a socket which binds its hostname and port number, then waits for connection from clients. For each time client sends a connection request, the server will create a thread with a unique port number for further communication while keeping the original port number alive for accepting other client's connection request. With multithreading, clients are able to acquire, add or update shared resources concurrently. While in some situation, in particular, bank account information retrieval, concurrent operation will cause a big problem, so we need to introduce synchronization to handle each concurrent operation.

1.3 Problem Context

The problem context in this task is to build a multithreaded dictionary server that has functionality of accepting the incoming connection from clients. After that, server will create a thread for serving the client. The functions of dictionary server include that provide the meaning of a given word (query), add a new word-meaning pair into the server (add), remove a new word-meaning pair from the server (remove), and alter the meaning of a given word (update). All of the operations support concurrence, which means client 1 can query a word "dictionary" while the same time client 2 can update the meaning of "dictionary" without losing consistency. With regard to client side, it requires us to design a graphical user interface (GUI) that user can interact with the server side. It includes connect/disconnect to server, query a word meaning, add a word meaning, remove a word meaning and update a word meaning. The diagram of dictionary server-client is shown in Figure 1.

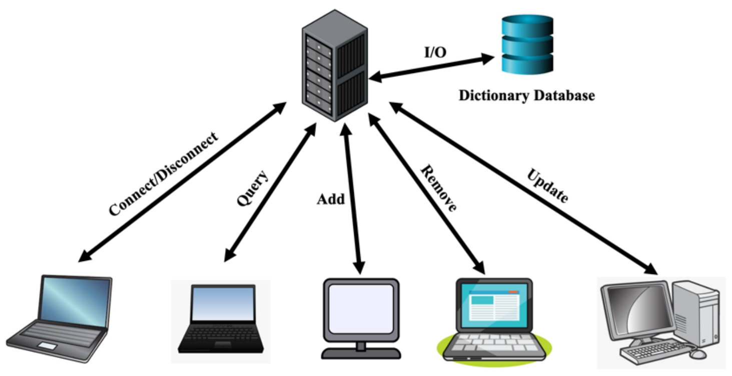


Figure 1. Overview of Dictionary Server-Client Model

2. Model Description

2.1 Server

There are two main components in a dictionary server. In Java, we use `DictionaryServer` class to define an instance of a dictionary server. The `accept` method is responsible to keep listening the incoming connection requests from client side. Before starting the server, we need to specify the port number the server will listen to, and the file name of external dictionary database. Then the server will load the initial server database into main memory as shared resources. Here the format of dictionary database is JSON, so we use Java standard JSON library to do I/O operation. After loading the dictionary file, we convert JSON file into a hashmap, where each word is the key and the corresponding meaning is the value. The next procedure is to initialize a server socket by binding the port number and the protocol we used is TCP, which means client must establish a connection with server before exchanging messages. When a client manages to connect with the server, the server will create a thread to serve this client, i.e., respond client's requests. In `serveClient` method, we will first initialize reader and writer for communication with peer. For better performance, we will use `BufferReader` and `BufferWriter` to buffer incoming and outcoming information and receive or send them at the proper time. Then we will use `BufferReader.readLine()` to read which mode the user will perform. No matter which mode the user chooses, the server will prompt user to input a valid string and if there are any invalid input, it will inform the reason and reprompt the user to input again. For example, when user chooses query mode, the server will send "Please input a word you want to query" to user. When an error occurs, e.g. there is no such word in the dictionary, it will send "Whops! Key xxx is not in the dictionary". In addition, dictionary server can handle different exceptions, such as `IOException` (buffer overflow), `UnknownHostException` (client disconnection). When a client closed connection, the server will print "client closed connection" on the terminal. Server also supports synchronization, which serializes concurrent operation from clients.

2.2 Client

The client takes hostname and port number as arguments and establish a connection with server. If connection is successful, user can choose mode to perform desired operations. Otherwise, the client will tell user the server is not reachable. In Java, the class name is `DictionaryClient`, with a bunch of API to user interface: constructor, which binds hostname and port number; `connect()`, which establishes connection and initializes input and output buffer; `disconnect()`, which closes connection; `status()`, which return whether the client is online; `write()`, which sends request to server; `read()`/`readAll()`, which reads a single line or multiple lines of data. All operations above can handle different unexpected exceptions (e.g. `UnknownHostException`, `SocketException`, `IOException`, etc.)

2.3 GUI

Client also provides graphical user interface (GUI), which interacts with client socket. The main interface is shown in Figure 2.

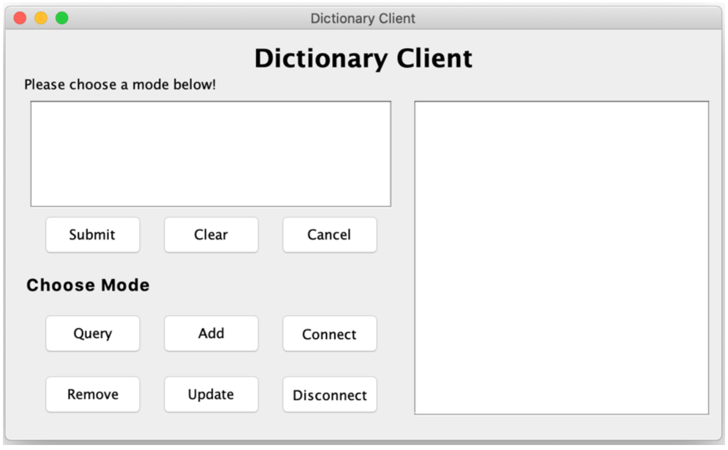


Figure 2. Main Interface of Client GUI

In this user interface, user can choose a mode at the bottom of the left side and input a string at the upper of the left side. The query result will be shown on the right text area block and the message that prompts the user to do operation will be presented at the top of the text box. In this case, "Please choose a mode below!" is the prompted message.

2.4 UML Diagram

Figure 3 illustrates the UML class diagram.

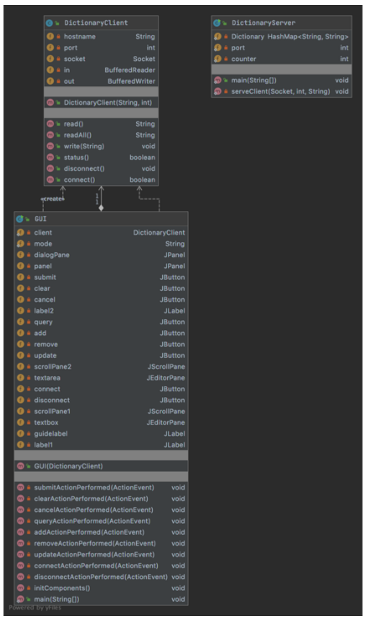
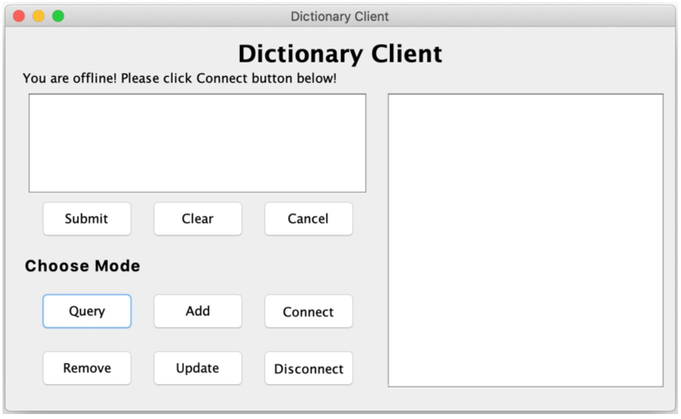


Figure 3. UML Diagram of All Classes and Interaction

3. Excellence

3.1 Error Notification

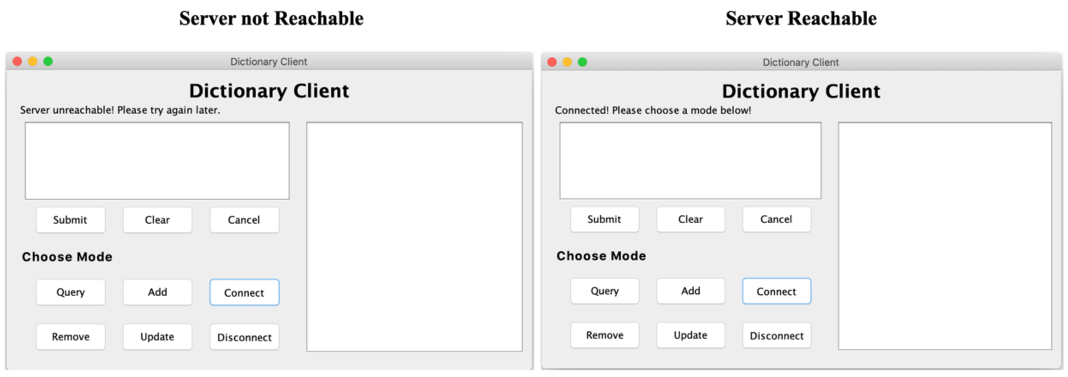
There are three main categories of error notification: One is offline error notification. When the client is offline, i.e., disconnected with the server, if the user tries to perform any operations, the system will inform the user that you are offline and prompt user to click Connection button to connect to the server before doing other operations. Figure 4 gives an example of this case.



You are offline! Please click Connect button below!

Figure 4. User Interface when Clicking Query Button and the Client is Offline

In such case that server is not reachable, which means the server has not been started. Clicking the connection button will result in showing a message "Server unreachable! Please try again later!" to GUI. Conversely, if the server is not closed, clicking connection button will give user a message of "Connected! Please choose a mode below!". Figure 5 illustrates user interface when the server is closed and not closed respectively.


Server not Reachable

Dictionary Client

Server unreachable! Please try again later.

Server Reachable

Dictionary Client

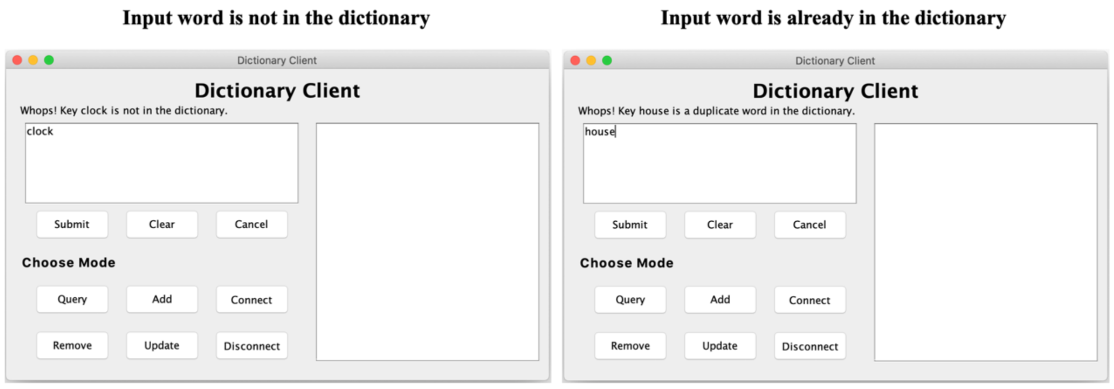
Connected! Please choose a mode below!

Figure 5. User Interface when Clicking Connection Button and the Server is (not) reachable

The last case is normal error notification. For query, remove or update mode, if there are no such word in the database, the client will send a message to user that the word is not in the dictionary. For add, mode and update mode, if the word we want to add is already in the database, the client will send a message to user that the word is a duplicate word. Figure 6 shows what the user interface would be like when the above four cases occur, respectively.

3.2 Pros and Cons of Design

The model we presented has several advantages. The first is manageability due to the separation of Client and GUI. Client provides some functions to GUI, so that when developing the GUI, it is unnecessary for us to consider how client works. Instead, what we should do is to invoke functions defined in `DictionaryClient` class. The second advantage is user-friendliness because of error notification. The client will prompt user to perform operations in different cases.


Input word is not in the dictionary

Dictionary Client

Whops! Key clock is not in the dictionary.

clock

Input word is already in the dictionary

Dictionary Client

Whops! Key house is a duplicate word in the dictionary.

house

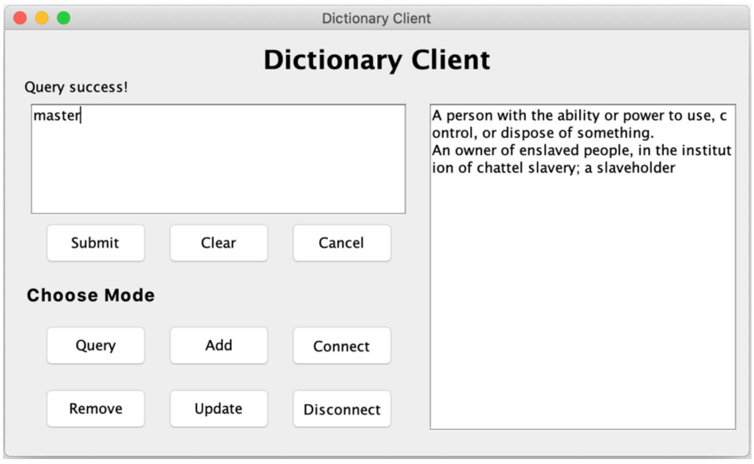
Figure 6. User Interface when Input Word is (not) in the Database

However, efficiency is one of the disadvantages in our system. Since the server associates a thread with each connection, responses client's requests and destroys the thread when client closes the connection. If the user does not close the connection, the thread in the server will never destroy the thread. Thus, it is possible that server is overwhelmed with idle threads, in which the thread keeps running, but never does any job.

4. Creativity

4.1 Multiple Lines of Input and Output

A word may have multiple meanings, which indicates we must separate those different meanings into several lines. Thus, our buffer reader and buffer writer must support transferring multiple lines of data. In Figure 7, we present an example of querying a word (master) with different meaning. The text area at the right side shows the meaning of "master" is listed in two lines.



Query success!

master

A person with the ability or power to use, c
ontrol, or dispose of something.
An owner of enslaved people, in the institut
ion of chattel slavery; a slaveholder

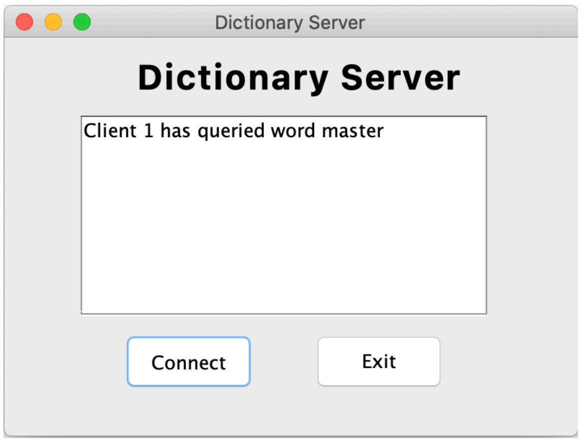
Figure 7. Displaying of Multiple Lines of a Word.

4.2 Automatic Disconnection

To address idle thread problem, where a client may keep connected with the server but does not perform any operations, so that the server will waste a lot of time running the thread. We set a predefined timer that if there are not any operations done by user within a period of time (here we set one minute), the client will shut down the connection automatically. When the user comes back, he/she needs to reconnect with the server by clicking the Connect button. If the user clicks other button, client will prompt user to do that before establishing connection.

4.3 Server Management

To better manage the server side, we also created a server GUI. You can click connect button to start the server and click exit button to close the server connection. There is a message box that shows the message when a critical operation has been performed, i.e., client closes the connection, client remove a word from the dictionary, etc. Figure 8 gives an example message shown in textbox in server GUI.



Client 1 has queried word master

Figure 8. Example Message in Textbox