CSE 4074 – Programming Assignment Report

Socket Programming – HTTP Server and Proxy Server

**Project Members:**  
Cem Eren Kula 150120059

Doğukan Onmaz 150120071

Ferhat Sirkeci 150120067

**HTTP Server**

**Overview**

The HttpServer.py file implements a simple HTTP server capable of handling client requests. It leverages multi-threading to handle multiple connections concurrently. The server responds to GET requests, checks the validity of the request URI, and generates appropriate HTTP responses.

**Features**

1. **Multi-threaded Design**:

* Each client connection is handled in a separate thread, ensuring the server can process multiple requests simultaneously.

1. **Request Parsing**:

* Validates the HTTP method (supports GET only).
* Checks the URI for integer values within a specified range (100 to 20,000).
* Handles non-integer URIs (e.g., favicon.ico).

1. **Response Generation**:

* Constructs HTTP responses based on the status of the request (200 OK, 400 Bad Request, or 501 Not Implemented).
* Allows for optional saving of responses to HTML files in a “Responses” directory.

1. **Port Configurability**:

* The port on which the server listens can be specified via command-line arguments.

**Code Details**

1. **Request Parsing**:

* The parse\_request() function validates the request method and URI.
* Unsupported methods return a 501 Not Implemented status, while invalid URIs return a 400 Bad Request.

2. **Response Generation**:

* The generate\_response() function creates HTML responses based on the status code.
* If enabled, it saves responses as files in the Responses directory.

3. **Concurrency**:

* The handle\_client() function uses threading to ensure independent handling of client connections.
* A shared counter (response\_counter) tracks the number of responses generated, protected by a thread-safe lock.

4. **Server Setup**:

* The start\_server() function initializes a socket and listens for incoming connections.
* Client connections are accepted and handled in separate threads.

**Potential Enhancements**

We can add a cache mechanism to store frequently requested responses, reducing processing time for repeated requests.

**Proxy Server**

**Overview**

ProxyServer.py implements a simple proxy server designed to forward HTTP requests to a specified web server. The proxy enforces restrictions on allowed hosts and ports, ensuring controlled access to backend resources. It employs multi-threading to handle multiple client connections concurrently.

**Features**

1. **Request Handling**:

* Parses incoming HTTP requests, extracting the method, URI, and HTTP version.
* Handles HTTP requests with the http:// scheme, routing them to the appropriate destination.
* Verifies host and port against predefined allowed values.

2. **Request Forwarding**:

* Rewrites the request line and headers before forwarding them to the target web server.
* Establishes a connection to the web server and sends the modified request.

3. **Response Transmission**:

* Receives responses from the web server and relays them back to the client without modification.

4. **Access Control**:

* Implements strict access controls by allowing requests only to specified hosts and ports (localhost:8080 in this implementation).
* Returns an HTTP 403 Forbidden status for disallowed requests.

5. **Multi-threading**:

* Uses threading to handle multiple client connections simultaneously, ensuring high responsiveness.

**Code Details**

1. **Request Parsing**:

* Extracts and validates the method, URI, and HTTP version.
* Parses the host and port from the URI for routing.

2. **Access Control**:

* The handle\_client() function checks if the requested host and port match the allowed configuration (ALLOWED\_HOST and ALLOWED\_PORT).
* Disallowed requests return a 403 Forbidden response.

3. **Request Forwarding**:

* Constructs a modified HTTP request with updated headers.
* Forwards the request to the specified backend server using a new socket connection.

4. **Response Handling**:

* Relays responses from the backend server back to the client.
* Returns a 502 Bad Gateway response if the backend server is unreachable.

5. **Thread Management**:

* Each client connection is managed in a separate thread using Python's threading module.
* Threads operate independently to ensure concurrent handling of requests.

**Potential Enhancements**

We can also add caching to proxy server.

**Stress Testing Proxy Server**

**Note: In first 6 test we are only changing Number of Threads.**

* 1. **Test**

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

* 1. **Test**

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

* 1. **Test**

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

* 1. **Test**

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

* 1. **Test**

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

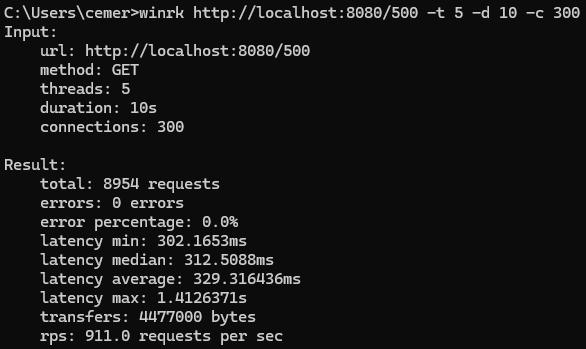
* 1. **Test**

metin, ekran görüntüsü, yazı tipi içeren bir resim

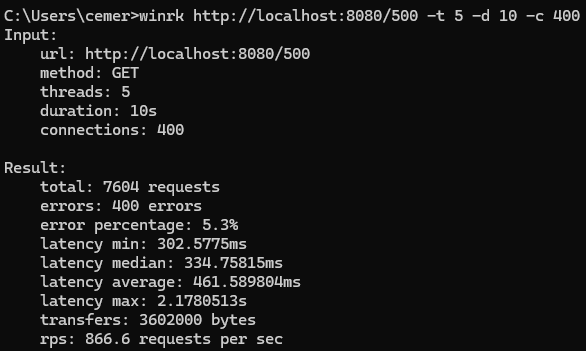
Açıklama otomatik olarak oluşturuldu

**Note: Now we are going to change parallel connection number.**

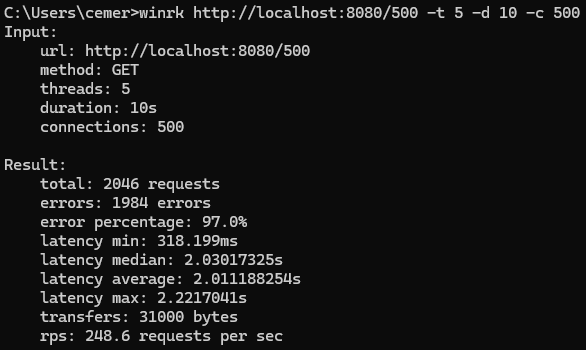
* 1. **Test**



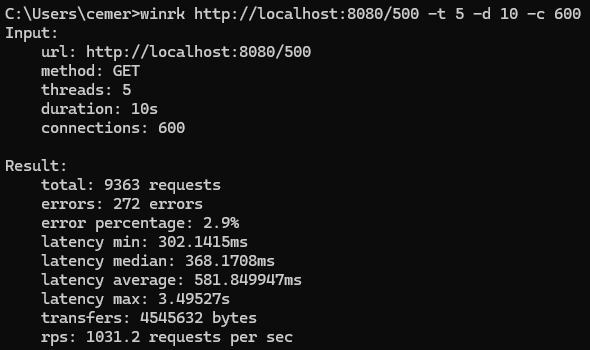
* 1. **Test**



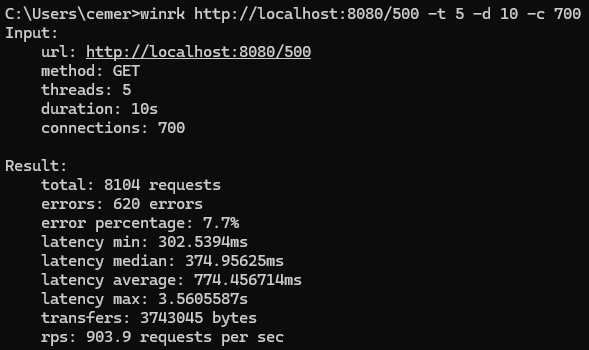
* 1. **Test**



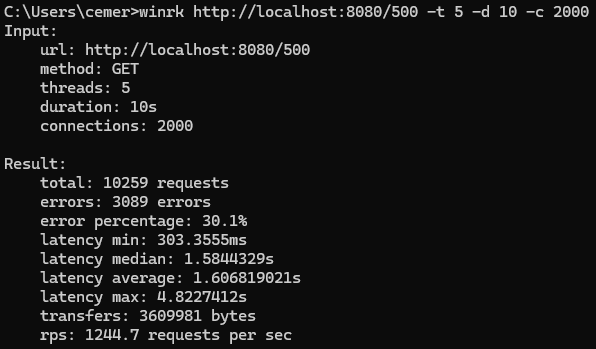
* 1. **Test**



* 1. **Test**

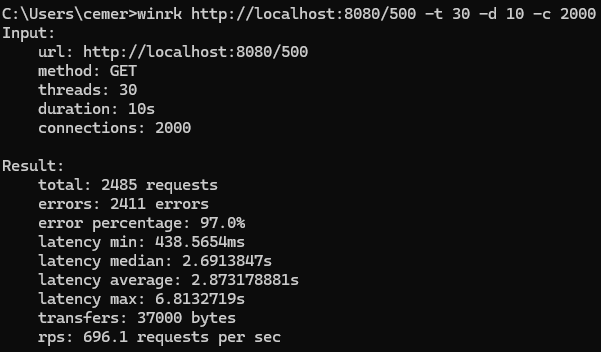


* 1. **Test**



**Note: Now we are going to change parallel connection number and thread number at the same time.**

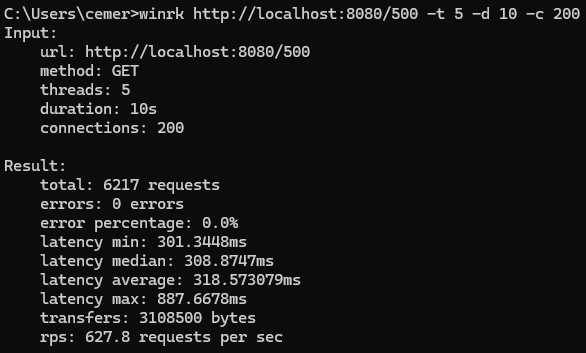
* 1. **Test**



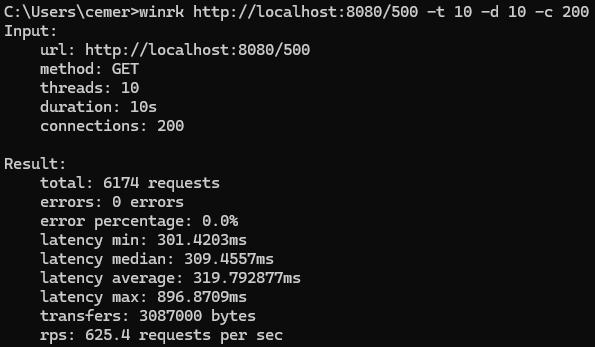
**Stress Testing HTTP Server**

**Note: We are going to change only Number of Threads.**

* 1. **Test**



* 1. **Test**

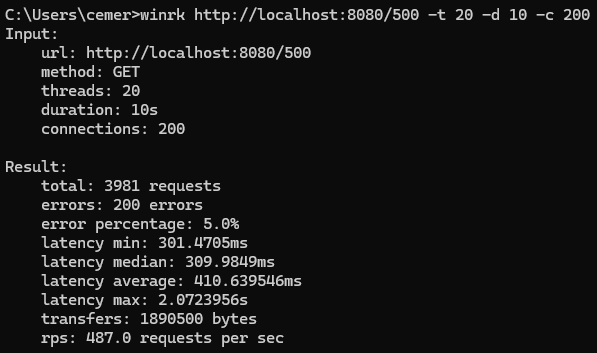


* 1. **Test**

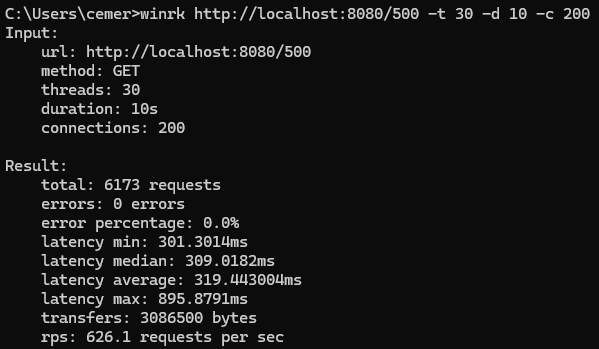
metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

* 1. **Test**



* 1. **Test**



**Note: Now we are going to change only parallel connection number.**

* 1. **Test**

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

* 1. **Test**

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

* 1. **Test**

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Note: Now we are going to also increase thread number with 500 connections.**

* 1. **Test**

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Conclusion**

When conducting stress tests on the proxy server, the following behaviors were observed:

1. **Impact of Increasing Parallel Connections**:
   * As the number of parallel connections increases, both the error rate and average latency rise significantly.
   * This occurs because the server struggles to manage a higher volume of simultaneous requests with a fixed number of threads.
2. **Effect of Increasing Thread Count**:
   * By increasing the number of threads, the server can handle a greater degree of concurrency more efficiently.
   * Higher thread counts allow the server to respond to simultaneous requests with reduced latency and error rates.
3. **Thread Count vs. Latency and Errors**:
   * Simply increasing the thread count without increasing the number of parallel connections does not affect latency or error rates.
   * This demonstrates that the server's performance is primarily limited by the demand for concurrency rather than the thread pool size under low-load conditions.

These insights highlight the importance of tuning thread count relative to expected traffic patterns for optimal performance during high-concurrency scenarios.