Design and Implementation of a Dictionary Server

Zhengtian Lu 1025107

April 6, 2023

Introduction

This task is to build the communication architecture between the server and the client, and realize the function of adding, deleting, modifying and checking the dictionary. This article introduces how the author uses TCP to realize the communication between the client and the server. In terms of troubleshooting, client and server issues were well managed and the system didn't crash with common errors like connection, bind, and null pointer exceptions. At the same time, it also fully reflects the mistakes that have occurred so that they can be corrected. Also uses a special string to tell the server and client what to do at the moment. This report also includes critical thinking on class design and the whole system. It also point out the disadvantage of the project and Improvement direction.

Components

Server, client and user interface are the three main components of the system. The server side handles sending requests from clients, manipulating dictionary files as they are read or written. The client is responsible for creating the client UI that the user interacts with. If the user provides accurate information, the request is forwarded to the server for processing, and the result is returned to the client.

Class Design

In order to ensure the reliability of communication, the author has done the following critical thinking and research in TCP and UDP [1]. The reason TCP is more reliable is that it uses a three-way handshake. If the connection fails, transmission will stop and no packet loss will occur. UDP, on the other hand, is fundamentally unreliable by design. Its design aims to consistently transmit packets to one or more receiving clients without the need to wait for a "listening" or acknowledgement condition [1]. The architecture is shown below (Figures 1 and 2):

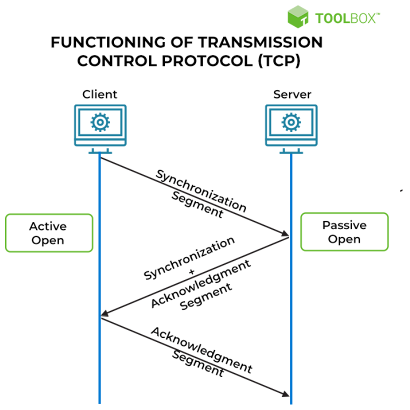

TOOLBOX
FUNCTIONING OF TRANSMISSION CONTROL PROTOCOL (TCP)
Client
Server
Synchronization Segment
Active Open
Passive Open
Synchronization + Acknowledgment Segment
Acknowledgment Segment

Figure 1: architecture of TCP [1]

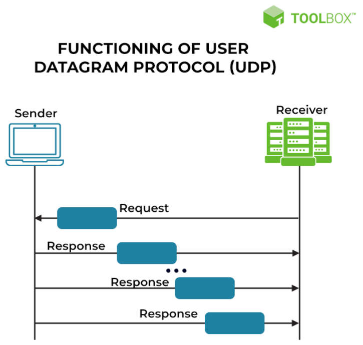


Figure 2: architecture of UDP [1]

In this project I also create the five classes to finish the Key features:

- **DicServer:** The main class responsible for handling server sockets and accepting incoming client connections. When the server shuts down, it also reads the templates from the file and writes them back to the file.

- **Connection:** Thread class that manages server-to-single-client communication. The request from the client is analysed, and the relevant method is then called to carry out the required operation on the dictionary.

- **DictionaryClient:** This is the dictionary application client. Connects to the server at localhost over port default 8080 using sockets or using the user setting port. Display a user error message and exit the program if the connection fails. The main task of this class is to read messages sent by the server and display them to the user. To accomplish this, a while loop is used that continuously monitors messages from the server. When a message is received it is checked whether it is a status update or a response to a request. Given a query response, the meaning of the query expression is displayed in her GUI text box. For status update messages, the message is displayed in the status area of the GUI. Read messages from a socket's input stream using a BufferedReader to read messages from the server. A BufferedWriter is also used to write messages to the socket's output stream as they are sent to the server. UTF-8 encoded strings are used to send and receive messages.

- **ClientUI:**Clients can access a graphical user interface thanks to the ClientUI class. It has text boxes where users may enter words and their definitions, buttons for adding, deleting, updating, and looking up terms in the dictionary, and text spaces where users can view dictionary and status updates. Additionally, this class manages user input and makes the proper request to the server to carry out the required dictionary action. Send the request using the DictionaryClient class' BufferedWriter.

- **CustomThreadPool:**This code implements a custom thread pool which mainly includes: BlockingQueue taskQueue: A task queue used to store tasks to run. Thread[] workerThreads: Thread array used to store worker threads. CustomThreadPool(int numberOfThreads): Initialize constructor, task queue and worker threads. submit(Runnable task): Submits a task to the task queue. shutdown(): Closes the thread pool and suspends worker threads. WorkerThread extends Thread: An inner static class, a worker thread class that retrieves tasks from the task queue and executes them. The main function of a custom thread pool is to control the number of threads, avoid system resource exhaustion due to too many threads, and improve system efficiency and stability. The thread pool uses the LinkedBlockingQueue blocking queue to store tasks. When the task queue is full, the thread that continues to add tasks blocks and waits for tasks to be dequeued and executed before adding new tasks. At the same time, each worker thread in the thread pool picks up a task from the task queue and executes it. After execution, it continues to accept tasks from the queue until the thread pool is closed.

Interaction Diagram

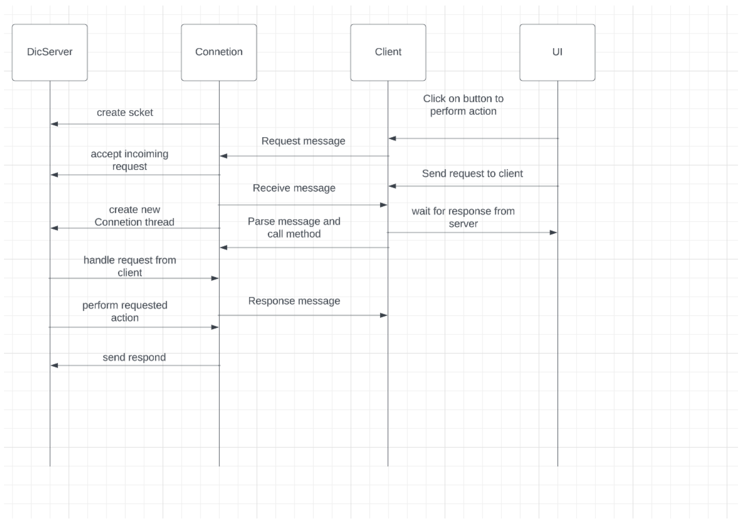


Figure 3: Interaction Diagram

Conclusion and Critical Analysis

This project implements dictionary functionality including add, update, delete, and query. Using Concurrent-Hash-Map is suitable for use in high concurrency environments. At the same time, it is also thread-safe. Because different threads can read and write different segments at the same time, it can support concurrent reads and writes more efficiently, and query efficiency can reach O(1). The same project also has flaws. For example, this project can add functions such as login and logout functions to your program, or it can add functions to server to detect if the request is malicious and prevent malicious reads and writes. it can also add The log engine is used to record server activity and anomaly information. Overall, as a low concurrency software, this project provides basic functionality and covers most situations, but in the future, if it is used in a highly concurrency complex environment, I would recommend using this software can spare more room for better expansion.

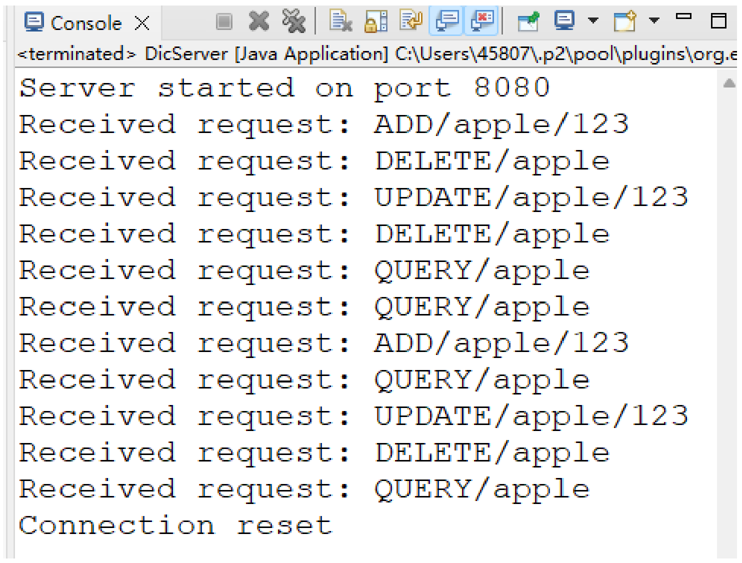
Excellence

- **server load:** When using the add and update operations, if the user enters only one of the meaning and word, or if the word contains spaces, this programme will not send the request back to the server for processing, but will instead display an error window. This work will result in To reduce the burden on the server and reduce read and write operations, the server does not have to handle incorrect requests.

- **port resource:** When the server is not started, the client will pop up a window displaying "Unable to connect to server. Please try again later." Then exit the program. When exiting the client without exiting the server, the program will print "Connection reset it means one clinet is closed" on the terminal. However, the server will continue to work for other users, it will not effect other clients, the other clinets can continue to do operations on the software. When the server shut down during the using, the client will be killed and terminal will print out "server is killed please retry", the clients can restart the server and client.

- **port error:** If no port number is specified, the default port 8080 will be used. If a port is chosen, the port chosen by the user is used. If an incorrect port number is entered, the command line will display "Invalid port number: " and a popup will appear. Windows is very friendly to both command line execution programme users and non-command line execution programme users.

- **print operation:**All operations will be printed out in the terminal, allowing you to see what the user did while using the software and what kind of changes were made to the file, allowing the operation to be recorded and traced back to the operation that caused the problem.


Console
<terminated> DicServer [Java Application] C:\Users\45807\.p2\pool\plugins\org.e

```
Server started on port 8080
Received request: ADD/apple/123
Received request: DELETE/apple
Received request: UPDATE/apple/123
Received request: DELETE/apple
Received request: QUERY/apple
Received request: QUERY/apple
Received request: ADD/apple/123
Received request: QUERY/apple
Received request: UPDATE/apple/123
Received request: DELETE/apple
Received request: QUERY/apple
Connection reset
```

Figure 4: Interaction Diagram

Creative:

In order to better control the number and life cycle of threads, improve the performance and stability of the server. It is more suitable for the stability of the program in the case of high concurrency, avoiding too many threads and improving the robustness of the server.number of threads is setted to 10.

```
private static final int NUMBER_OF_THREADS = 10;
private static CustomThreadPool threadPool;
```

when server close the threadPool also be closed:

```
threadPool.shutdown();
```

References

[1] Spiceworks. *TCP vs UDP: Understanding the Difference*. Accessed: 2023-04-02. 2023. URL: https://www.spiceworks.com/tech/networking/articles/tcp-vs-udp/#:~:text=Transmission%20control%20protocol%20(TCP)%20and,UDP%20prioritizes%20speed%20and%20efficiency.