



THE HONG KONG
POLYTECHNIC UNIVERSITY
香港理工大學

COMP2322 COMPUTER NETWORKING
SPRING 2024

**A Multi-Thread HTTP Web Server
Project Report**

Student:

Wang Ruijie 22103808D
Department of Computing
The Hong Kong Polytechnic University

April 16, 2024

Contents

1	Introduction	2
2	Design and Implementation	2
2.1	Platform, Language and Programming Conventions	2
2.2	Program Architecture	2
2.3	Thread and Connection Management	3
2.4	Request Parsing	4
2.5	GET/HEAD Request Handling	5
2.6	Log File Generation	7
3	Test Cases	7
3.1	Handling of GET Request for a new File	7
3.2	Handling of GET Request for an Unmodified File	8
3.3	Handling of GET Request for a Modified Request	9
3.4	Handling of GET Request for an Image File	11
3.5	Handling of HEAD Request for an Unmodified File	12
3.6	Handling of HEAD Request for a Modified File	13
3.7	Handling of Uncovered Request Methods	14
3.8	Handling of Request for a Non-Existent File	15
3.9	Multi-Thread Handling	16
3.10	Handling of Keep-Alive Connection	17
4	Conclusion	21

1 Introduction

The document is aimed at explaining and displaying the functions achieved for the programming project, which is a part of the PolyU COMP2322 Computer Networking, Spring 2024. In the project, a multi-thread HTTP web server that is able to concurrently handle multiple requests and return appropriate responses is properly implemented, along with the ability of handling the header fields, managing a thread list, displaying corresponding console output, and generating a log file.

The subsequent sections of this article provides detailed descriptions of the architecture of the program, the implementation of the aforementioned features, and the testing and evaluation conducted on these functionalities in a refined manner.

2 Design and Implementation

2.1 Platform, Language and Programming Conventions

Please note that the platform used may cause a fatal error when running the program because of the different format of requests. For example, if the program is run on Windows, the time stamp will be in the format of `Thu, 01 Jan 1970 00:00:00 GMT`, while on macOS, the time stamp will be in the format of `1970-01-01 00:00:00.000000`.

The program assumes that the server and the client are all running on the platform sending the first format given above. Meanwhile, it also assumes the first letters of `If-Modified-Since` and `Connection: Keep-Alive` in the request header are capitalized. `Connection: keep-alive` may also be accepted. Though in this report, the program is tested on macOS, the server and clients are expected to run on Windows or any operating system that supports the request formats mentioned above. When testing the handling of the `If-Modified-Since` header, a script, rather than a real web browser, is used to prevent the unsupported format.

The server is implemented and tested with Python 3.13.0a1. Only Python standard libraries `socket`, `os`, `time`, `datetime`, and `threading` are imported to the source code. Some new features of the new version of Python may be used, and they are not guaranteed to be compatible with the older versions.

For the programming conventions, the source code adapts the snake case naming convention for variables and functions with words written in lowercase and separated by underscores, and the name of the class is written in CamelCase. The code is well-structured with proper indentation and spacing, and the comments follows the *reStructuredText* format.

2.2 Program Architecture

A simple object-oriented design is used with a class of `HTTPServer`. The class contains the following members:

- `host` and `port`: the server address and port number.
- `threaded_connections`: a list of threads that are currently handling the requests.
- `server_socket`: the socket object of the server.
- `server_log`: the log file of the server.

- `__init__`: the constructor of the class, initializing the server address, port, and the thread pool size.
- `run`: the main function of the server, creating a server socket, listening to the incoming connections, and handling the requests.
- `receive`: parse the incoming request and pass the parsed arguments to different handlers.
- `response_GET` and `response_HEAD`: handle the GET and HEAD requests, respectively.
- `response_file_not_found` and `response_bad_request`: handle the mentioned errors.
- `response_log`: generate the log file.
- other helper functions that are repeatedly called for the logics.

Once the program is executed, the server will start listening to the incoming connections on the address and port specified (127.0.0.1 and 8000 by default). When a connection is established, a new thread will be created to handle the request from the connection. Afterwards, the server will parse the request and generate appropriate behavior of response accordingly along with the mentioned functions.

2.3 Thread and Connection Management

The class of `HTTPServer` contains a list named `threaded_connections` to manage the threads that are currently handling the requests. Iteratively, until being interrupted, a new thread will be created and appended to the list while a new connection is established to execute the logic in the `receive` function.

```

1 # Defined in the run() function,
2 # that is, the main function of the server
3 while True:
4     connection, address = self.server_socket.accept()
5     thread = threading.Thread(target = self.receive,
6         args = (connection, address))
7     thread.start()
8     self.threaded_connections.append(connection)

```

The method of terminating the program is to press `Ctrl + C` in the terminal to generate the `KeyboardInterrupt`. When the interrupt is fetched, the main thread will stop and wait for all other current threads to exit, and then the program will be terminated.

```

1 # Defined in the run() function,
2 except KeyboardInterrupt:
3     """
4     Fetch the KeyboardInterrupt and print a message.....
5     """
6 finally:
7     """
8     Close the server socket and log file
9     Close the connections in the list of threaded connections

```

```

10     """
11     self.server_log.close()
12     self.server_socket.close()
13     for connection in self.threaded_connections:
14         connection.close()

```

However, since Python does not support the termination of threads that is outside the thread itself, a timeout mechanism is used to make sure the threads will exit in a reasonable time, which also corresponds to the requirement of implementing the **keep-alive** feature in the web server. When the thread finishes handling the request, that is, the client disconnects from the server due to timeout or the connection is completed, it will be removed from the list, and the thread will be terminated. In that way, the server can handle multiple requests concurrently and exit gracefully.

```

1  # Defined in the receive function,
2  # that is, for each thread handling the request
3  except socket.timeout:
4      """
5      If the connection is keep-alive and the timeout is reached,
6      the server closes the connection
7      Print messages...
8      """
9      connection.close()
10     break
11 except KeyboardInterrupt:
12     """
13     If the server is interrupted, the server closes the connection,
14     and removes it from the list of threaded connections
15     """
16     self.threaded_connections.remove(connection)
17     break

```

If the connection is not keep-alive, the thread will exit directly after the request is handled, and the connection will be closed and removed from the list.

2.4 Request Parsing

At first, the server will detect whether there exists **keep-alive** or **Keep-Alive** in the request header, and use a boolean flag to denote the status of the connection.

```

1  # Defined in the receive function
2  """
3  The server checks if the connection is keep-alive or not
4  """
5  keep_alive = False
6  for line in request.split("\n"):
7      if ("Connection: keep-alive" in line)
8          or ("Connection: Keep-Alive" in line):
9          keep_alive = True
10         break

```

After that, the server will parse the request and extract the path and method of the request.

```
1 """
2 Parse the request into method and url
3 """
4 request_method = request.split("\n")[0].split(" ")[0]
5 request_url = request.split("\n")[0].split(" ")[1]
```

According to the method, the server will call the corresponding handler to handle the request. If the method is not GET or HEAD, the server will return a 400 Bad Request response. However, if the method can not be accepted, the server will let the bad request handler to handle the request.

```
1 #Defined in the receive function
2 """
3 The server calls the appropriate response function,
4 based on the method of the request
5 """
6 if (request_method == "GET"):
7     self.response_GET(connection, request,
8         request_url, address)
9 elif (request_method == "HEAD"):
10     self.response_HEAD(connection, request,
11         request_url, address)
12 else:
13     self.response_bad_request(connection, request_url, address)
```

2.5 GET/HEAD Request Handling

The handling of the GET and HEAD requests are quite similar, except the content to be sent to the connection. The server will examine the type of the requested file by calling a helper function that checks the extension of the file, which determines the way of reading (binary or text) and sending (decode the binary content or not) the file.

```
1 # Defined in the HTTPServer class
2 def get_file_type(self, request_url):
3     """
4     This function returns the type of the file requested
5     :request_url: url of the request
6     """
7     if request_url.endswith(".html"):
8         return "text/html"
9     elif request_url.endswith(".jpg"):
10         return "image/jpg"
11     elif request_url.endswith(".png"):
12         return "image/png"
13     else:
14         return "text/plain"
```

After that, to handle the If-Modified-Since header field, the server first examines whether the field exists in the request header, and then compares the time of the last modification of the file with the time in the field. If the file is not modified, the server will return a 304 Not Modified response. Otherwise, the server will read

the file and send the content to the connection. If there is no If-Modified-Since header, the server will send the file directly as well.

```
1 # Defined in the response_GET function
2 if ("If-Modified-Since" in request):
3
4     for line in request.split("\n"):
5         if "If-Modified-Since" in line:
6             check_time = line[19:48]
7             break;
8
9     check_time = datetime.strptime(check_time,
10                                     '%a, %d %b %Y %H:%M:%S %Z')
11     file_time =
12         datetime.fromtimestamp(os.path.getmtime("htdocs"
13                                                 + request_url))
14
15     if (file_time <= check_time):
16
17         response_parts = ["HTTP/1.1 304 Not Modified\r\n",
18                           "Last-Modified: " +
19                               str(datetime.fromtimestamp(
20                                   os.path.getmtime("htdocs" +
21                                                   request_url))) + "\r\n",
22                           "Content-Length: " +
23                               str(len(contents)) + "\r\n",
24                           "Content-Type: " +
25                               file_type + "\r\n\r\n"]
26         response = "".join(response_parts)
27         connection.sendall(response.encode())
28
29         print(response)
30         self.write_log(address, request_url, "304 Not Modified")
31
32     else:
33
34         response_parts = ["HTTP/1.1 200 OK\r\n",
35                           "Last-Modified: " +
36                               str(datetime.fromtimestamp(os.path.getmtime("htdocs"
37                                                   + request_url))) + "\r\n",
38                           "Content-Length: " +
39                               str(len(contents)) + "\r\n",
40                           "Content-Type: " +
41                               file_type + "\r\n\r\n"]
42         response = "".join(response_parts)
43         connection.sendall(response.encode())
44
45         print("Server response:\n")
46         print(response)
47         self.write_log(address, request_url, "200 OK")
48 else:
49     # Same as above
```

For the HEAD request, the server will not send the content of the file, but only

the header fields. If the file is not found, the server will return a 404 Not Found response.

2.6 Log File Generation

Note that for each response of the server, the server will generate a log file that records the time of the request, the address of the client, the requested file, and the status of the response, by calling the `write_log` function below. The log file will be closed when the server is terminated.

```
1 # Defined in the HTTPServer class
2 def write_log(self, address, file_name, response_type):
3     """
4     This function writes the request and response to the log file
5     :address: address of the client
6     :file_name: name of the requested file
7     :response_type: type of the response
8     :writing_parts: parts of the log message
9     :writing: log message
10    """
11
12    writing_parts = [str(datetime.now()),
13                    ": client", str(address),
14                    " requested file ", file_name,
15                    " with response ",
16                    response_type, "\n"]
17    writing = "".join(writing_parts)
18    self.server_log.write(writing)
19    self.server_log.flush()
```

3 Test Cases

Please refer to the section 2.1 for the platform and the environment used for testing. The cases below might not be applicable to the other platforms.

3.1 Handling of GET Request for a new File

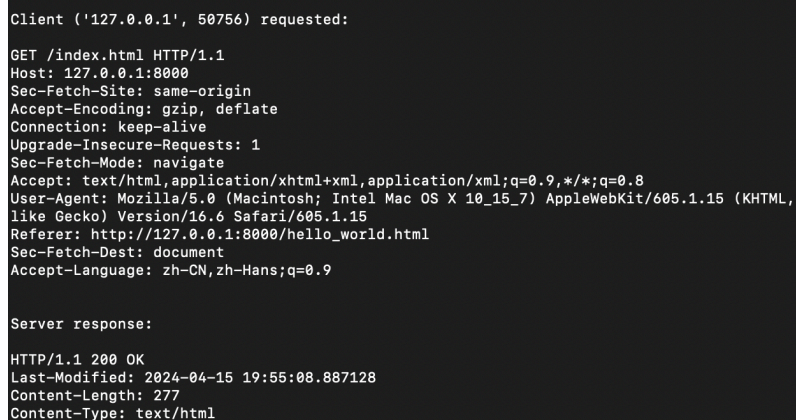
By cleaning the web cache and running the server and visit the web page by typing `http://127.0.0.1:8000` in the browser, the server will return the content of the `index.html` file in the `htdocs` directory successfully.



The client request and server response are as follows:

```
1 Client ('127.0.0.1', 50756) requested:
2
3 GET /index.html HTTP/1.1
4 Host: 127.0.0.1:8000
5 Sec-Fetch-Site: same-origin
6 Accept-Encoding: gzip, deflate
7 Connection: keep-alive
8 Upgrade-Insecure-Requests: 1
9 Sec-Fetch-Mode: navigate
10 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
11 User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)
12     AppleWebKit/605.1.15 (KHTML, like Gecko) Version/16.6 Safari/605.1.15
13 Referer: http://127.0.0.1:8000/hello_world.html
14 Sec-Fetch-Dest: document
15 Accept-Language: zh-CN,zh-Hans;q=0.9
16
17
18 Server response:
19
20 HTTP/1.1 200 OK
21 Last-Modified: 2024-04-15 19:55:08.887128
22 Content-Length: 277
23 Content-Type: text/html
```

It can be seen that the server returns a 200 OK response. Screen capture of the console output:



```
Client ('127.0.0.1', 50756) requested:
GET /index.html HTTP/1.1
Host: 127.0.0.1:8000
Sec-Fetch-Site: same-origin
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Sec-Fetch-Mode: navigate
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/605.1.15 (KHTML,
like Gecko) Version/16.6 Safari/605.1.15
Referer: http://127.0.0.1:8000/hello_world.html
Sec-Fetch-Dest: document
Accept-Language: zh-CN,zh-Hans;q=0.9

Server response:
HTTP/1.1 200 OK
Last-Modified: 2024-04-15 19:55:08.887128
Content-Length: 277
Content-Type: text/html
```

3.2 Handling of GET Request for an Unmodified File

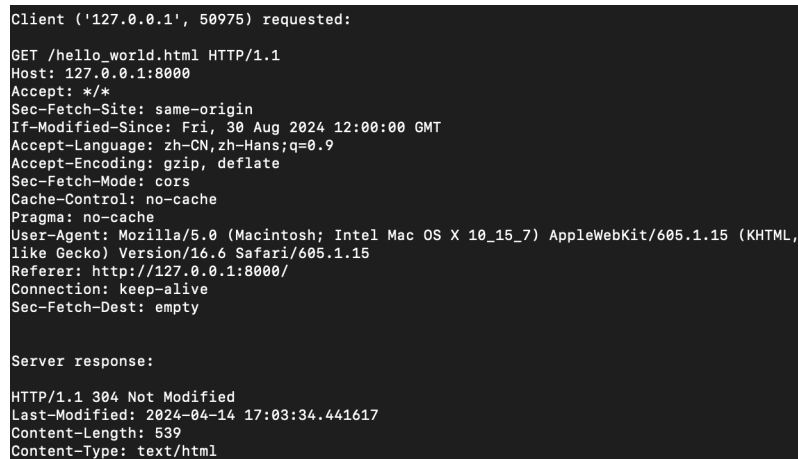
Again, but not cleaning the web cache, the server will return a 304 Not Modified response, indicating that the file is not modified. On macOS, you may use the following script to test the feature:

```
1 const headers = new Headers();
2 headers.append('If-Modified-Since',
3     'Fri, 30 Aug 2024 12:00:00 GMT');
4 fetch('http://127.0.0.1:8000/hello_world.html',
5     { method: 'GET', headers: headers})
```

A time stamp in the future, i.e., Fri, 30 Aug 2024 12:00:00 GMT, is used to test the feature. The client request and server response are as follows:

```
1 Client ('127.0.0.1', 50975) requested:
2
3 GET /hello_world.html HTTP/1.1
4 Host: 127.0.0.1:8000
5 Accept: */*
6 Sec-Fetch-Site: same-origin
7 If-Modified-Since: Fri, 30 Aug 2024 12:00:00 GMT
8 Accept-Language: zh-CN,zh-Hans;q=0.9
9 Accept-Encoding: gzip, deflate
10 Sec-Fetch-Mode: cors
11 Cache-Control: no-cache
12 Pragma: no-cache
13 User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)
14     AppleWebKit/605.1.15 (KHTML, like Gecko) Version/16.6 Safari/605.1.15
15 Referer: http://127.0.0.1:8000/
16 Connection: keep-alive
17 Sec-Fetch-Dest: empty
18
19
20 Server response:
21
22 HTTP/1.1 304 Not Modified
23 Last-Modified: 2024-04-14 17:03:34.441617
24 Content-Length: 539
25 Content-Type: text/html
```

It can be seen that the server returns a 304 Not Modified response for the future time stamp. The screen capture of the console output is as follows:



```
Client ('127.0.0.1', 50975) requested:
GET /hello_world.html HTTP/1.1
Host: 127.0.0.1:8000
Accept: */*
Sec-Fetch-Site: same-origin
If-Modified-Since: Fri, 30 Aug 2024 12:00:00 GMT
Accept-Language: zh-CN,zh-Hans;q=0.9
Accept-Encoding: gzip, deflate
Sec-Fetch-Mode: cors
Cache-Control: no-cache
Pragma: no-cache
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/605.1.15 (KHTML,
like Gecko) Version/16.6 Safari/605.1.15
Referer: http://127.0.0.1:8000/
Connection: keep-alive
Sec-Fetch-Dest: empty

Server response:
HTTP/1.1 304 Not Modified
Last-Modified: 2024-04-14 17:03:34.441617
Content-Length: 539
Content-Type: text/html
```

3.3 Handling of GET Request for a Modified Request

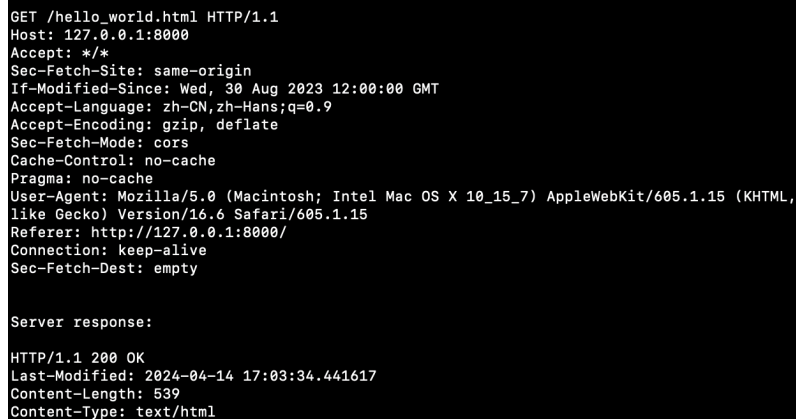
A past time stamp is used, i.e., Wed, 30 Aug 2023 12:00:00 GMT. The script used is as follows:

```
1 const headers = new Headers();
2 headers.append('If-Modified-Since', 'Wed, 30 Aug 2023 12:00:00 GMT');
3 fetch('http://127.0.0.1:8000/hello_world.html',
4     { method: 'GET', headers: headers})
```

The client request and server response are as follows:

```
1 Client ('127.0.0.1', 51001) requested:
2
3 GET /hello_world.html HTTP/1.1
4 Host: 127.0.0.1:8000
5 Accept: */*
6 Sec-Fetch-Site: same-origin
7 If-Modified-Since: Wed, 30 Aug 2023 12:00:00 GMT
8 Accept-Language: zh-CN,zh-Hans;q=0.9
9 Accept-Encoding: gzip, deflate
10 Sec-Fetch-Mode: cors
11 Cache-Control: no-cache
12 Pragma: no-cache
13 User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)
14     AppleWebKit/605.1.15 (KHTML, like Gecko) Version/16.6 Safari/605.1.15
15 Referer: http://127.0.0.1:8000/
16 Connection: keep-alive
17 Sec-Fetch-Dest: empty
18
19
20 Server response:
21
22 HTTP/1.1 200 OK
23 Last-Modified: 2024-04-14 17:03:34.441617
24 Content-Length: 539
25 Content-Type: text/html
```

The server also returns a 200 OK response. The screen capture of the console output is as follows:



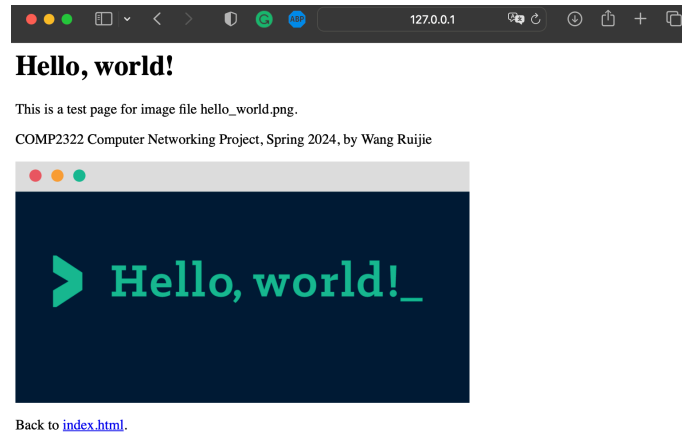
```
GET /hello_world.html HTTP/1.1
Host: 127.0.0.1:8000
Accept: */*
Sec-Fetch-Site: same-origin
If-Modified-Since: Wed, 30 Aug 2023 12:00:00 GMT
Accept-Language: zh-CN,zh-Hans;q=0.9
Accept-Encoding: gzip, deflate
Sec-Fetch-Mode: cors
Cache-Control: no-cache
Pragma: no-cache
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/605.1.15 (KHTML,
like Gecko) Version/16.6 Safari/605.1.15
Referer: http://127.0.0.1:8000/
Connection: keep-alive
Sec-Fetch-Dest: empty

Server response:

HTTP/1.1 200 OK
Last-Modified: 2024-04-14 17:03:34.441617
Content-Length: 539
Content-Type: text/html
```

3.4 Handling of GET Request for an Image File

By visiting the `hello_world.html` web page, an image object is requested.

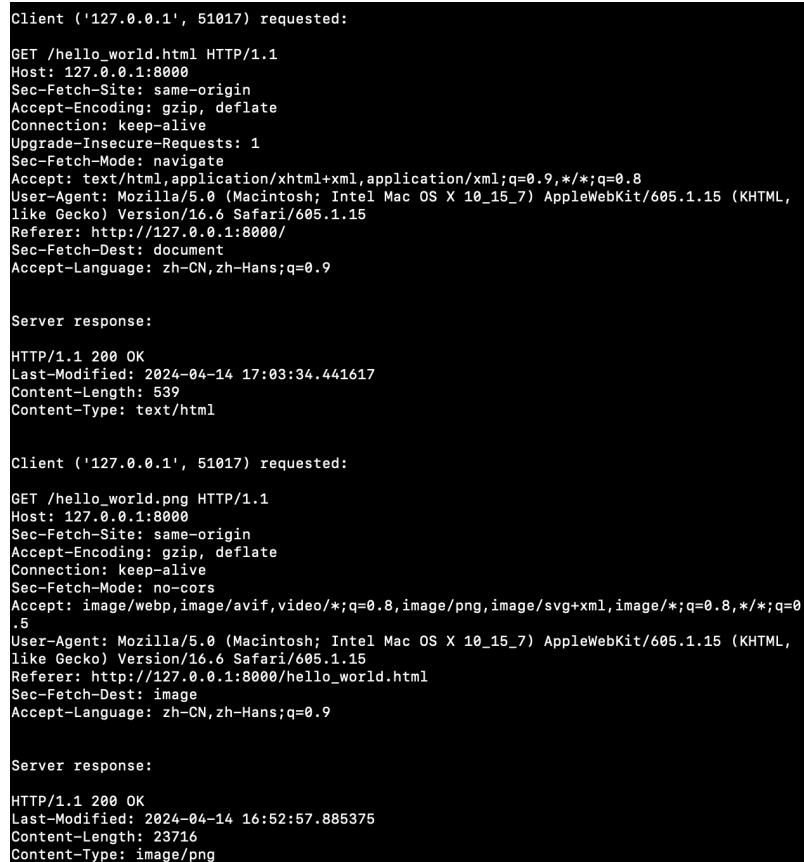


The client request and server response are as follows:

```
1 Client ('127.0.0.1', 51017) requested:
2
3 GET /hello_world.html HTTP/1.1
4 Host: 127.0.0.1:8000
5 Sec-Fetch-Site: same-origin
6 Accept-Encoding: gzip, deflate
7 Connection: keep-alive
8 Upgrade-Insecure-Requests: 1
9 Sec-Fetch-Mode: navigate
10 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
11 User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)
12     AppleWebKit/605.1.15 (KHTML, like Gecko) Version/16.6 Safari/605.1.15
13 Referer: http://127.0.0.1:8000/
14 Sec-Fetch-Dest: document
15 Accept-Language: zh-CN,zh-Hans;q=0.9
16
17
18 Server response:
19
20 HTTP/1.1 200 OK
21 Last-Modified: 2024-04-14 17:03:34.441617
22 Content-Length: 539
23 Content-Type: text/html
24
25
26 Client ('127.0.0.1', 51017) requested:
27
28 GET /hello_world.png HTTP/1.1
29 Host: 127.0.0.1:8000
30 Sec-Fetch-Site: same-origin
31 Accept-Encoding: gzip, deflate
32 Connection: keep-alive
33 Sec-Fetch-Mode: no-cors
34 Accept: image/webp,image/avif,image/svg+xml,image/*;q=0.8,*/*;q=0.1
35 User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)
36     AppleWebKit/605.1.15 (KHTML, like Gecko) Version/16.6 Safari/605.1.15
```

```
37 Referer: http://127.0.0.1:8000/hello_world.html
38 Sec-Fetch-Dest: image
39 Accept-Language: zh-CN,zh-Hans;q=0.9
40
41
42 Server response:
43
44 HTTP/1.1 200 OK
45 Last-Modified: 2024-04-14 16:52:57.885375
46 Content-Length: 23716
47 Content-Type: image/png
```

According to the client request, the server returns a 200 OK response for the image file `hello_world.png`. The screen capture of the console output is as follows:



```
Client ('127.0.0.1', 51017) requested:
GET /hello_world.html HTTP/1.1
Host: 127.0.0.1:8000
Sec-Fetch-Site: same-origin
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Sec-Fetch-Mode: navigate
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/16.6 Safari/605.1.15
Referer: http://127.0.0.1:8000/
Sec-Fetch-Dest: document
Accept-Language: zh-CN,zh-Hans;q=0.9

Server response:
HTTP/1.1 200 OK
Last-Modified: 2024-04-14 17:03:34.441617
Content-Length: 539
Content-Type: text/html

Client ('127.0.0.1', 51017) requested:
GET /hello_world.png HTTP/1.1
Host: 127.0.0.1:8000
Sec-Fetch-Site: same-origin
Accept-Encoding: gzip, deflate
Connection: keep-alive
Sec-Fetch-Mode: no-cors
Accept: image/webp,image/avif,image/*;q=0.8,image/png,image/svg+xml,image/*;q=0.8,*/*;q=0.5
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/16.6 Safari/605.1.15
Referer: http://127.0.0.1:8000/hello_world.html
Sec-Fetch-Dest: image
Accept-Language: zh-CN,zh-Hans;q=0.9

Server response:
HTTP/1.1 200 OK
Last-Modified: 2024-04-14 16:52:57.885375
Content-Length: 23716
Content-Type: image/png
```

3.5 Handling of HEAD Request for an Unmodified File

Unlike the GET request, the HEAD request is not convenient to test in the browser. Hence, the following script is used. Please note that the script also tests the handling of the If-Modified-Since header field under the HEAD request.

```
1 const headers = new Headers();
2 headers.append('If-Modified-Since', 'Fri, 30 Aug 2024 12:00:00 GMT');
3 fetch('http://127.0.0.1:8000/hello_world.html',
4     { method: 'HEAD', headers: headers})
```

The client request and server response are as follows:

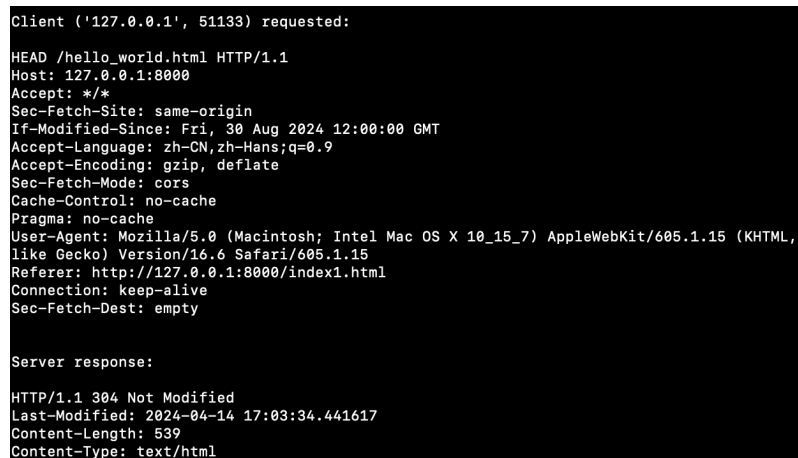
```
1 Client ('127.0.0.1', 51133) requested:
2
```

```

3 HEAD /hello_world.html HTTP/1.1
4 Host: 127.0.0.1:8000
5 Accept: */*
6 Sec-Fetch-Site: same-origin
7 If-Modified-Since: Fri, 30 Aug 2024 12:00:00 GMT
8 Accept-Language: zh-CN,zh-Hans;q=0.9
9 Accept-Encoding: gzip, deflate
10 Sec-Fetch-Mode: cors
11 Cache-Control: no-cache
12 Pragma: no-cache
13 User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/605.1.
14 Referer: http://127.0.0.1:8000/index1.html
15 Connection: keep-alive
16 Sec-Fetch-Dest: empty
17
18
19 Server response:
20
21 HTTP/1.1 304 Not Modified
22 Last-Modified: 2024-04-14 17:03:34.441617
23 Content-Length: 539
24 Content-Type: text/html

```

Since the script is with a future time stamp, the server returns a 304 Not Modified response. The screen capture of the console output is as follows:



```

Client ('127.0.0.1', 51133) requested:

HEAD /hello_world.html HTTP/1.1
Host: 127.0.0.1:8000
Accept: */*
Sec-Fetch-Site: same-origin
If-Modified-Since: Fri, 30 Aug 2024 12:00:00 GMT
Accept-Language: zh-CN,zh-Hans;q=0.9
Accept-Encoding: gzip, deflate
Sec-Fetch-Mode: cors
Cache-Control: no-cache
Pragma: no-cache
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/605.1.15 (KHTML,
like Gecko) Version/16.6 Safari/605.1.15
Referer: http://127.0.0.1:8000/index1.html
Connection: keep-alive
Sec-Fetch-Dest: empty

Server response:

HTTP/1.1 304 Not Modified
Last-Modified: 2024-04-14 17:03:34.441617
Content-Length: 539
Content-Type: text/html

```

3.6 Handling of HEAD Request for a Modified File

The script used with a past time stamp is as follows:

```

1 const headers = new Headers();
2 headers.append('If-Modified-Since', 'Wed, 30 Aug 2023 GMT');
3 fetch('http://127.0.0.1:8000/hello_world.html',
4       { method: 'HEAD', headers: headers})

```

The client request and server response are as follows:

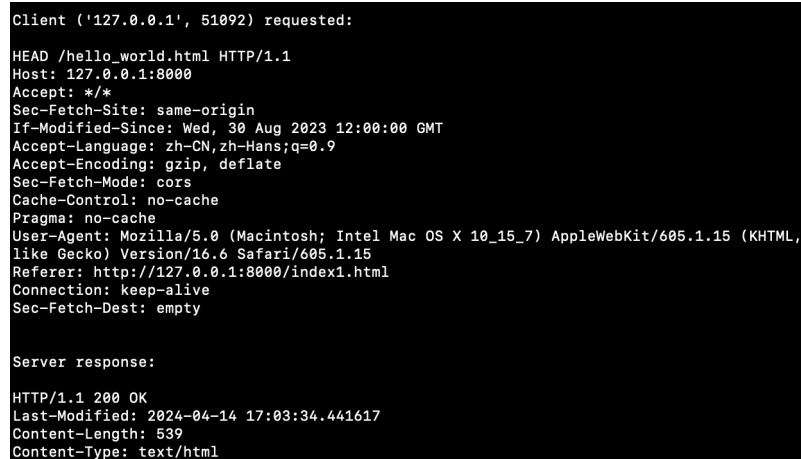
```

1 HEAD /hello_world.html HTTP/1.1
2 Host: 127.0.0.1:8000
3 Accept: */*
4 Sec-Fetch-Site: same-origin

```

```
5 If-Modified-Since: Wed, 30 Aug 2023 12:00:00 GMT
6 Accept-Language: zh-CN,zh-Hans;q=0.9
7 Accept-Encoding: gzip, deflate
8 Sec-Fetch-Mode: cors
9 Cache-Control: no-cache
10 Pragma: no-cache
11 User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)
12     AppleWebKit/605.1.15 (KHTML, like Gecko) Version/16.6 Safari/605.1.15
13 Referer: http://127.0.0.1:8000/index1.html
14 Connection: keep-alive
15 Sec-Fetch-Dest: empty
16
17
18 Server response:
19
20 HTTP/1.1 200 OK
21 Last-Modified: 2024-04-14 17:03:34.441617
22 Content-Length: 539
23 Content-Type: text/html
```

The server returns a 200 OK response for the HEAD request. The screen capture of the console output is as follows:



```
Client ('127.0.0.1', 51092) requested:
HEAD /hello_world.html HTTP/1.1
Host: 127.0.0.1:8000
Accept: */*
Sec-Fetch-Site: same-origin
If-Modified-Since: Wed, 30 Aug 2023 12:00:00 GMT
Accept-Language: zh-CN,zh-Hans;q=0.9
Accept-Encoding: gzip, deflate
Sec-Fetch-Mode: cors
Cache-Control: no-cache
Pragma: no-cache
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/605.1.15 (KHTML,
like Gecko) Version/16.6 Safari/605.1.15
Referer: http://127.0.0.1:8000/index1.html
Connection: keep-alive
Sec-Fetch-Dest: empty

Server response:

HTTP/1.1 200 OK
Last-Modified: 2024-04-14 17:03:34.441617
Content-Length: 539
Content-Type: text/html
```

3.7 Handling of Uncovered Request Methods

The following script is used to test the handling of the uncovered request method, i.e., POST.

```
1 fetch('http://127.0.0.1:8000/', { method: 'POST' })
```

The client request and server response are as follows:

```
1 Client ('127.0.0.1', 51156) requested:
2
3 POST / HTTP/1.1
4 Host: 127.0.0.1:8000
5 Sec-Fetch-Site: same-origin
6 Accept-Encoding: gzip, deflate
7 Accept-Language: zh-CN,zh-Hans;q=0.9
8 Sec-Fetch-Mode: cors
9 Accept: */*
```

```

10 Origin: http://127.0.0.1:8000
11 Content-Length: 0
12 Connection: keep-alive
13 User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/605.1.
14 Referer: http://127.0.0.1:8000/index1.html
15 Sec-Fetch-Dest: empty
16
17
18 Server response:
19
20 HTTP/1.1 400 Bad Request
21 Content-Length : 0
22 Content-Type: text/plain

```

The server returns a 400 Bad Request response for the POST request. The screen capture of the console output is as follows:

```

Client ('127.0.0.1', 51156) requested:

POST / HTTP/1.1
Host: 127.0.0.1:8000
Sec-Fetch-Site: same-origin
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh-Hans;q=0.9
Sec-Fetch-Mode: cors
Accept: */*
Origin: http://127.0.0.1:8000
Content-Length: 0
Connection: keep-alive
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/605.1.15 (KHTML,
like Gecko) Version/16.6 Safari/605.1.15
Referer: http://127.0.0.1:8000/index1.html
Sec-Fetch-Dest: empty

Server response:

HTTP/1.1 400 Bad Request
Content-Length : 0
Content-Type: text/plain

```

3.8 Handling of Request for a Non-Existent File

By entering `http://127.0.0.1:8000/a.html` in the browser, the server will return a 404 Not Found response.

```

1 Client ('127.0.0.1', 51165) requested:
2
3 GET /a.html HTTP/1.1
4 Host: 127.0.0.1:8000
5 Sec-Fetch-Site: none
6 Connection: keep-alive
7 Upgrade-Insecure-Requests: 1
8 Sec-Fetch-Mode: navigate
9 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
10 User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)
11     AppleWebKit/605.1.15 (KHTML, like Gecko) Version/16.6 Safari/605.1.15
12 Accept-Language: zh-CN,zh-Hans;q=0.9
13 Sec-Fetch-Dest: document
14 Accept-Encoding: gzip, deflate
15
16
17 Server response:
18
19 HTTP/1.1 404 Not Found

```



```
20 Content-Length : 0
21 Content-Type: text/plain
```

The server returns a 404 Not Found response for the non-existent HTML file. The screen capture of the console output is as follows:

```
Client ('127.0.0.1', 51165) requested:
GET /a.html HTTP/1.1
Host: 127.0.0.1:8000
Sec-Fetch-Site: none
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Sec-Fetch-Mode: navigate
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/16.6 Safari/605.1.15
Accept-Language: zh-CN,zh-Hans;q=0.9
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate

Server response:
HTTP/1.1 404 Not Found
Content-Length : 0
Content-Type: text/plain
```

3.9 Multi-Thread Handling

By visiting the web page <http://127.0.0.1:8000> in different browsers or different tabs, the server will handle the requests concurrently. We can see that the server can handle multiple requests at the same time.

```
1 Client ('127.0.0.1', 51215) requested:
2
3 GET / HTTP/1.1
4 Host: 127.0.0.1:8000
5 Connection: keep-alive
6 sec-ch-ua: "Google Chrome";v="123", "Not:A-Brand";v="8", "Chromium";v="123"
7 sec-ch-ua-mobile: ?0
8 sec-ch-ua-platform: "macOS"
9 Upgrade-Insecure-Requests: 1
10 User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)
11     AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.0.0 Safari/537.36
12 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image
13     image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
14 Sec-Fetch-Site: none
15 Sec-Fetch-Mode: navigate
16 Sec-Fetch-User: ?1
17 Sec-Fetch-Dest: document
18 Accept-Encoding: gzip, deflate, br, zstd
19 Accept-Language: zh-CN,zh;q=0.9
20
21
22 Server response:
23
24 HTTP/1.1 200 OK
25 Last-Modified: 2024-04-15 19:55:08.887128
26 Content-Length: 277
27 Content-Type: text/html
28
29
30 Client ('127.0.0.1', 51218) requested:
```

```

31
32 GET / HTTP/1.1
33 Host: 127.0.0.1:8000
34 Sec-Fetch-Site: none
35 Connection: keep-alive
36 Upgrade-Insecure-Requests: 1
37 Sec-Fetch-Mode: navigate
38 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
39 User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)
40     AppleWebKit/605.1.15 (KHTML, like Gecko) Version/16.6 Safari/605.1.15
41 Accept-Language: zh-CN,zh-Hans;q=0.9
42 Sec-Fetch-Dest: document
43 Accept-Encoding: gzip, deflate
44
45
46 Server response:
47
48 HTTP/1.1 200 OK
49 Last-Modified: 2024-04-15 19:55:08.887128
50 Content-Length: 277
51 Content-Type: text/html

```

Google Chrome and Safari are used to visit the web page concurrently. The server returns a 200 OK response for both requests. The screen capture of the console output is as follows:

```

Client ('127.0.0.1', 51215) requested:

GET / HTTP/1.1
Host: 127.0.0.1:8000
Connection: keep-alive
sec-ch-ua: "Google Chrome";v="123", "Not:A-Brand";v="8", "Chromium";v="123"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "macOS"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, li
ke Gecko) Chrome/123.0.0.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image
/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: zh-CN,zh;q=0.9

Server response:

HTTP/1.1 200 OK
Last-Modified: 2024-04-15 19:55:08.887128
Content-Length: 277
Content-Type: text/html

Client ('127.0.0.1', 51218) requested:

GET / HTTP/1.1
Host: 127.0.0.1:8000
Sec-Fetch-Site: none
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Sec-Fetch-Mode: navigate
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/605.1.15 (KHTML,
like Gecko) Version/16.6 Safari/605.1.15
Accept-Language: zh-CN,zh-Hans;q=0.9
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate

```

3.10 Handling of Keep-Alive Connection

Normally, the browser will send a request with the `Connection: Keep-Alive` header field. Since the server adapts the timeout mechanism to handle the `keep-alive` connection, if the client requests a keep-alive connection, the message indicating the

client disconnects will be printed after the timeout is reached, and the client is able to send multiple requests to the server without multiple messages are printed in the console. To test this, the `index.html` and `hello_world.html` web pages are visited in the browser sequentially within the same time slot.

```
1 Client ('127.0.0.1', 51269) requested:
2
3 GET / HTTP/1.1
4 Host: 127.0.0.1:8000
5 Connection: keep-alive
6 Cache-Control: max-age=0
7 sec-ch-ua: "Google Chrome";v="123", "Not:A-Brand";v="8", "Chromium";v="123"
8 sec-ch-ua-mobile: ?0
9 sec-ch-ua-platform: "macOS"
10 Upgrade-Insecure-Requests: 1
11 User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)
12     AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.0.0 Safari/537.36
13 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,
14     image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b
15 Sec-Fetch-Site: none
16 Sec-Fetch-Mode: navigate
17 Sec-Fetch-User: ?1
18 Sec-Fetch-Dest: document
19 Accept-Encoding: gzip, deflate, br, zstd
20 Accept-Language: zh-CN,zh;q=0.9
21
22
23 Server response:
24
25 HTTP/1.1 200 OK
26 Last-Modified: 2024-04-15 19:55:08.887128
27 Content-Length: 277
28 Content-Type: text/html
29
30
31 Client ('127.0.0.1', 51269) requested:
32
33 GET /hello_world.html HTTP/1.1
34 Host: 127.0.0.1:8000
35 Connection: keep-alive
36 sec-ch-ua: "Google Chrome";v="123", "Not:A-Brand";v="8", "Chromium";v="123"
37 sec-ch-ua-mobile: ?0
38 sec-ch-ua-platform: "macOS"
39 Upgrade-Insecure-Requests: 1
40 User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)
41     AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.0.0 Safari/537.36
42 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,
43     image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
44 Sec-Fetch-Site: same-origin
45 Sec-Fetch-Mode: navigate
46 Sec-Fetch-User: ?1
47 Sec-Fetch-Dest: document
48 Referer: http://127.0.0.1:8000/
49 Accept-Encoding: gzip, deflate, br, zstd
```

```

50 Accept-Language: zh-CN,zh;q=0.9
51
52
53 Server response:
54
55 HTTP/1.1 200 OK
56 Last-Modified: 2024-04-14 17:03:34.441617
57 Content-Length: 539
58 Content-Type: text/html
59
60
61 Client ('127.0.0.1', 51269) requested:
62
63 GET /hello_world.png HTTP/1.1
64 Host: 127.0.0.1:8000
65 Connection: keep-alive
66 sec-ch-ua: "Google Chrome";v="123", "Not:A-Brand";v="8", "Chromium";v="123"
67 sec-ch-ua-mobile: ?0
68 User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)
69     AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.0.0 Safari/537.36
70 sec-ch-ua-platform: "macOS"
71 Accept: image/avif,image/webp,image/apng,image/svg+xml,image/*,*/*;q=0.8
72 Sec-Fetch-Site: same-origin
73 Sec-Fetch-Mode: no-cors
74 Sec-Fetch-Dest: image
75 Referer: http://127.0.0.1:8000/hello_world.html
76 Accept-Encoding: gzip, deflate, br, zstd
77 Accept-Language: zh-CN,zh;q=0.9
78
79
80 Server response:
81
82 HTTP/1.1 200 OK
83 Last-Modified: 2024-04-14 16:52:57.885375
84 Content-Length: 23716
85 Content-Type: image/png
86
87
88
89 *****
90 Client ('127.0.0.1', 51270) disconnected: keep-alive timeout
91 *****
92
93
94 *****
95 Client ('127.0.0.1', 51269) disconnected: keep-alive timeout
96 *****

```

The screen capture of the console output is as follows:

```

Client ('127.0.0.1', 51269) requested:
GET /hello_world.html HTTP/1.1
Host: 127.0.0.1:8000
Connection: keep-alive
sec-ch-ua: "Google Chrome";v="123", "Not:A-Brand";v="8", "Chromium";v="123"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "macOS"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://127.0.0.1:8000/
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: zh-CN,zh;q=0.9

Server response:
HTTP/1.1 200 OK
Last-Modified: 2024-04-14 17:03:34.441617
Content-Length: 539
Content-Type: text/html

Client ('127.0.0.1', 51269) requested:
GET /hello_world.png HTTP/1.1
Host: 127.0.0.1:8000
Connection: keep-alive
sec-ch-ua: "Google Chrome";v="123", "Not:A-Brand";v="8", "Chromium";v="123"
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.0 Safari/537.36
sec-ch-ua-platform: "macOS"
Accept: image/avif,image/webp,image/apng,image/svg+xml,image/*,*/*;q=0.8
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: no-cors
Sec-Fetch-Dest: image
Referer: http://127.0.0.1:8000/hello_world.html
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: zh-CN,zh;q=0.9

Server response:
HTTP/1.1 200 OK
Last-Modified: 2024-04-14 16:52:57.885375
Content-Length: 23716
Content-Type: image/png

*****
Client ('127.0.0.1', 51270) disconnected: keep-alive timeout
*****

*****
Client ('127.0.0.1', 51269) disconnected: keep-alive timeout
*****

```

The browser sends multiple requests before timeout, which proves that the **Keep-Alive** feature is implemented successfully. However, if the request does not contain the **Connection: Keep-Alive** header field, the server will close the connection immediately after the request is handled. For example, if such request is sent via a client program, the server will close the connection after the request is sent. Please note the difference of the message printed in the console, which shows that the client disconnects because of **Connection: Close** rather than timeout.

```
1 "GET /a.html HTTP:/1.1\r\nConnection: Close\r\n"
```

The output is as follows:

```

1
2 Client ('127.0.0.1', 51421) requested:
3
4 GET /a.html HTTP:/1.1
5 Connection: Close
6
7 Server response:
8
9 HTTP/1.1 404 Not Found
10 Content-Length : 0
11 Content-Type: text/plain
12
13
14 *****
15 Client('127.0.0.1', 51421) disconnected: no keep-alive
16 *****

```

The screen capture of the console output is as follows:

```
Client ('127.0.0.1', 51462) requested:
GET /a.html HTTP/1.1
Connection: Close

Server response:
HTTP/1.1 404 Not Found
Content-Length : 0
Content-Type: text/plain

*****
Client('127.0.0.1', 51462) disconnected: no keep-alive
*****
```

4 Conclusion

The multi-thread HTTP web server is implemented successfully according to the given project requirement. Please refer to the `README.txt` file for the instructions of running the program, and `sample_server_log.txt` for the sample log file generated during the test of the server above. The HTML and image files are stored in the `htdocs` directory, and the server is able to handle the requests for the files. Finally, sincere thanks for your patience and consideration!