COMP90015: Distributed Systems – Assignment 1 Report

Name: Tong He **Student id: 867488** **Tutor name: Shashikant Ilager**

1. Introduction

Internet-enabled applications are ubiquitous in the so-call information era. Socket and Thread have emerged as two fundamental technologies for creating portable, efficient, and maintainable large and complex Internet applications. This project is to leverage the two technologies to design and implement a multi-threaded server to provide basic dictionary functions to concurrent clients.

2. Architecture

This project use client-server model, where one program requests a communication called client process and other who is listening the specific bounded port to waiting for a connection establishment called server process. The socket provides an interface at transport layer by which server/client's interaction is very much similar to performing file I/O, namely, the datastreams in file I/O operation are also applicable to socket-based I/O.

3. Protocol and Socket

TCP and UDP can be used for client-server connection at transportation level, using *ports* to map incoming data to a computer process. This program uses TCP protocol for connection-oriented communication as provides a more reliable flow of data. Java provides TCP socket and UDP socket. In TCP socket there is a Socket class for client socket and a Serversocket for server socket, both in java.net package. The Client part in the program written using Socket class have following lifecycle:

1) Constructer Socket (String host, int port) is called to create a stream socket. (in the test, host = 192.168.1.4; port = 1234)

2) Connect to the specific port of the host(server)

3) Client and server follow TCP (handshake before sending data) to exchange data via DataInputStream and DataOutPutStream class.

4) Close the connection.

About “Mainserver” class in the program which is a multi-thread server using ServerSocket class, the lifecycle is as follows:

1) Constructor ServerSocket(int port) is called to create a ServerSocket on a specific port(in the test, port = 1234).

2) accept () is called in while(true) body to continually listen the port until there appears a client’s request.

3) Server creates a thread to the connection.

4) The thread starts to DataInputStream to receive client’s data and DataOutputStream to send data to client.

5) The server goes back to second step and waits for a new connection.

Java also provides UDP socket including class DatagramPacket and DatagramSocket. Both client and server use the former’s send () and receive () methods to exchange data, uses the latter to encapsulate data. This program only uses TCP socket.

4. Implementation of functional requirements

A client connects to the server and the server creates a thread for the following interactions. Inputstream, outputstream, and three functions (search, add, delete a word) are implemented in ServerThread class. The concrete implementations are as follows:

Query the meanings of a word: In run method of ServerThread class, DataInputStream class is used to read an input from client. The format of Input is “Operation(space)word”. Use StringTokenizer class to get the first token. If the token equals string “Search”, get the word (next token) and call containWord method of

Mydictionary class to verify whether the dictionary contains the word(return meanings) or not(send “no such word” to client).

Add a word: Front steps are same as query except operation equals “Add”. If the word has existed, sends the reminder to client, else add meanings to the dictionary.

Remove a word: same as above except operation equals “Remove”. If the word exists, delete it in the dictionary and send “delete successfully” to the client, else send “not exist “to the client.

Process data in the dictionary: use hashmap to store data: word as key, and meanings as value. Load contents from designated file. When the server is closed, upload the file via hashmap.

5. Class

5.1 Class types

There are two classes in Client part:

Client_UI: offering graphical interface handling input and output with users.

Client: contain the core functions of the client: connecting to the server, sending service requests.

Server part contains 4 classes:

Sever_UI: offering graphical interface to input port number of dictionary name to start the server and return the server IP.

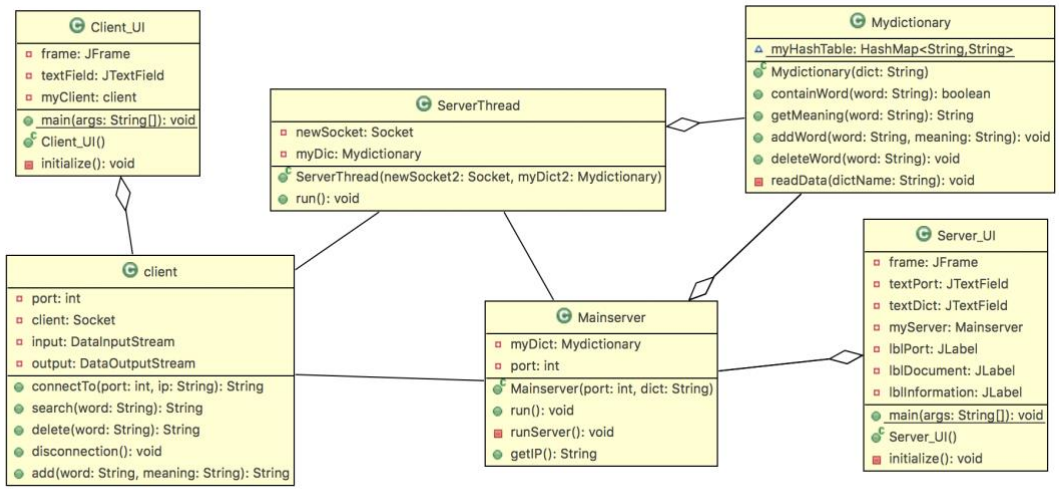
Mydictionary: handles dictionary-based operations: add, delete, and find. Only one unique object is generated corresponding to the current dictionary.

Mainserver: The server main thread listens on the specific port and generates a new thread for each connection request, creates an instance of Mydictionary and binds it to each thread.

ServerThread:

Connect the client and the dictionary: parse the client message as the corresponding request and hand it to the dictionary for processing, and finally return the result to the client.

5.2 UML



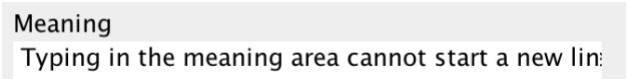
The server-side user establishes the Mainserver object through Server_UI, and the Mainserver establishes the Mydictionary object and waits for the connection. The client user establishes a client object through Client_UI, and the client sends a request. The Mainserver accepts the request and transfers the service to the newly created ServerTread for execution.

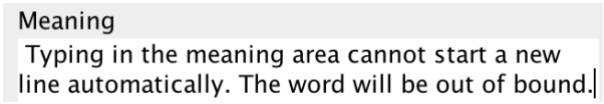
6. Analysis

According to test results, there are some details that should be improved, some of which did some not.

6.1. Bug: Typing in the meaning area cannot start a new line automatically. The word will be out of bound.

e.g. test result:



It shoud be 

Fix: add two lines:

```
textArea.setLineWrap(true);
```

// Sets the line-wrapping policy of the text area

```
textArea.setWrapStyleWord(true);
```

// Use the style of wrapping to ensure the integrity of the texts.

6.2 Multithread Concurrency

Sharing dictionary sources by threads may cause the data to be inaccurate and conflict with each other, for example, a thread which is reading the word may read the interim information while another thread is adding the word meanings. Thus, it is necessary to add a synchronization lock to avoid being called by other threads before the thread completes the operation. This program first tried to add "synchronized" in method header just in two methods of modifying the contents in MyDictionary class. However, one thread can still read content when the other is modifying the content. Thus, containWord(), getMeaning(), addWord() and deleteWord(), the four methods are added "synchronized" in method header. This may manage the concurrency, however, even other threads cannot read the word when a thread just reads the word. It needs to be improved.