Software Engineering Assessment Report

Candidate: Jeremiah Kerosi Abunga

Project: Standardized Introductory Task for Software Engineers

Company: Algorithmic Sciences

1. Project Overview

This report documents the implementation and evaluation of the server-based search system as per the provided specifications. The primary objectives were to develop a high-performance TCP server that supports concurrent connections, efficiently searches a large text file, and adheres to best software engineering practices, including robust exception handling, security, and PEP8 compliance.

2. Implementation Summary

2.1 Server Implementation

The server was implemented using Python and designed to handle multiple concurrent client requests efficiently. This was achieved through the use of multithreading, allowing multiple connections to be processed simultaneously without performance degradation. To enhance flexibility, the server retrieves configurable file paths from a `config.ini` file, ensuring ease of customization.

The search functionality is based on a full-line matching logic, ensuring that only exact matches are returned in response to client queries. Additionally, the server supports real-time file updates, which can be enabled or disabled through a configuration setting. When the `REREAD_ON_QUERY` option is set to `True`, the file is read on every query to capture the latest updates, while setting it to `False` ensures that the file is loaded once and remains static during execution.

To facilitate monitoring and debugging, the server incorporates a logging mechanism that records essential details, including timestamps, client IP addresses, and execution times. This ensures comprehensive tracking of operations and aids in troubleshooting. Security is also a critical component of the implementation, with SSL authentication integrated using self-signed certificates to enable

encrypted communication between the server and clients. Lastly, the server is designed to run as a Linux daemon, with detailed installation instructions provided for setting up and managing it as a background service.

2.2 Client Implementation

The client script was developed to streamline testing and interaction with the server. Its primary function is to send search queries and receive responses indicating whether the specified string exists in the server's dataset. Responses are displayed as either "STRING EXISTS" or "STRING NOT FOUND," ensuring clear feedback to the user.

To accommodate different security requirements, the client supports both SSL and non-SSL connections. The preferred mode can be selected through command-line options, allowing for flexibility based on the environment and security needs. This configurability ensures that users can test the server under various conditions, making it adaptable to different network setups.

2.3 Configuration File

The `config.ini` file provides a structured way for users to customize various server settings, ensuring flexibility and adaptability to different environments. One of its key parameters is `linux-path`, which specifies the file path to the text file that the server reads and processes. This allows users to define the location of the dataset without modifying the server script.

Another important setting is `reread_on_query`, which determines whether the text file is reloaded for every query. When set to `True`, the server reads the file each time a request is received, ensuring that the latest data is always used. Conversely, when set to `False`, the file is loaded only once, reducing overhead and improving performance for static datasets.

The configuration file also includes the `ssl_enabled` option, which allows users to enable or disable SSL authentication. When SSL is enabled, the server enforces secure, encrypted communication with clients, enhancing data security. If disabled, the server operates over a standard, unencrypted connection, which may be preferable for testing or environments where encryption is not required.

These configurable options ensure that the server can be tailored to meet specific security and performance requirements.

2.4 Security Considerations

Security was a critical aspect of the implementation, ensuring data integrity, confidentiality, and system stability. One of the key measures taken was robust input validation and exception handling to prevent potential vulnerabilities, such as buffer overflows and malformed queries. By thoroughly validating user inputs and implementing structured error handling, the system mitigates risks associated with unexpected or malicious data entries.

To ensure secure data transmission between the client and server, SSL/TLS encryption was integrated. This encryption mechanism protects sensitive information from interception and unauthorized access, particularly in network environments where data security is a concern. The use of self-signed certificates further enhances security by enabling encrypted communication without relying on external certificate authorities.

Efficient resource management was also a priority to handle large-scale parallel requests without system performance degradation. By optimizing memory allocation and thread handling, the server can effectively manage multiple concurrent client connections, preventing resource exhaustion and ensuring stable operation under high-load conditions. These security measures collectively enhance the reliability and resilience of the system against potential threats.

3. Performance Analysis

3.1 Benchmarking Methodology

To assess the efficiency of various file-search techniques, a series of benchmarks were conducted. The evaluation considered multiple performance factors, including execution time in relation to file size, which ranged from 10,000 to 1,000,000 lines. Additionally, the impact of concurrent queries on response time was analyzed to determine how well each approach handled simultaneous requests. The tests also measured CPU and memory usage under high-load conditions to identify potential bottlenecks and ensure optimal resource utilization.

3.2 Tested Algorithms

Several search algorithms were tested to compare their performance under different conditions. The basic linear search method served as the baseline for comparison. A binary search was implemented for pre-sorted files to explore its efficiency. A hash table lookup was evaluated, leveraging a dictionary-based approach for rapid search times. Additionally, a Trie data structure was tested to determine its effectiveness in optimizing prefix-based searches. Finally, a memory-mapped file search utilizing the `mmap` module was assessed to explore its efficiency in handling large files without excessive memory consumption.

3.3 Performance Results

The benchmarking results revealed distinct advantages and limitations for each approach. The basic linear search performed the worst, particularly for large files, as its execution time scaled linearly with file size. The hash table lookup provided the fastest search times but required a pre-processing step to construct the data structure. The memory-mapped file search (`mmap`) demonstrated significant improvements in efficiency, particularly for large files, achieving near 40ms response times for files containing 250,000 lines when `REREAD_ON_QUERY` was set to `True`.

While the binary search method exhibited good performance, it required the file to be sorted beforehand, making it impractical for frequently updated datasets. The Trie structure proved useful for prefix-based searches but was unnecessary in this case, given the requirement for full-line matching. Based on the findings, the final implementation adopted the `mmap`-based search due to its optimal balance between performance and efficient file handling.

4. Test Coverage & Robustness

A comprehensive testing strategy was implemented to ensure the reliability and robustness of the system. Unit tests were developed using `pytest`, covering all critical functional aspects to validate the correctness and stability of the implementation. These tests ensured that the system correctly processed search queries and returned expected results under various conditions.

Special attention was given to edge cases, including handling empty queries, processing exceptionally large files, and managing invalid file paths. Additionally, SSL authentication failures were tested to verify that unauthorized or improperly configured connections were correctly rejected. Performance constraints were also examined by subjecting the system to heavy loads, evaluating its ability to maintain responsiveness and stability under high-demand scenarios.

The test results indicated a 100% pass rate across all cases, demonstrating the system's robustness and adherence to functional requirements. This thorough validation process ensured that the implementation was both reliable and capable of handling diverse operational conditions effectively.

5. Compliance with Best Practices

Criterion	Status
PEP8 Compliance	Pass
PEP20 Adherence	Pass
Static Typing (`mypy`)	Pass
Robust Exception Handling	Pass
Security Best Practices	Pass
Documentation & Docstrings	Pass
Logging & Debugging Features	Pass

6. Conclusion & Recommendations

The implemented system successfully meets all the requirements outlined in the task specification. It provides efficient search functionality, robust security measures, and comprehensive configurability. The use of the `mmap`-based search method ensures optimal performance, particularly for handling large files while minimizing memory overhead.

The solution is designed to be secure, scalable, and well-documented, making it adaptable for various deployment environments. SSL/TLS encryption, input validation, and resource management contribute to its overall security and reliability. Additionally, all necessary unit tests and benchmarking reports were conducted, confirming the system's stability and efficiency.

Testing revealed no critical issues, demonstrating the robustness of the implementation. Based on these findings, the system is ready for deployment, with future enhancements potentially focusing on further optimizations and extended search functionalities to accommodate evolving require-ments.