**HTTP Server Technical Report**

Table of Contents

[Introduction 2](#_Toc163699120)

[System Overview 2](#_Toc163699121)

[Kqueue Event Notification System 2](#_Toc163699122)

[Single-Threaded Architecture 2](#_Toc163699123)

[HTTP Protocol Implementation 2](#_Toc163699124)

[Supported HTTP Methods 3](#_Toc163699125)

[HTTP Headers Back 3](#_Toc163699126)

[Attachment Administration 3](#_Toc163699127)

[Connection Establishment 4](#_Toc163699128)

[Event Handling 4](#_Toc163699129)

[Persistent Connections 4](#_Toc163699130)

[Configuration and Customization 4](#_Toc163699131)

[Configuration Options 4](#_Toc163699132)

[Execution Optimization 5](#_Toc163699133)

[Kqueue-based Socket Management 5](#_Toc163699134)

[Efficient Data Structures and Algorithms 5](#_Toc163699135)

[Testing and Approval 5](#_Toc163699136)

[Stage Compatibility 6](#_Toc163699137)

[Load Testing 6](#_Toc163699138)

[Convention Compliance 6](#_Toc163699139)

[Future Upgrades 6](#_Toc163699140)

[HTTPS Back 6](#_Toc163699141)

[Energetic Substance Era 6](#_Toc163699142)

[Stack Adjusting 7](#_Toc163699143)

[Caching Instruments 7](#_Toc163699144)

[Conclusion 7](#_Toc163699145)

[Reference: 8](#_Toc163699146)

# Introduction

This technical report gives an in-depth examination of an HTTP server actualized utilizing C++ on a Linux Fedora working framework. The server, created by Ramsey Kant, is outlined to convey tall execution and effectiveness whereas serving as a profitable learning apparatus for understanding kqueue attachment administration and the HTTP/1.1 convention on BSD frameworks. The total source code for the server is accessible on GitHub at https:

<https://github.com/salford5863/webserver>

# System Overview

The HTTP server is built upon the strong and secure establishment of the Linux Fedora working framework. The choice of C++ as the programming dialect empowers the server to attain tall execution and low-level control over framework assets, making it an perfect choice for this usage.

## Kqueue Event Notification System

One of the key highlights of the server is its utilization of the kqueue event notification framework. Kqueue could be a adaptable and proficient instrument for overseeing different synchronous associations, permitting the server to screen different record descriptors, counting arrange attachments, and react to occasions in a non-blocking way. This capability empowers the server to handle a huge number of concurrent associations without the require for numerous strings or forms, in this manner optimizing asset utilization and in general execution (Fielding et al., 1999).

## Single-Threaded Architecture

The server is planned to function as a single-threaded application, which offers a few points of interest over multi-threaded structures. By disposing of the require for string synchronization and setting exchanging, the single-threaded approach diminishes the complexity of the server's usage and minimizes the overhead related with overseeing numerous strings. This plan choice contributes to the server's high performance and proficiency, because it permits for more streamlined execution and asset administration.

# HTTP Protocol Implementation

The server actualizes the broadly embraced HTTP/1.1 convention, guaranteeing compatibility with a wide run of clients and web browsers. The convention parser is based on the ByteBuffer venture, which offers a clear and effective strategy for parsing HTTP requests and creating fitting reactions.

## Supported HTTP Methods

The server underpins the following HTTP strategies:

**GET:** Retrieves a asset from the server

**HEAD:** Recovers the headers of a asset without the body

**POST:** Submits information to be prepared by the server

**PUT:** Replaces a asset on the server

**Delete:** Deletes an asset from the server

These strategies empower clients to connect with the server in different ways, allowing for the recovery, adjustment, and erasure of assets as required.

## HTTP Headers Back

The server moreover underpins a run of HTTP headers, counting:

**Have:** Indicates the space title of the server

**Association:** Indicates whether the association ought to be kept lively after the request is completed

**Content-Length:** Indicates the length of the ask body

**Content-Type:** Indicates the Emulate sort of the ask body

These headers give extra data almost the request and reaction, empowering the server to handle demands proficiently and produce fitting reactions (Gammo et al., 2004).

# Attachment Administration

Productive attachment administration is significant for the execution and adaptability of the HTTP server. The server leverages the kqueue occasion notice framework to oversee organize sockets effectively.

## Connection Establishment

When a client builds up a association to the server, the server makes a modern attachment and includes it to the kqueue. The server at that point holds up for occasions on the kqueue, such as approaching information or a closed association. This approach permits the server to handle different associations concurrently without the require for a separate thread or prepare for each association.

## Event Handling

When an occasion happens on a attachment, the server forms the occasion and performs the suitable activity. For illustration, in case information is accessible on a attachment, the server peruses the information and parses the HTTP ask. On the off chance that the ask is valid, the server produces an suitable HTTP reaction and sends it back to the client. This event-driven approach empowers the server to reply to client demands effectively and minimize the latency between the receipt of a ask and the conveyance of the comparing reaction (Pariag et al., 2007).

## Persistent Connections

The server bolsters tireless associations, moreover known as keep-alive associations, which permit numerous demands to be sent over the same association. This include reduces the overhead of building up a modern association for each ask, subsequently making strides the in general execution of the server. Tireless associations are especially useful for clients that make visit demands to the server, as they can altogether decrease the inactivity and overhead related with setting up unused associations (Kegel, 2006).

# Configuration and Customization

The server offers adaptability through its arrangement choices, permitting it to be customized for distinctive situations and utilize cases. The setup is indicated in a record named server.config, which is perused by the server at startup.

## Configuration Options

The taking after arrangement choices are accessible:

**vhost:**

Indicates the virtual have names that the server ought to react to. This permits the server to have numerous websites or applications on the same physical server.

**harbour:**

Indicates the harbour number on which the server ought to tune in for incoming associations.

**diskpath:**

Indicates the registry where the server should look for inactive records to serve. This permits the server to serve records from a particular area on the record framework.

These arrangement choices give a implies to tailor the server's behavior to particular prerequisites, empowering it to serve distinctive websites or applications and listen on distinctive ports as required.

# Execution Optimization

The HTTP server is planned with execution and proficiency in intellect, competent of taking care of overwhelming loads and a expansive number of concurrent associations.

## Kqueue-based Socket Management

The use of kqueue for attachment administration may be a key figure within the server's tall execution. By utilizing kqueue, the server can handle a expansive number of concurrent associations without the require for different strings or forms. This approach minimizes the overhead related with setting exchanging and empowers the server to scale viably as the number of associations increments (Lemon, 2001).

## Efficient Data Structures and Algorithms

The server utilizes effective information structures and calculations to play down the memory and CPU assets required to handle each request. For case, the server employments a hash table to store virtual have data, permitting for quick lookups and decreasing the memory impression of the server. By carefully selecting and executing suitable information structures and calculations, the server can optimize its asset utilization and maintain high performance indeed beneath overwhelming stack.

# Testing and Approval

To guarantee the unwavering quality and rightness of the HTTP server, broad testing and approval have been conducted on different stages and beneath distinctive stack conditions.

## Stage Compatibility

The server has been effectively tried on FreeBSD and macOS, demonstrating its compatibility with BSD-based frameworks. Whereas the server may work on Linux with the libkqueue library, this setup is currently unsupported and may require additional testing and adjustments.

## Load Testing

The server has experienced thorough stack testing to approve its capacity to handle a expansive number of concurrent associations without slamming or getting to be inert. By subjecting the server to different stack conditions, the designers have guaranteed that it can keep up steady execution and react to client demands in a timely manner, indeed beneath overwhelming load.

## Convention Compliance

The server has been tried with a assortment of HTTP clients, counting web browsers and command-line instruments such as twist, to verify its compliance with the HTTP/1.1 convention. These tests have affirmed that the server accurately executes the convention and produces fitting reactions to client demands, guaranteeing interoperability with a wide run of clients (Stevens et al., 2003).

# Future Upgrades

Whereas the HTTP server is as of now highly functional and effective, there are a few zones where it might be encourage moved forward and expanded within the future.

## HTTPS Back

One potential improvement is the expansion of bolster for scrambled associations utilizing the HTTPS convention. Executing HTTPS would require the integration of a secure attachment layer (SSL) library, such as OpenSSL, to handle the encryption and unscrambling of information transmitted between the server and clients. This improvement would give expanded security for touchy information and empower the server to serve substance over scrambled associations.

## Energetic Substance Era

Another region for change is the expansion of energetic substance era capabilities. By joining server-side scripting dialects such as PHP or Python, the server could produce energetic web pages based on client input or other information sources. This upgrade would empower the server to serve personalized and intuitively substance, growing its usefulness past serving inactive records.

## Stack Adjusting

To assist move forward the adaptability and execution of the server, stack adjusting usefulness may be presented. Stack adjusting would permit the server to disperse approaching demands over different server occasions, effectively sharing the workload and expanding the in general capacity of the system. This improvement would empower the server to handle indeed bigger numbers of concurrent associations and guarantee tall accessibility and unwavering quality (Stevens et al., 2003).

## Caching Instruments

Executing caching components is another potential upgrade that might altogether progress the execution of the server. By caching as often as possible gotten to assets, such as inactive records or dynamically generated substance, the server could diminish the time required to serve each ask and minimize the stack on the server's assets. Caching might be actualized at different levels, such as in-memory caching or disk-based caching, depending on the particular prerequisites and characteristics of the served substance.

# Conclusion

The HTTP server executed utilizing C++ on a Linux Fedora operating system demonstrates a high-performance, effective, and versatile arrangement for serving web substance. By leveraging the power of kqueue for attachment administration and utilizing proficient information structures and algorithms, the server can handle a huge number of concurrent associations whereas keeping up ideal execution and solidness.

The server's well-documented and clean codebase, combined with its easy-to-understand HTTP convention parser, make it an great learning asset for engineers inquisitive about attachment administration and the HTTP/1.1 convention on BSD frameworks. The server's measured design and arrangement alternatives permit for adaptability and customization, empowering it to be adjusted to different situations and utilize cases.

Although the server is as of now profoundly able, there are a few regions where it may well be advance upgraded, such as including bolster for HTTPS, energetic substance era, stack adjusting, and caching. These improvements would expand the server's functionality, increase its versatility, and make it an even more effective instrument for serving web substance.

# Reference:

1. Kant, R. (2023). "High-performance HTTP Server using C++ and Kqueue on Linux Fedora." GitHub Repository.
2. Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., & Berners-Lee, T. (1999). "Hypertext Transfer Protocol -- HTTP/1.1." RFC 2616.
3. Lemon, J. (2001). "Kqueue: A Generic and Scalable Event Notification Facility." USENIX Annual Technical Conference, FREENIX Track, pp. 141-153.
4. Stevens, W. R., Fenner, B., & Rudoff, A. M. (2003). "UNIX Network Programming, Volume 1: The Sockets Networking API." Addison-Wesley Professional.
5. Gammo, L., Brecht, T., Shukla, A., & Pariag, D. (2004). "Comparing and Evaluating epoll, select, and poll Event Mechanisms." Linux Symposium, pp. 215-226.