Design and Implementation of Multi-threaded Dictionary Server

Author: FuQuan Gao, 1648979 Date: April, 2025

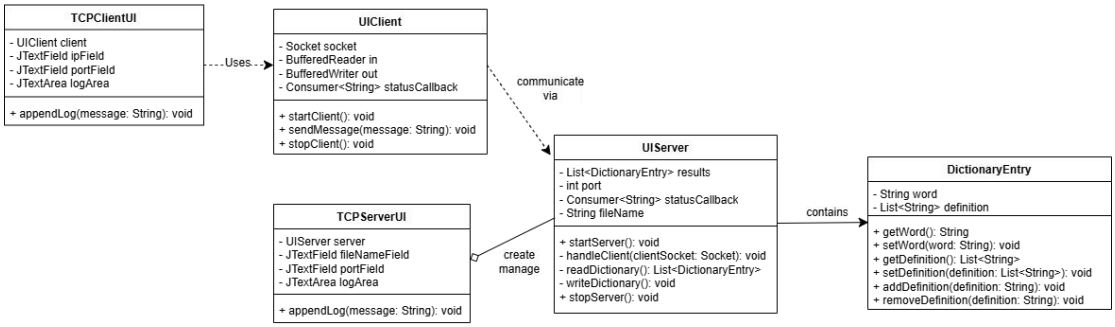
1. Introduction:

In the lecture that the distribution system has been discussed and there are 2 fundamental technologies has been mentioned which is Sockets and Threads.

This assignment will implement a dictionary system which has a multi-threaded server that can handle concurrent clients to query or modify the meaning of an existing word, add or remove a word, add or remove the meaning of an existing word, save all the changes to dictionary file and display all words and theirs meanings that current dictionary has.

2. System Architecture and Components:

2.1 System architecture:



The dictionary use a client-server architecture:

2.2 Components:

Client:

Through GUI, it communicate with the user to send the request like query the meaning or adding words.

It just pass information, does not handled data logic apart from not allowing user enter nothing.

Server:

As the provider of the service, server will handle all the features.

Listening to the clients connection, with the concurrency and the data consistency.

Protocol:

TCP connection is used between server and client. It is a reliable communication that meet the requirement of the assignment.

The communication between client and server is text-based.

3. Client architecture:

3.1 Main component:

Graphic user interface:

Implement by the TCPClientUI Class, Swing framework is used to providing following features:

Text field for input the Server IP, the port number and the message that

send to the server.

Button to start the connection, stop the connection and send the message to the server.

Display the log information, which including the connection status, menu option from server and error information.

Network communication:

Implement by the UIClient Class, which has for following responsibilities:

Establish the connection with the server using TCP protocol.

Sending request to the server.

Listen to the sever and display the respond form the sever.

Data validation:

The client will provide the preliminary verification on user input, which is not allowed user to make empty input.

Error handle:

Handle the server connection error, including timeout, port not exist, server IP and port format error.

3.2 Data stream:

User input -> GUI -> Encapsulate request -> Send to server -> Receive response -> Update GUI display

4. Server architecture:

4.1 Main component:

Network listening module:

Implement by the UIServer Class, duty including:

Start the sever(serverSocket) at the appoint port.

Accept the connection from the client, for each connection create a new thread.

Business logic processing:

Core feature implement by the method in UIServer Class:

Dictionary management: queryMeaning(), addWord(), removeWord() etc.

Thread synchronization: using synchronized keyword to protect the shared data(List<DictionaryEntry>).

Data persistence:

Implement by readDictionary() and writeDictionary() methods, read and write dictionary data.

Error handle:

Catch and handle exceptions:

Network exceptions: Client disconnected (IOException), Connection time out exception (SocketTimeoutException).

Data exception: JSON analysis error (JsonMappingException), file I/O error.

Input validation: for all unexpected input format, respond with a message to indicate the input is not valid.

4.2 Data Stream:

Client request -> server listening thread -> distribute to handling thread -> execute the business logic -> return to the respond -> update the data file

5. Thread modeling(Concurrency):

Using Thread-per-connection:

Implementation mode:

Every time when server receives a new connection of client, a new thread will be created to handle the connection.

6. Communication protocol design:

Message format: Plain text.

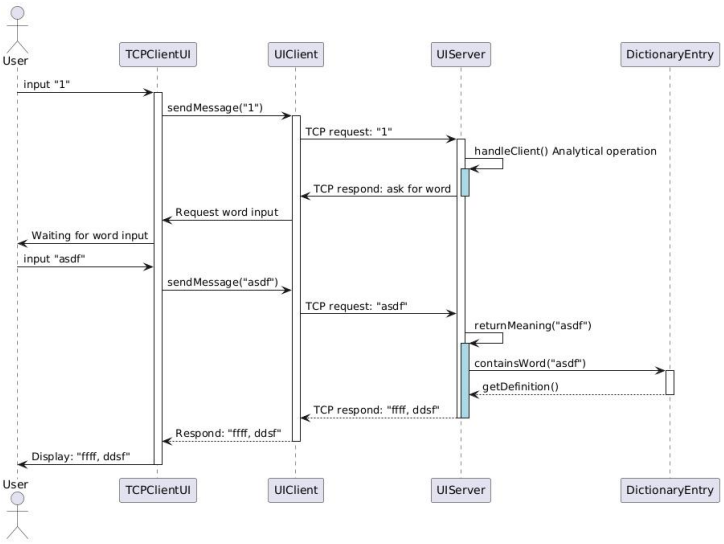
Transport method:

Using BufferedReader / BufferedWriter based on TCP streaming.

All message end with a string: “\nEND\n” to implement simple message boundary processing.

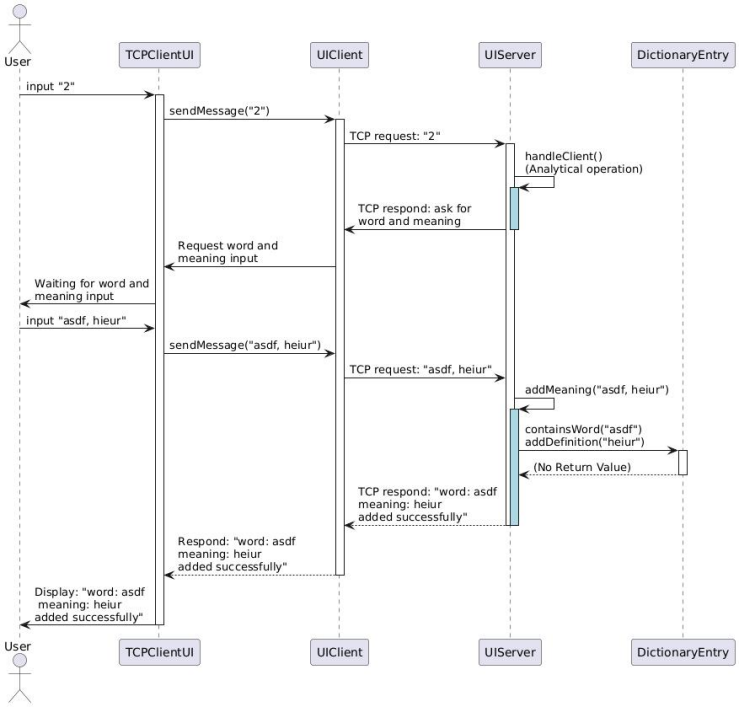
7. Key interaction flows(Sequence Diagram):

7.1 Query meaning:



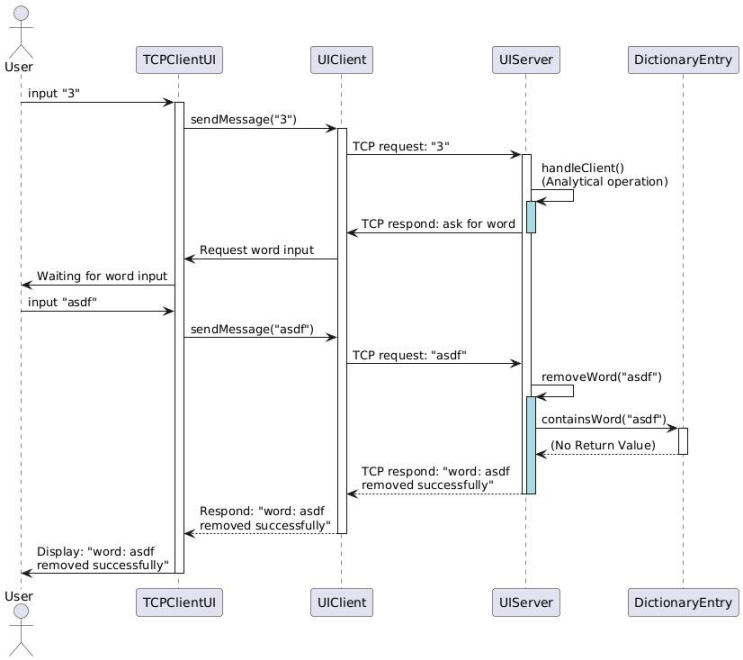
User types in 1 through the client GUI to the client and the client sends the message to the server. The server responds to the client and waits for the next input. The client displays the message to the client GUI. The user types in the word to query. The client GUI sends the word to the server. The server looks for the word in dictionary, gets the definition and sends it back to the server. The server then responds to the client. The client displays the definition to the GUI for the user.

7.2 Add word:



User types in 2 through the client GUI to the client and the client sends the message to the server. The server responds to the client and waits for the next input. The client displays the message to the client GUI. The user types in the word and meaning to add. The client GUI sends the new word and meaning to the server through the client. The server sends the word and the meaning to the dictionary. Dictionary adds the new word and meaning, then the server returns the successful message. The client sends the message to GUI for the user.

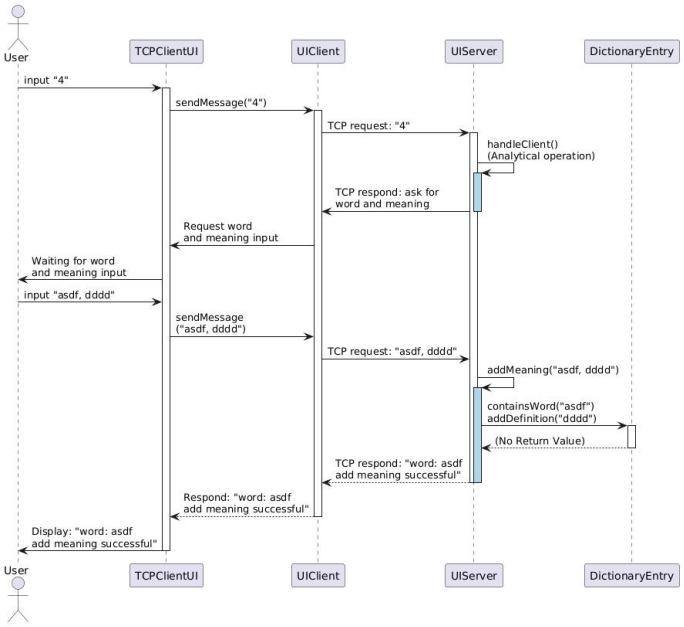
7.3 Remove word:



User types in 3 through the client GUI to the client and the client sends the message to the server. The server responds to the client and wait for the next input. The client displays the message to the client GUI. The user types in the word to remove. The client GUI sends the

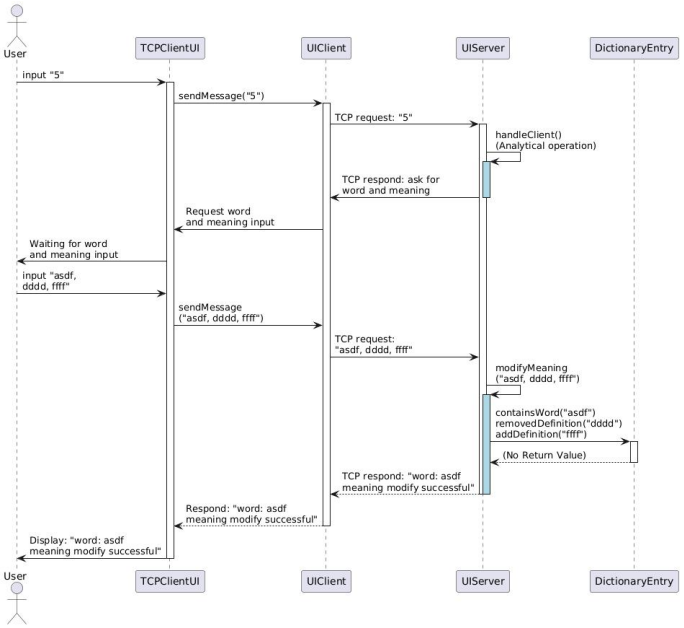
word to the server. The server looks for the word in dictionary and the dictionary removes it. Then the server responds the word removed successfully message to the client. The client displays to the GUI for the user.

7.4 Adding additional meaning to an existing word:



User types in 4 through the client GUI to the client and the client sends the message to the server. The server responds to the client and waits for the next input. The client displays the message to the client GUI. The user types in the word and meaning. The client GUI sends the message to the server through the client. The server looks for the word and meaning in the dictionary and the dictionary adds the new meaning to the word. Then the server responds the meaning added successfully message to the client. The client displays the message to the GUI for the user.

7.5 Update an existing meaning of an existing word:



User types in 5 through the client GUI to the client and the client sends the message to the server. The server responds to the client and wait for the next input. The client displays the message to the client GUI. The user types in the word and meaning. The client GUI sends the message to the server. The server looks for the word and meaning in the dictionary. The dictionary replaces the existing meaning with the new meaning. The server responds the meaning modified successfully message to the server through the client. The client displays to the GUI for the user.

8. Error handling:

The program handles all possible error which may appear:

File IO error, File data format error, JSON mapping error, Server socket error, I/O error, port error: input none number, server connection refused, network unreachable, unknown host error, User input unexpected: entry not valid, and so on.

9. Conclusion:

This multi-threaded dictionary server and client system successfully address the core requirements:

The program supports concurrent client operations(query, add words, remove words, add meaning and modify meaning) with real-time data synchronization across clients.

A GUI-Based client architecture ensures user-friendly interaction.

Data persistence is achieved through JSON file read from file/save to file.

Explicit use of TCP sockets and thread-per-connection model aligns with the assignment's foundational requirements.

Proper error handling for network exceptions(connection, timeout), file I/O errors and invalid user input.

10. Critical Analysis:

Design choices and trade-offs:

Thread-per-connection Model:

Strength: simplifies concurrent request handling and isolates client sessions, easy to scale while facing unknown number of clients.

Limitation: when facing high concurrency, it is easy to exhausted the thread resources.

Custom Text-based Protocol:

Strength: easy to debug, lightweight system for design.

Limitation: lack of the structured data validation, increasing the parsing risks.

Synchronized Data Access:

Strength: ensures thread safety for shared dictionary data via synchronized methods.

Limitation: when facing heavy contention, the performance might be bottleneck.

Identified Limitations:

Protocol rigidity: the text format limits the extensibility for the features.

Lack of Encryption: sensitive data is transmitted in plain text.

I/O Overhead: frequent file writes may degrade the performance for large datasets.

Future recommendations:

Performance Optimization:

Replace the thread-per-connection model with a thread pool to manage concurrent connections efficiently.

Implement asynchronous file I/O operations to reduce blocking during data persistence.

Protocol Enhancement:

Using structured protocol(e.g. JSON) for better inter communication and error handling.

Security Improvements:

Integrate encryption(e.g. TLS) for secure client-server communication.

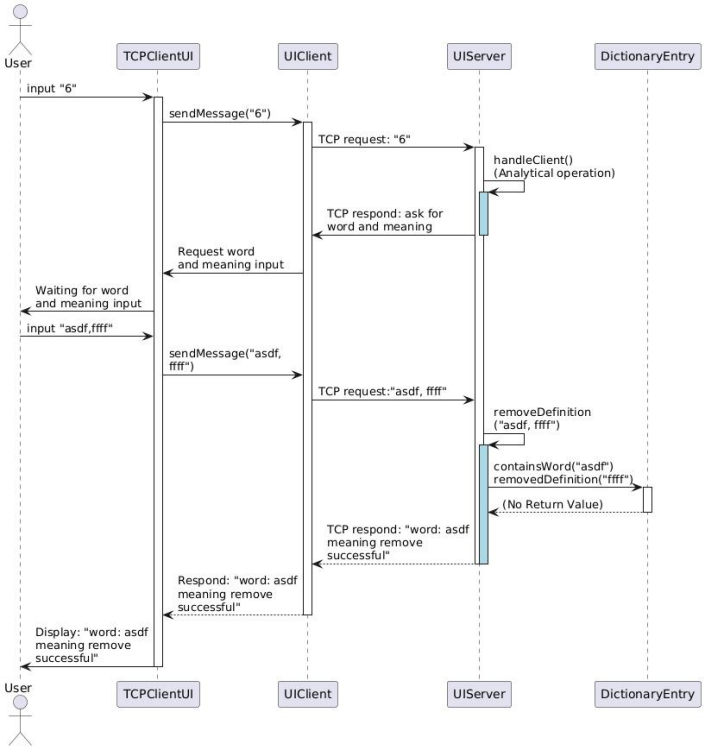
Add user authentication to restrict unauthorized access.

Advanced Features:

Implement versioning control for collaborative editing.

11. Excellence elements:

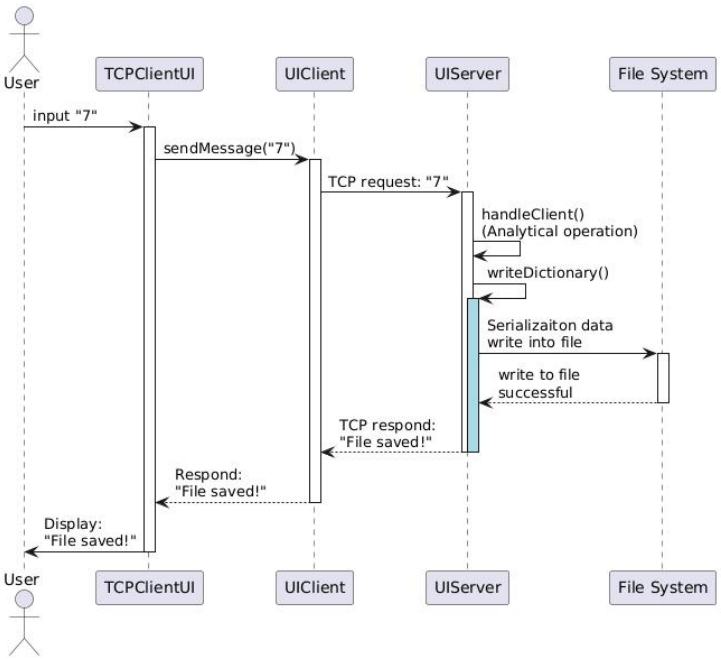
(1) Remove an existing meaning of an existing word:



User types in 6 through the client GUI to the client and the client sends the message to the server. The server responds to the client and wait for the next input. The client displays the message to the client GUI. The user types in the word and meaning. The client GUI sends the message to the server through the client. The server looks for the word in the dictionary. The dictionary finds the word and removes the meaning. The server responds the word meaning remove successfully message to the client. The

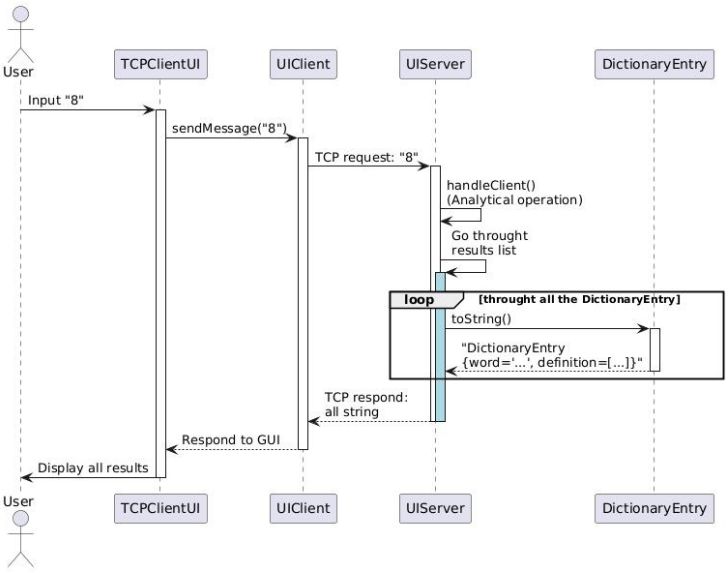
client sends the message to the GUI for the user.

(2) Save to the file:



User types in 7 through the client GUI to the client and the client sends the message to the server. The server handles the client and starts to write to the file with json format serialization. Once the file system responds to the server with successful, the server responds to the client. The client display the message to the GUI for the user.

(3) Print all dictionary:



User types in 8 through the client GUI to the client and the client sends the message to the server. The server handles the client and starts to go through all the results list. The server using loop gets toString from the dictionaryEntry and responds back to the client with all the dictionary data. The client displays all the data to the GUI for the user.

(4) Error handling:
When user types the option or the word, if the user type an unexpected input, the program responds an error message as entry type not valid. Each time the user chooses a function, the server displays the respond message with the expect input format.

12. Creativity Elements:
    (1) Using Socket Factory can easily fit with the scalability. The factory can be customized to suit different kind of needs.
    (2) Unless the user close the client UI, it will always allow the user to try to connect to the server, even through if access fails. The client UI will display the message of the error and reset so the user can retry to connect to the server.
    (3) An Server GUI has implemented. With the GUI, the user can set different port to listen to, open different dictionary file as long as the file match the DictionaryEntry format.