COMP 2322 Project Report

Multi-threaded Web Server

Cagan Cakir
24140489x

Summary Of Design and Implementation

This web server uses the socket library of the Python(3.10) programming language for establishing a TCP/IP connection. First, the server tries to bind with a particular port, verifying for port availability; if the port is being utilized, the server moves to the next available port, informing the user of the port change distinctly within the console output.

```
cagancakir@Cagans-MacBook-Pro network % python3.10 network.py
[info] Serving www on port 8080 — Ctrl+C to stop…
```

Figure 1(Website running on port 8080)

```
cagancakir@Cagans-MacBook-Pro network % python3.10 network.py
[warning] Port 8080 is already in use
[info] Port changed from 8080 to 8081
[info] Serving www on port 8081 — Ctrl+C to stop…
```

Figure 2(Website running on port 8081, when 8080 is being used)

Each new incoming connection is handled using Python's threading module. Within the function run_server, all new incoming clients create a separate thread using the respond function, enabling multiple requests from clients to be handled concurrently without hindering the server's performance.

With the help of the respond function, valid request and response message transactions are supported through reading from as well as parsing of the raw HTTP request. HTTP request headers are systematically split up and checked strictly for complete compliance with HTTP standards, rejecting malformed requests directly with a 400 Bad Request status.

The execution of GET commands for text files as well as image files is accurately done within the respond function. The server relies on the built-in method of Python, which is mimetypes.guess_type, to determine the type of the requested files. Properly detected files are sent using the send_simple method, which creates the right HTTP headers, i.e., Content-Type, Content-Length, as well as Date. If the requested resource is "index.html" or a slash, the server serves "index.html" from the given document root (by default, "www") by default.

Moreover, the HEAD commands are executed inside the respond function. This command sends the same header details as the GET command but does not send the content of the actual file, strictly as per HTTP requirements.

The server processes six categories of HTTP response statuses systematically: 200 OK, 304 Not Modified, 400 Bad Request, 403 Forbidden, 404 Not Found, and 415 Unsupported Media Type. Responses for successful requests (200, 304) are crafted with exact headers and content with send_simple. Error messages are handled with caution using the send_error method,

offering explicit and helpful error messages to users. The handling is implemented explicitly for Last-Modified and If-Modified-Since headers under the respond function. By matching modification timestamps of the file with the timestamp sent by the client, the server can efficiently control caching at the client end, avoiding redundant data transmission. It provides a 304 Not Modified response if the resource hasn't been modified since the previous request.

Persistent connections (Keep Alive) are controlled through the Connection header of the respond method. Persistent connections are kept alive by servers for clients requesting persistent connections, for efficient utilization of network resources. Keep-alive timeout duration is set specifically as 10 seconds, within which the server will keep accepting subsequent requests over a single connection. If no more requests are accepted within a time specified above, the connection gets closed automatically for releasing resources.

Security is managed through strict path sanitization in the respond method. Paths are normalized and checked against the root of the document to reject unauthorized accesses, returning a 403 Forbidden error response directly for unauthorized requests.

Detailed logging for every request and response interaction is managed by the log_request function. It captures essential details including the client's IP address, request method, requested resource, response status, and payload size, all formatted in an Apache-style log file, aiding in monitoring and debugging.

Demonstration Of Execution

1.Go to the correct directory and run the following to start the website:

Figure 3

2.Now you can visit the website with the given URL.

2.1.1. When http://172.16.156.148:8080/index.html is run in browser:

```
[Request] GET /index.html HTTP/1.1
[Client] 172.16.156.148:64762
[Header] Host: 172.16.156.148:8080
[Header] User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/135.0.0.0 Safari
/537.36
[Header] Connection: keep-alive
[Path] /index.html -> www/index.html
[MIME] text/html
[Response] 200 OK (44 bytes, Last-Modified: Tue, 22 Apr 2025 09:33:31 GMT)

[Response Headers]
  HTTP/1.1 200 OK
  Date: Sun, 27 Apr 2025 09:58:03 GMT
  Server: COMP2322/1.0
  Connection: keep-alive
  Content-Length: 44
  Content-Type: text/html
  Last-Modified: Tue, 22 Apr 2025 09:33:31 GMT
  [Body] 44 bytes
[Connection] Closed for 172.16.156.148:64763
[Connection] Closed for 172.16.156.148:64762
```

Figure 4(200 OK)

2.1.2. When http://172.16.156.148:8080/IMG.JPG is run in browser:

```
[Connection] New connection from 172.16.156.148:62754

[Request] GET /IMG.JPG HTTP/1.1
[Client] 172.16.156.148:62754
[Header] Host: 172.16.156.148:8080
[Header] User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/135.0.0.0 Safari/537.36
[Header] Connection: keep-alive
[Path] /IMG.JPG -> www/IMG.JPG
[MIME] image/jpeg
[Connection] New connection from 172.16.156.148:62755
[Response] 200 OK (828440 bytes, Last-Modified: Mon, 31 Mar 2025 01:39:10 GMT)

[Response Headers]
  HTTP/1.1 200 OK
  Date: Tue, 29 Apr 2025 07:09:40 GMT
  Server: COMP2322/1.0
  Connection: keep-alive
  Content-Length: 828440
  Content-Type: image/jpeg
  Last-Modified: Mon, 31 Mar 2025 01:39:10 GMT
  [Body] 828440 bytes
[Connection] Closed for 172.16.156.148:62755
[Connection] Closed for 172.16.156.148:62754
```

Figure 5(200 OK)

2.2. When refreshed:

```
[Connection] New connection from 172.16.156.148:64786
[Connection] New connection from 172.16.156.148:64787

[Request] GET /index.html HTTP/1.1
[Client] 172.16.156.148:64786
[Header] Host: 172.16.156.148:8080
[Header] User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/135.0.0.0 Safari
/537.36
[Header] If-Modified-Since: Tue, 22 Apr 2025 09:33:31 GMT
[Header] Connection: keep-alive
[Path] /index.html -> www/index.html
[MIME] text/html
[Response] 304 Not Modified (Last-Modified: Tue, 22 Apr 2025 09:33:31 GMT)

[Response Headers]
  HTTP/1.1 304 Not Modified
  Date: Sun, 27 Apr 2025 09:59:44 GMT
  Server: COMP2322/1.0
  Connection: keep-alive
  Last-Modified: Tue, 22 Apr 2025 09:33:31 GMT
  [Body] None
[Connection] Closed for 172.16.156.148:64787
[Connection] Closed for 172.16.156.148:64786
```

Figure 6(304 NOT MODIFIED)

2.3. When http://172.16.156.148:8080/index.html is run on Testfully as Post method:

```
[Connection] New connection from 172.16.137.135:60158

[Request] POST / HTTP/1.1
[Client] 172.16.137.135:60158
[Header] Host: 172.16.156.148:8080
[Header] User-Agent: Testfully/1.0.0
[Header] Connection: close
[Error] Unsupported method: POST

[Response Headers]
  HTTP/1.1 400 Bad Request
  Date: Sun, 27 Apr 2025 10:13:26 GMT
  Server: COMP2322/1.0
  Connection: close
  Content-Length: 24
  Content-Type: text/html
  [Body] None
[Connection] Closed for 172.16.137.135:60158
```

Figure 7(400 BAD REQUEST)

2.4. When “curl -v http://172.16.156.148:8080/some_dir/%2E%2E/index.html” is run on terminal:

```
[Connection] New connection from 172.16.156.148:65245

[Request] GET /some_dir/%2E%2E/index.html HTTP/1.1
[Client] 172.16.156.148:65245
[Header] Host: 172.16.156.148:8080
[Header] User-Agent: curl/8.7.1
[Error] Path traversal attempt: /some_dir/../index.html

[Response Headers]
  HTTP/1.1 403 Forbidden
  Date: Sun, 27 Apr 2025 10:19:34 GMT
  Server: COMP2322/1.0
  Connection: keep-alive
  Content-Length: 22
  Content-Type: text/html
  [Body] 22 bytes
[Connection] Closed for 172.16.156.148:65245
```

Figure 8(403 FORBIDDEN)

2.5. When http://172.16.156.148:8080/nonexistent.html is run on browser:

```
[Connection] New connection from 172.16.156.148:65288

[Request] GET /nonexistent.html HTTP/1.1
[Client] 172.16.156.148:65288
[Header] Host: 172.16.156.148:8080
[Header] User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome
/135.0.0.0 Safari/537.36
[Header] Connection: keep-alive
[Path] /nonexistent.html -> www/nonexistent.html
[Connection] New connection from 172.16.156.148:65289
[Error] File not found: www/nonexistent.html

[Response Headers]
  HTTP/1.1 404 Not Found
  Date: Sun, 27 Apr 2025 10:21:52 GMT
  Server: COMP2322/1.0
  Connection: keep-alive
  Content-Length: 22
  Content-Type: text/html
  [Body] 22 bytes
[Connection] Closed for 172.16.156.148:65289
[Connection] Closed for 172.16.156.148:65288
```

Figure 9(404 NOT FOUND)

2.6. When http://172.16.156.148:8080/file.unknownext is run on browser:

```
[Connection] New connection from 172.16.156.148:49242

[Request] GET /file.unknownext HTTP/1.1
[Client] 172.16.156.148:49242
[Header] Host: 172.16.156.148:8080
[Header] User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome
/135.0.0.0 Safari/537.36
[Header] Connection: keep-alive
[Path] /file.unknownext -> www/file.unknownext
[Error] Unknown MIME type for: www/file.unknownext

[Response Headers]
  HTTP/1.1 415 Unsupported Media Type
  Date: Sun, 27 Apr 2025 10:36:20 GMT
  Server: COMP2322/1.0
  Connection: keep-alive
  Content-Length: 35
  Content-Type: text/html
  [Body] 35 bytes
[Connection] New connection from 172.16.156.148:49243
[Connection] Closed for 172.16.156.148:49242
[Connection] Closed for 172.16.156.148:49243
```

Figure 10(415 UNSUPPORTED MEDIA TYPE)

2.7. When “curl -I http://localhost:8080/index.html” is run on terminal:

```
[Connection] New connection from 127.0.0.1:49422

[Request] HEAD /index.html HTTP/1.1
[Client] 127.0.0.1:49422
[Header] Host: localhost:8080
[Header] User-Agent: curl/8.7.1
[Path] /index.html -> www/index.html
[MIME] text/html
[Response] 200 OK (0 bytes, Last-Modified: Tue, 22 Apr 2025 09:33:31 GMT)

[Response Headers]
  HTTP/1.1 200 OK
  Date: Sun, 27 Apr 2025 10:45:18 GMT
  Server: COMP2322/1.0
  Connection: keep-alive
  Content-Length: 0
  Content-Type: text/html
  Last-Modified: Tue, 22 Apr 2025 09:33:31 GMT
  [Body] None
[Connection] Closed for 127.0.0.1:49422
```

Figure 11(Servers response for HEAD command)

```
cagancakir@Cagans-MacBook-Pro network % curl -I http://localhost:8080/index.html
HTTP/1.1 200 OK
Date: Sun, 27 Apr 2025 10:45:18 GMT
Server: COMP2322/1.0
Connection: keep-alive
Content-Length: 0
Content-Type: text/html
Last-Modified: Tue, 22 Apr 2025 09:33:31 GMT
```

Figure 12(Terminal output for HEAD command)

Server's Log File

```
cagancakir@Cagans-MacBook-Pro network % cat /Users/cagancakir/Desktop/network/server.log
# Server started at 2025-04-28 03:01:24.142202 on port 8080
172.16.156.148 - - [27/Apr/2025:19:01:55 +0000] "GET /index.html HTTP/1.1" 200 44
172.16.156.148 - - [27/Apr/2025:19:01:57 +0000] "GET /index.html HTTP/1.1" 304 0
172.16.137.135 - - [27/Apr/2025:19:02:58 +0000] "POST /index.html HTTP/1.1" 400 24
172.16.156.148 - - [27/Apr/2025:19:03:32 +0000] "GET /some_dir/../index.html HTTP/1.1" 403 22
172.16.156.148 - - [27/Apr/2025:19:03:47 +0000] "GET /nonexistent.html HTTP/1.1" 404 22
172.16.156.148 - - [27/Apr/2025:19:03:55 +0000] "GET /file.unknownext HTTP/1.1" 415 35
172.16.156.148 - - [27/Apr/2025:19:03:56 +0000] "GET /file.unknownext HTTP/1.1" 415 35
127.0.0.1 - - [27/Apr/2025:19:04:13 +0000] "HEAD /index.html HTTP/1.1" 200 0
```

Figure 13(Log File when the above commands are run)

ReadME

Multi-threaded HTTP Web Server

A lightweight, multi-threaded HTTP web server implemented in Python that can handle multiple client requests simultaneously.

Features

- Multi-threaded architecture for handling concurrent connections
- Support for HTTP/1.1 with keep-alive connections
- Handles GET and HEAD request methods
- Automatic MIME type detection for served files
- Proper handling of conditional requests (If-Modified-Since)
- Common Log Format (CLF) logging
- Path traversal protection
- Automatic port selection if default port is in use

Functions

- Multi-threaded Web server
- Proper request and response message exchanges
- GET command for both text files and image files
- HEAD command
- Six types of response statuses (200 OK, 400 Bad Request, 404 File Not Found, 304 Not Modified, 415 Unspupported Media Type, 401 Forbidden)
- Last-Modified and If-Modified-Since header fields
- Keep-Alive header field

Requirements

- Python 3.10+
- No external dependencies required (uses only standard library modules)

Usage

Starting the Server

- Put the website source files into folder - ./www

```
python network.py [port] [document_root]
```