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

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 00:00:00.00000`.

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.

```python
# Defined in the run() function,
# that is, the main function of the server
while True:
    connection, address = self.server_socket.accept()
    thread = threading.Thread(target = self.receive,
        args = (connection, address))
    thread.start()
    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.

```python
# Defined in the run() function,
except KeyboardInterrupt:
    """
    Fetch the KeyboardInterrupt and print a message......
    """
finally:
    """
    Close the server socket and log file
    Close the connections in the list of threaded connections
    """
    self.server_log.close()
    self.server_socket.close()
    for connection in self.threaded_connections:
        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.

```
# Defined in the receive function,
# that is, for each thread handling the request
except socket.timeout:
    """
    If the connection is keep-alive and the timeout is reached,
    the server closes the connection
    Print messages...
    """
    connection.close()
    break
except KeyboardInterrupt:
    """
    If the server is interrupted, the server closes the connection,
    and removes it from the list of threaded connections
    """
    self.threaded_connections.remove(connection)
    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.

```
# Defined in the receive function
"""
The server checks if the connection is keep-alive or not
"""
keep_alive = False
for line in request.split("\n"):
    if ("Connection: keep-alive" in line)
        or ("Connection: Keep-Alive" in line):
        keep_alive = True
        break
```

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

```
"""
Parse the request into method and url
"""
request_method = request.split("\n")[0].split(" ")[0]
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.

```
#Defined in the receive function
"""
The server calls the appropriate response function,
based on the method of the request
"""
if (request_method == "GET"):
    self.response_GET(connection, request,
        request_url, address)
elif (request_method == "HEAD"):
    self.response_HEAD(connection, request,
        request_url, address)
else:
    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 examines 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.

```
# Defined in the HTTPServer class
def get_file_type(self, request_url):
"""
This function returns the type of the file requested
:request_url: url of the request
"""
    if request_url.endswith(".html"):
        return "text/html"
    elif request_url.endswith(".jpg"):
        return "image/jpg"
    elif request_url.endswith(".png"):
        return "image/png"
    else:
        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.

```python
# Defined in the response_GET function
if ("If-Modified-Since" in request):

    for line in request.split("\n"):
        if "If-Modified-Since" in line:
            check_time = line[19:48]
            break;

    check_time = datetime.strptime(check_time,
         '%a, %d %b %Y %H:%M:%S %Z')
    file_time =
        datetime.fromtimestamp(os.path.getmtime("htdocs"
            + request_url))

    if (file_time <= check_time):

        response_parts = ["HTTP/1.1 304 Not Modified\r\n",
                        "Last-Modified: " +
                        str(datetime.fromtimestamp(
                            os.path.getmtime("htdocs" +
                                request_url))) + "\r\n",
                        "Content-Length: " +
                            str(len(contents)) + "\r\n",
                        "Content-Type: " +
                            file_type + "\r\n\r\n"]
        response = "".join(response_parts)
        connection.sendall(response.encode())

        print(response)
        self.write_log(address, request_url, "304 Not Modified")

    else:

        response_parts = ["HTTP/1.1 200 OK\r\n",
                        "Last-Modified: " +
                    str(datetime.fromtimestamp(os.path.getmtime("htdocs"
                            + request_url))) + "\r\n",
                        "Content-Length: " +
                            str(len(contents)) + "\r\n",
                        "Content-Type: " +
                            file_type + "\r\n\r\n"]
        response = "".join(response_parts)
        connection.sendall(response.encode())

        print("Server response:\n")
        print(response)
        self.write_log(address, request_url, "200 OK")
else:
    # 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.

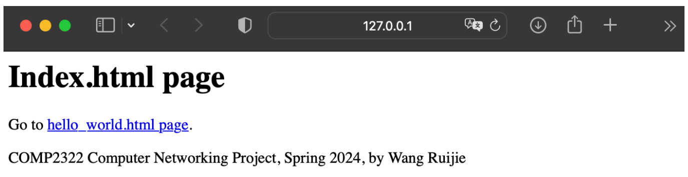
```python
# Defined in the HTTPServer class
def write_log(self, address, file_name, response_type):
"""
This function writes the request and response to the log file
:address: address of the client
:file_name: name of the requested file
:response_type: type of the response
:writing_parts: parts of the log message
:writing: log message
"""

    writing_parts = [str(datetime.now()),
        ": client", str(address),
        " requested file ", file_name,
        " with response ",
        response_type, "\n"]
    writing = "".join(writing_parts)
    self.server_log.write(writing)
    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.



Index.html page

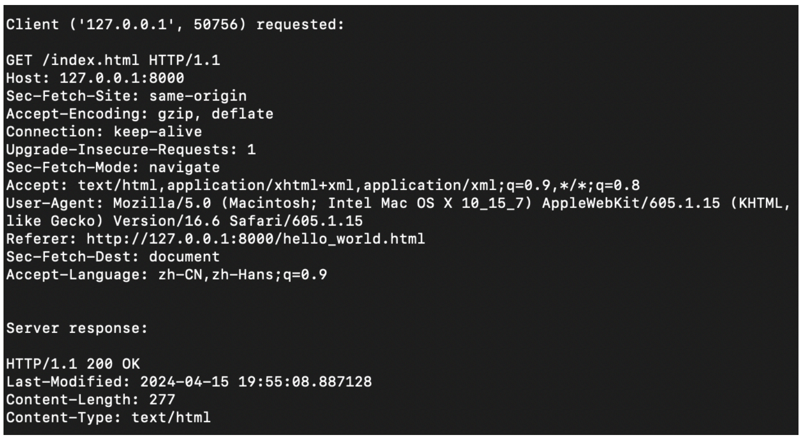
Go to hello_world.html page.

COMP2322 Computer Networking Project, Spring 2024, by Wang Ruijie

The client request and server response are as follows:

```
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
```

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

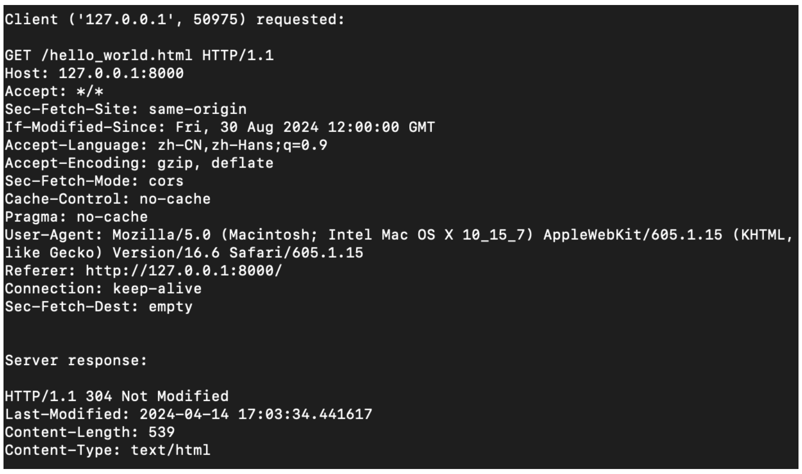
Again, but not cleaning the web cache, the server will return a `304 Not Modified` response, indicating that the file is not modified. On macOS, you may use the following script to test the feature:

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

```
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
```

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

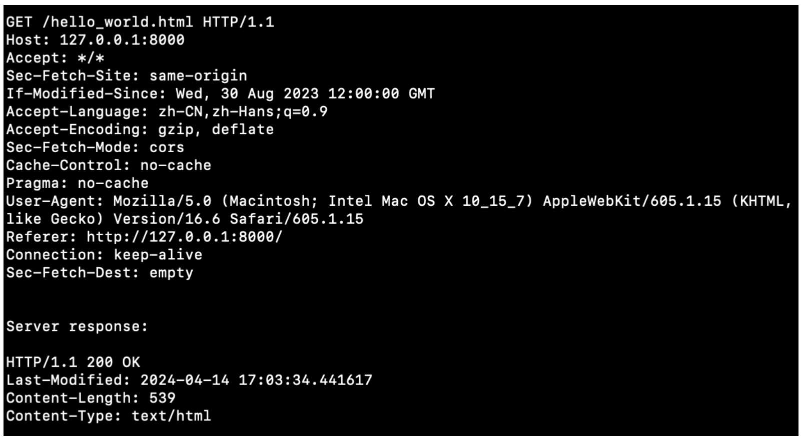
A past time stamp is used, i.e., Wed, 30 Aug 2023 12:00:00 GMT. The script used is as follows:

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

The client request and server response are as follows:

```
Client ('127.0.0.1', 51001) requested:

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
```

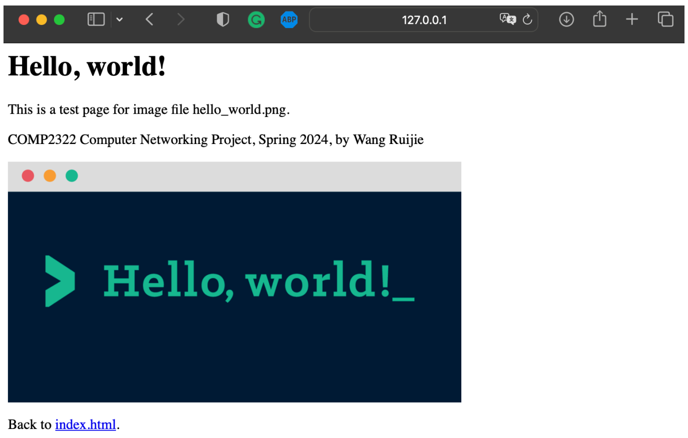
The server also returns a `200 OK` response. The screen capture of the console output is as follows:



```
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.


127.0.0.1

Hello, world!

This is a test page for image file hello_world.png.

COMP2322 Computer Networking Project, Spring 2024, by Wang Ruijie

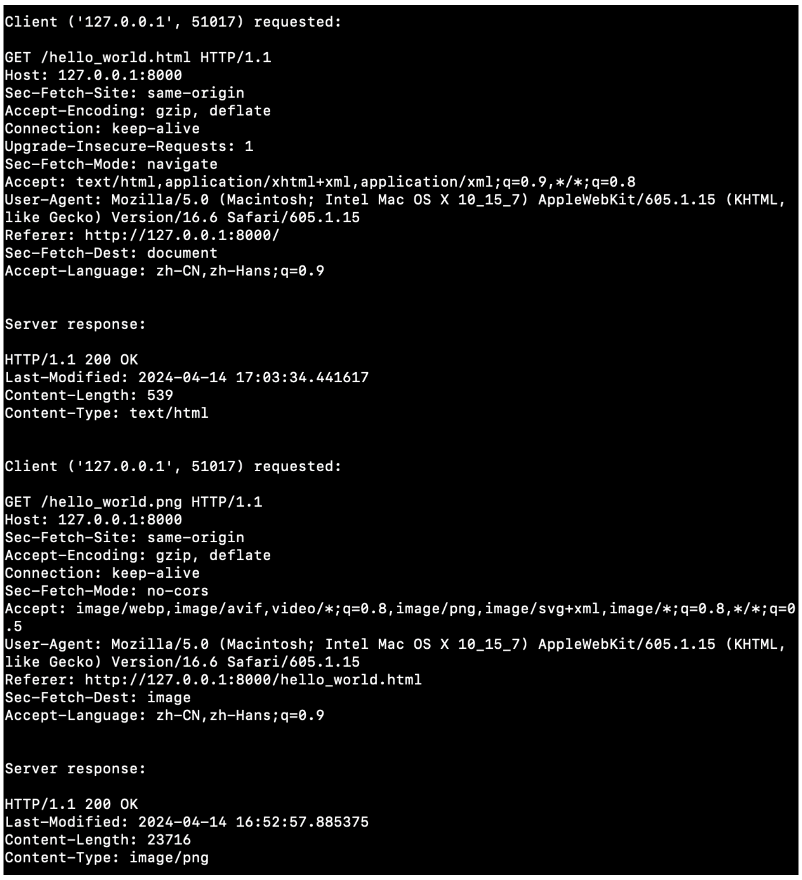
Back to index.html.

The client request and server response are 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,video/*;q=0.8,image/png,image/svg+xml,image/*;q=
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
```

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,video/*;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.

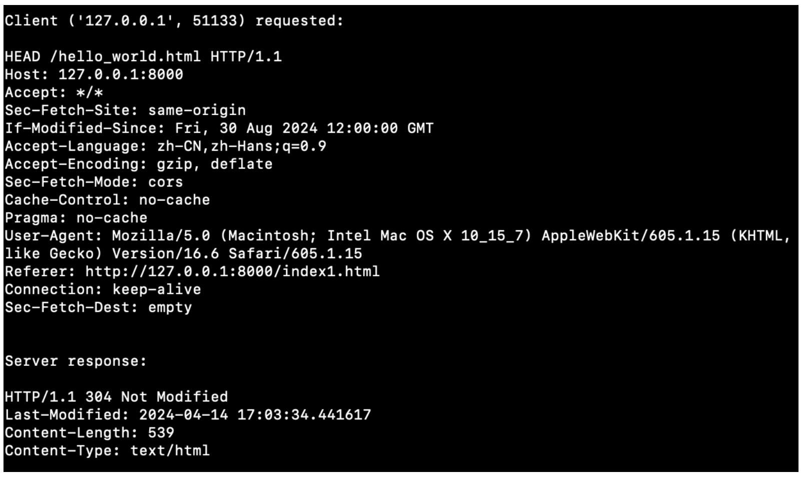
```
const headers = new Headers();
headers.append('If-Modified-Since', 'Fri, 30 Aug 2024 12:00:00 GMT');
fetch('http://127.0.0.1:8000/hello_world.html',
    { method: 'HEAD', headers: headers})
```

The client request and server response are 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.
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
```

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:

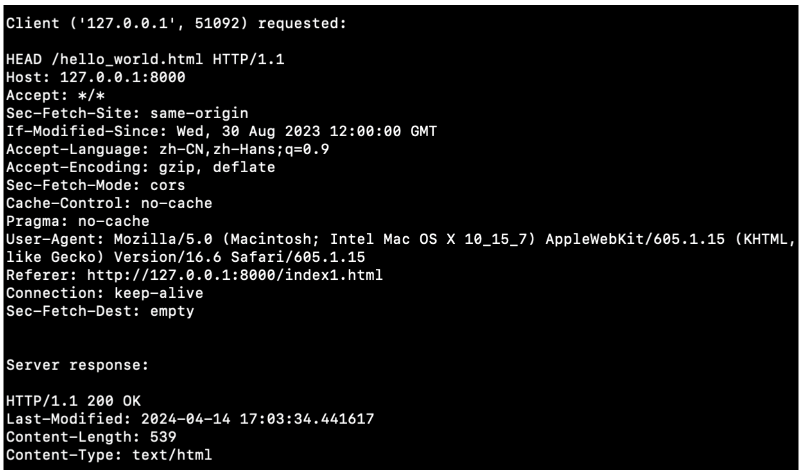
```
const headers = new Headers();
headers.append('If-Modified-Since', 'Wed, 30 Aug 2023 GMT');
fetch('http://127.0.0.1:8000/hello_world.html',
    { method: 'HEAD', headers: headers})
```

The client request and server response are as follows:

```
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
```

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`.

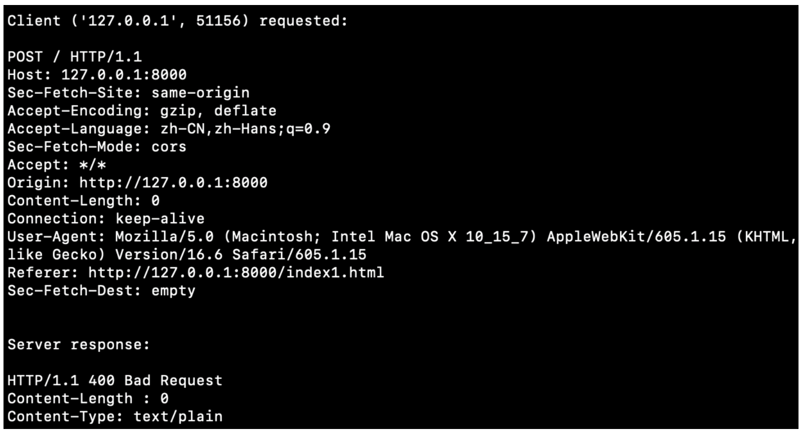
```
fetch('http://127.0.0.1:8000/', { method: 'POST' })
```

The client request and server response are 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.
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
```

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
```

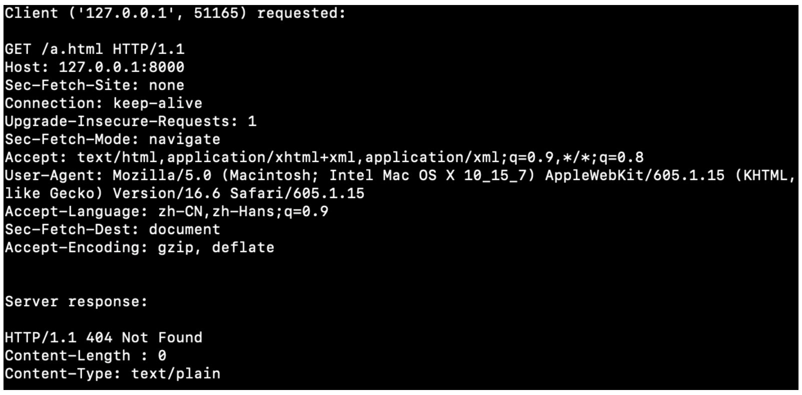
3.8 Handling of Request for a Non-Existent File

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

```
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
```

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
```

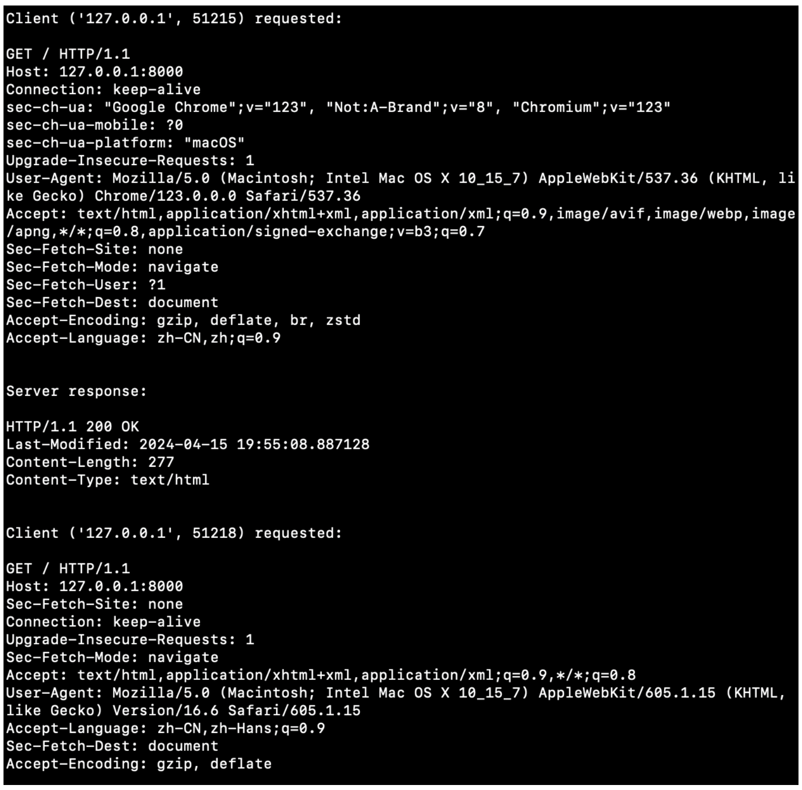
3.9 Multi-Thread Handling

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.

```
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, like Gecko) Chrome/123.0.0.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image
    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


Server response:

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

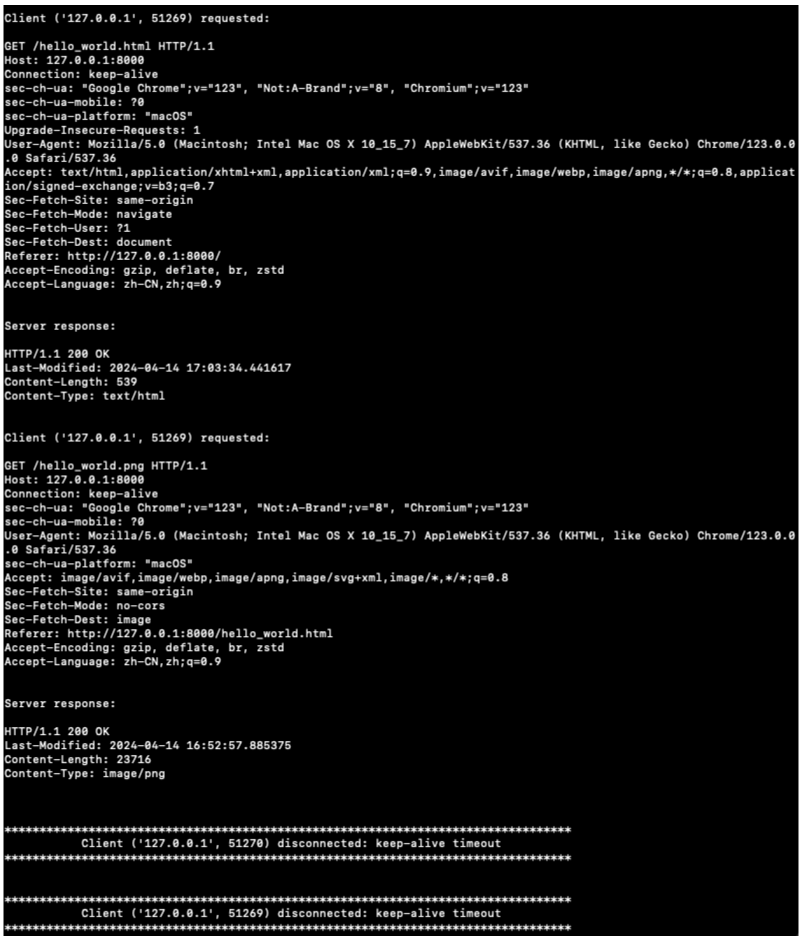
3.10 Handling of Keep-Alive Connection

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.

```
Client ('127.0.0.1', 51269) requested:

GET / HTTP/1.1
Host: 127.0.0.1:8000
Connection: keep-alive
Cache-Control: max-age=0
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/123.0.0.0 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=b
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', 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/123.0.0.0 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/123.0.0.0 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 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
.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,applicat
ion/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
.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
********************************************************************************
```

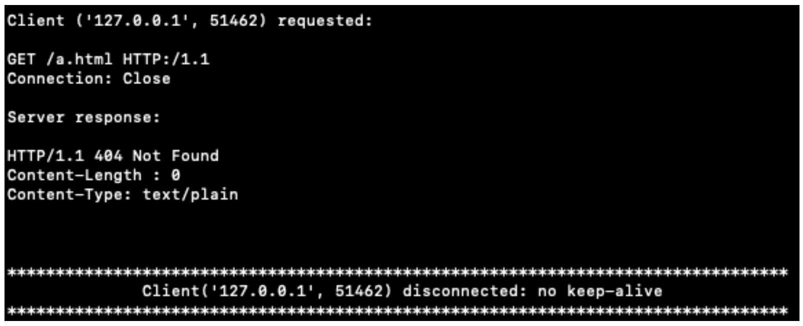
The browser sends multiple requests before timeout, which proves that the `Keep-Alive` feature is implemented successfully. However, if the request does not contain the `Connection: Keep-Alive` header field, the server will close the connection immediately after the request is handled. For example, if 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.

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

The output is as follows:

```

Client ('127.0.0.1', 51421) 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', 51421) disconnected: no keep-alive
****************************************************
```

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!