Design and Implementation of a Multithreaded Web Server

Project plan

Qasim Ijaz

CS3821 — BSc Final Year Project

Supervised by: Dr DongGyun Han
Department of Computer Science
Royal Holloway, University of London

1 Abstract

In an era marked by the burgeoning growth of web-based services and applications, the need for high-performance web servers has never been more critical. Web servers are the linchpins of the internet, serving as the conduit for delivering content to myriad client applications, such as web browsers. A significant challenge that web servers face is efficiently managing an increasing volume of simultaneous client requests, which, if not addressed adequately, can lead to performance degradation and resource exhaustion. This project aims to confront this challenge head-on by designing, implementing, and rigorously evaluating a multithreaded web server optimized for both efficiency and robustness.

The server's design employs the principle of multithreading, a strategy that permits multiple client requests to be processed concurrently by allocating individual threads to each request. This architecture aims to resolve the limitations found in traditional, single-threaded web servers by mitigating thread contention and avoiding excessive thread creation and destruction. The project unfolds in a structured manner, organized into early and final deliverables to ensure a focused and methodical implementation.

The early deliverables include a rudimentary, single-threaded web server, which serves as a foundation for grasping the core concepts of client-server models, HTTP protocol, and network programming. Accompanying this are comprehensive reports that dissect the intricacies of HTTP communication, elaborating on the request/response model, methods, status codes, and headers. Various proof-of-concept (PoC) implementations are also provided, showcasing a thread pool, memory-mapped files for efficient I/O operations, and an LRU (Least Recently Used) cache for effective data retrieval. These PoCs will be instrumental in addressing challenges related to multithreading, such as race conditions, deadlocks, and synchronization issues, by proposing well-thought-out strategies.

The final deliverables are highlighted by a fully-functional, multithreaded web server capable of efficiently managing a high volume of simultaneous client requests. This server will be the embodiment of rigorous academic research and practical programming skills, offering a real-world solution to a real-world problem. To supplement the server's technical capabilities, comprehensive documentation will be provided, shedding light on the server's design architecture, key features, rationale behind implementation choices, and potential bottlenecks. Moreover, exhaustive performance tests and analyses will be carried out to assess the server's behavior under varying conditions. Metrics like response time, throughput, and resource utilization will be collected and compared against a single-threaded server to offer a quantifiable measure of the benefits achieved through multithreading.

In summation, this project aspires to bridge the gap between theoretical knowledge and practical skills in server architecture, multithreading, HTTP protocols, and network programming. By successfully achieving these aims, we anticipate setting a new benchmark in web server technology, offering a scalable, robust, and high-performance solution suitable for modern internet demands.

2 Timeline

The timeline section provides a structured and chronological overview of the project's significant milestones and activities. This timeline serves as a roadmap, illustrating the project's progression from its inception to its conclusion. For each term I have defined the various tasks I aim to complete and what those tasks involve and which weeks I aim to complete each task in.

2.1 Term 1

- Week 1: Study C++ Multi-threading and Concurrency Material
- Week 2: Design Thread Pool and It's Functionality, Begin Writing Report 1
- Week 3-4: Implement Thread Pool and Conduct Performance Testing For Optimisations.
- Week 5-6: Design & Research LRU Caching Mechanism, Begin Writing Design Document 1
- Week 7: Implement, Test and Integrate LRU Caching Mechanism Into Server
- Week 8-9: Research Memory-Mapping and How to Implement, Begin writing Report 2
- Week 10-11: Implement Memory-Mapping Functionality PoC and Integrate Into Project

2.2 Term 2

- Week 1: Integrating Thread Pool Into Single-Threaded Server
- Week 2: Begin Writing Documentation 1, and Final Project Report
- Week 3-4: Integrate LRU Caching Mechanism Into Server
- Week 5-6: Integrate Memory-Mapping Functionality Into Server
- Week 7: Your long sentence for Week 7 here
- Week 8-9: Your long sentence for Week 8-9 here
- Week 10-11: Your long sentence for Week 10-11 here

3 Proof of Concept Programs

A key part of this project is the proof of concept (PoC) programs which will highlight the key functionality of the web server. The PoC programs will tackle the fundamental aspects of the project including multi-threading, I/O performance optimisation and efficient caching mechanisms. By delving into these I am to gather improve the functionality and performance of the web server using various means.

3.1 Thread Pool PoC

The thread pool is a fundamental and critical component of the web server. The server will utilise a thread pool to offload tasks to consumer threads in a more efficient way compared to creating new threads to execute new tasks are destroying those threads after the task is complete. The thread pool will consist of a pool of consumer threads and consumer threads will wait for tasks to be pushed into the task queue. Once tasks have been pushed into the queue by producers, consumers will then execute those tasks. By utilising a thread pool the performance of the web server can be improved since thread creation and destruction is minimised.

3.2 LRU Cache PoC

By introducing a LRU (Least Recently Used) caching mechanism we can cache certain pieces of data which have recently been used or requested by clients. Using a mechanism like this allows us to identify which pieces of data have recently been requested or interacted with in an efficient manner. By using an LRU caching mechanism the web server can retrieve recently requested data without having to go through the trouble of retrieving the same piece of data multiple times in a row.

3.3 Memory-Mapping PoC

Memory-mapping is a technique that enables files or devices to be directly loaded into the memory address space of a process. Within the context of the web server, utilizing memory-mapped files can greatly boost I/O performance. Instead of traditional file reading methods, which can be slow and cumbersome, memory-mapping permits the server to access file contents as if they were part of its own memory. This approach minimizes explicit read/write operations, thereby reducing I/O latency significantly. By incorporating this proof of concept, the web server aims to deliver faster response times, especially when handling large files or frequently accessed data.

4 Project Reports, Documents & Documentation

Throughout the project, I will be writing multiple reports which will form the basis of the final project report. Each report will dive into various topics in line with the project's evolution and objectives. Furthermore I will produce a design document outlining the servers architecture and documentation for the web server.

4.1 Report 1:

A report showing a strong understanding and overview of the HTTP protocol, including the request/response model, methods, status codes, headers, and the underlying TCP/IP model.

4.2 Report 2:

A report on optimization strategies for low latency and high performance in web servers which will delve into the various techniques and principles employed to enhance the performance of a web server.

4.3 Report 3:

Rigorous testing and performance analysis reports, detailing the server's behaviour under different loads, error conditions, and comparison with a single-threaded server. These reports should include measures of response time, throughput, and resource usage, among other metrics.

4.4 Design Document 1:

A comprehensive design document illustrating the architecture and workflow of the multi-threaded server. This should include identifying potential challenges in multi-threading, such as race conditions or deadlocks, and proposing strategies for handling these issues.

4.5 Documentation 1:

Comprehensive documentation including server design, key features, explanation behind implementation choices, potential bottlenecks, and clear instructions for setup, use and troubleshooting.

5 Risks and Mitigations

5.1 Race Conditions and Deadlocks

A common risk in multi-threaded applications is the potential for race conditions and deadlocks to occur. These issues can induce unpredictable behavior in a web server, ranging from system instability to outright crashes. Such erratic behavior epitomizes undefined behavior in computing contexts. To ensure the web server remains thread-safe and consistently operational, it's essential to vigilantly address and prevent these risks by through code analysis and testing, appropriate synchronisation primitives and deadlock prevention protocols.

5.2 Thread Contention Issues

Thread contention represents a pivotal concern in high-performance systems. In multi-threaded contexts, pronounced contention can significantly impede overall program performance, as numerous threads vie for access to concurrently occupied resources. To mitigate such contention, it's imperative to minimize or, if feasible, entirely abstain from data sharing between threads.

5.3 Inefficient Memory Management

Another risk is the inefficiency of memory management. In low-latency environments, the overhead from frequent run-time allocations and subsequent de-allocations can substantially compromise performance. Continual memory operations can hamper runtime efficiency. To counteract this, the use of memory pools is often advocated. Memory pools involve pre-allocating a substantial, fixed-size buffer at the program's inception. This strategy curtails the overhead linked to repeated memory operations by facilitating quicker and more predictable memory access.

5.4 False Sharing Overhead

A significant performance-related risk, especially in multi-threaded environments, is false sharing. This phenomenon occurs when cache lines are invalidated due to simultaneous writes to unrelated data residing on the same cache line. Consequently, even if only a portion of the data on the cache line is modified, the entire cache line may be refreshed from the main memory. To mitigate this, memory padding and alignment techniques can be employed. By ensuring data is strategically positioned on separate cache lines, one can effectively reduce the risk of false sharing.

5.5 C++ Language Feature Overhead

C++ is a robust and versatile programming language endowed with a multitude of features. However, when developing performance-critical software, it is imperative to understand the intricacies of the underlying data structures and language constructs. Overlooking these details can inadvertently compromise the software's performance. To ensure optimal performance, it's crucial to comprehend the underlying mechanics of each language feature and assess its potential impact.

6 Performance Evaluation

6.1 Testing Methodology

6.2 Performance Metrics

6.3 Comparison: Multi-threaded vs. Single-threaded Server

7 References