David Xiang

1083691

Distributed Systems

**Problem Context**

* Design and implement mutli-threaded server that contains a dictionary (the shared resource) that multiple users can access over a distributed network.
* Implemnet using the client-server architecture
* The system will follow a client-server architecture in which multiple clients can connect to a (single) multi-threaded server and perform operations concurrently.
* Provide diagram of what this may look like
* The operations that a client can make are the following:
  + Query Word
    - Output: Meaning(s) of the word
    - Error: The client should clearly indicate if the word was not found or if an error occurred. In case of an error, a suitable description of the error should be given to the user.
  + Add Word
    - Output: Status of the operation (e.g., success, duplicate)
    - Error: The user should be informed if any errors occurred while performing the operation.
  + Remove Word
    - Output: Status of the operation (e.g., success, not found)
    - Error: The user should be informed if any errors occurred while performing the operation.
  + Append Meaning
    - Output: Status of the operation (e.g., success, not found)
    - Error: The user should be informed if any errors occurred while performing the operation.
  + Update Meaning
    - Output: Status of the operation (e.g., success, not found)
    - Error: The user should be informed if any errors occurred while performing the operation.
  + Exit

**System Components**

* sockets and threads must be the lowest level of abstraction for network communication and concurrency.
* TCP sockets was used
  + As seen with java serversockets
  + Expand on why TCP is reliable (maybe provide a diagram into how TCP ensure reliable data transfer)
  + If you are using TCP, then the reliability guarantees offered by the protocol are sufficient
* Data format of original dictionary file is CSV.
  + This decision was driven by the ease of processing
  + Users are given instructions on how to correctly input words and meanings to conform to CSV file format
  + Error handling is in-place when user deviate from instructions. \*need to confirm\*
  + When the server is launched, it loads the dictionary data from a file containing the initial list of words and their meanings. This data is maintained in memory in a data structure (Java concurrent.map??)that enables an efficient word search. When words are added or removed, the data structure is updated to reflect the changes. These changes are also saved to the original CSV file such that when the server is closed and activated another time, these changes will be permanent.
    - Expand on why concurren hash map was chosen
      * It is a concrete class that implements the ConcurrentMap interface, providing a thread-safe, concurrent hash map implementation.
      * It uses a segmented locking mechanism to allow multiple threads to access different segments of the map concurrently, improving performance in multi-threaded scenarios.
      * It's the standard, out-of-the-box implementation of a thread-safe hash map in Java.
      * It does not allow null keys or values. It is part of the java.util.concurrent package.
* Chose to go with a worker pool architecture? \*Need to confirm just what is the architecture\*

Concurrency in Action in the Server (Thread-per-connection)

1. **main method** takes port and dictionaryFilePath from the command line
2. It calls **StartServer**, which:
   1. Loads the dictionary into memory.
   2. Sets up a **ServerSocket** listening on the given port.
   3. Accepts incoming client connections **in a loop**.
   4. **For every client**, it creates and starts a **new thread** using ClientHandler.
3. The following code handles acceping multiple clients and uses threads to

// Accept and handle each client connection in a new thread  
while (true) {  
 Socket clientSocket = serverSocket.accept();  
 System.*out*.println("Accepted connection from: " + clientSocket.getInetAddress());  
  
 // Handle client request using a new thread to allow concurrent clients  
 new Thread(new ClientHandler(clientSocket, dictionary)).start();

This loop:

* + Accepts **one connection at a time** (sequentially).
  + But then it **delegates handling** to a new thread, freeing up the loop to accept the **next connection**.
  + Thus, the **server remains responsive** to new clients.

1. This specific line of code shows:

* new Thread(new ClientHandler(clientSocket, dictionary)).start();
* is **where concurrency happens**. Here’s what this means:
* ClientHandler implements Runnable and defines how to handle **each client’s requests**.
* A new **thread** is started for each client — this means:
  + **Multiple clients can be served simultaneously.**
  + They don’t block each other.
  + Your server is **multi-threaded**.

This model allows clients to have persistent connection and be able to make multiple request. A client is always assigned a clientHandler which is its individual thread.

Concurrency in Action in a client’s clientHandler (Worker Pool)

1. Inside each ClientHandler, There is not another layer of concurrency and the reason for this is simple: A client will be assigned to a clientHandler that is created on a new thread. Then clients can perform operations hwoever, a client cant perform multiple operations at the same time. It doesn’t make sense then to have concurrency at this level. After all, a client can query a word or add a word but it shouldn’t be able to do both.
2. All ClientHandler threads share the same Dictionary instance, since the Dictonary instance is instantiated in the DictionaryServer, each new connection is given the same Dictonary instance.

A screen shot of a computer

Description automatically generated

• **Persistent connection**: Great choice — the client can stay connected and send multiple requests over time.

• **Sequential requests**: Since the client can only send one request at a time (i.e., it waits for a response before sending the next), there’s **no need for internal concurrency within ClientHandler**.

• **Dedicated thread per client**: That thread can safely handle all of that client’s requests one-by-one, maintaining the persistent session.

Each client has its own personal attendant (thread), who:

• Stays with them throughout the session,

• Handles one request at a time,

• And closes up shop when the client disconnects or says “EXIT”.

Protecting against concurrency related problems inside Dictionary:

|  |  |  |  |
| --- | --- | --- | --- |
| **Component / Method** | **Thread-Safe?** | **Why It’s Safe or Not** | **Notes / Recommendations** |
| Class (Dictionary) | ✅ Yes | Uses ConcurrentHashMap and CopyOnWriteArraySet with synchronization where needed | Designed for concurrent access |
| query(String word) | ✅ Mostly | Read from ConcurrentHashMap is thread-safe | ⚠️ Return a **defensive copy** to avoid external mutation |
| addWord(String, Set) | ✅ Yes | Uses putIfAbsent + CopyOnWriteArraySet | Safe for concurrent inserts |
| removeWord(String) | ✅ Yes | Uses ConcurrentHashMap.remove() | Atomic and thread-safe |
| addMeaning(String, String) | ✅ Yes | Synchronizes on per-word Set<String> | Ensures atomic modification of meanings |
| updateMeaning(...) | ✅ Yes | Synchronizes on per-word Set<String> | Prevents race conditions across check-modify actions |
| saveToFile() | ✅ Yes | Entire method is synchronized on fileLock | Prevents file corruption from simultaneous writes |
| loadInitialDictionary() | ⚠️ Partially | Not synchronized but safe if only called at startup | ❗ Synchronize or lock if used during runtime |

* GUI was created using Java Swing
  + ClientFrame – The frame that automatically opens when the DictionaryClient program is executed. Simple frame that has two textfields labelled “hostname” and “port number”.
    - Design choice to move this functionality outside of command line argument to GUI was that it more mimicked what would be expected in reality.
  + DictionaryOperationsFrame – What the client can see after successfully connecting to the server. Contains a home page with the operations they can perform.
  + Clicking through will take you to a separate frame that handles and prompts the user the instructions,

**Class and Interaction Diagram**

A screenshot of a computer

Description automatically generated

**Critical Analysis and Design Rationale**