

Comp 2322 Computer Networking

# Multi-thread Web Server

-----Individual project report

Student name: LI Tong

Student ID: 21101988D

# [Index]

- 1 Summary of project design and implementation (1 - 5)
- 2 Demonstration (6 - 13)
- 3 Log file (13)

## 1 Summary of project design and implementation

This computer network project mainly implements a simple multi-threaded network server. It can establish a connection with the client and receive the client's HTTP request (GET and HEAD) and make different responses according to different requests and actual situations. In addition, the server can record all user requests in a log file.

### 1.1 Infrastructure and implementation logic

This project is implemented using python language, based on the content of LAB5 in the computer network course. Some of the content is referenced from LAB5.

#### Examples:

##### The use of socket package

```
# Define socket host and port
SERVER_HOST = 'localhost'
SERVER_PORT = 8000
# Create socket
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
server_socket.bind((SERVER_HOST, SERVER_PORT))
server_socket.listen(1)
```

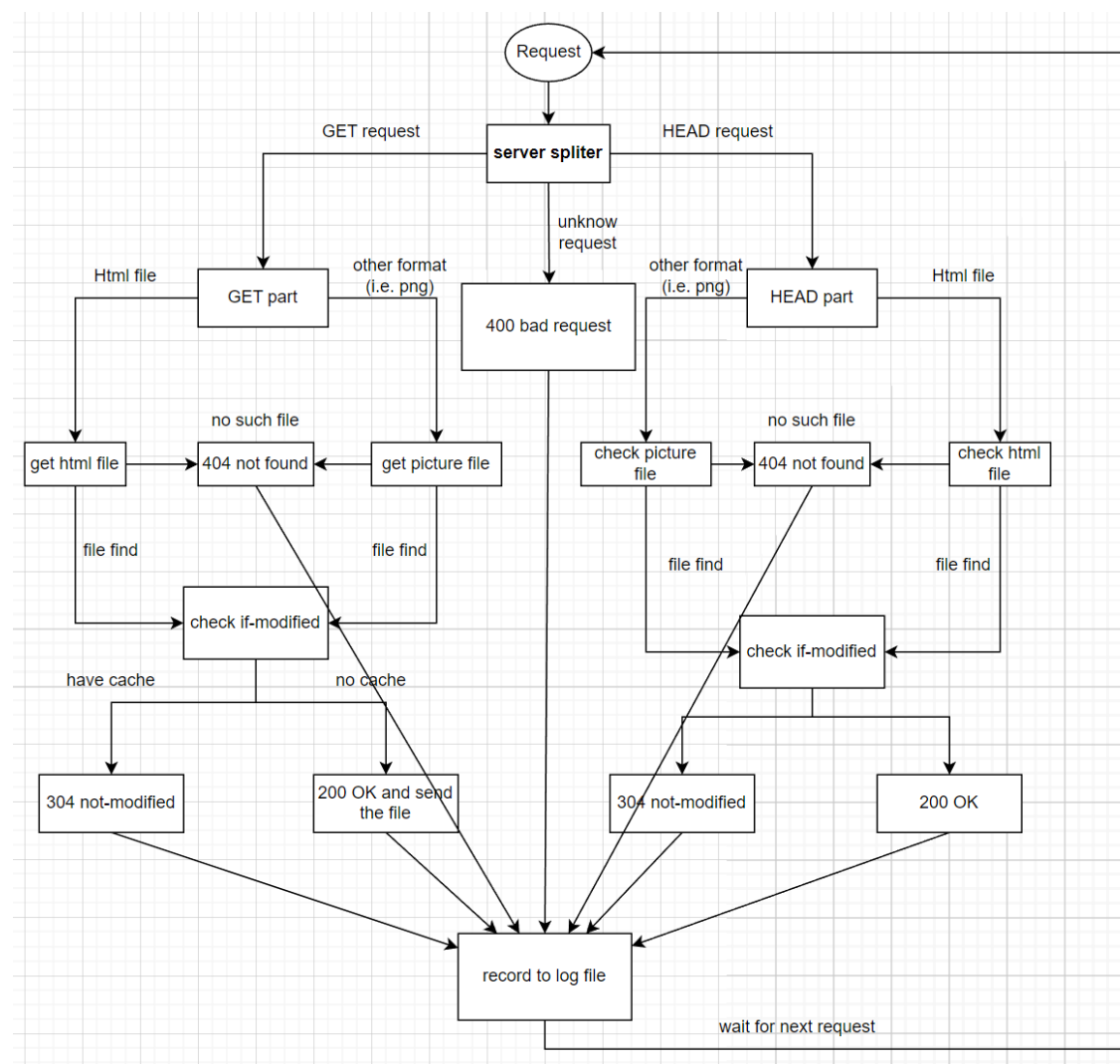
Use the socket package that comes with the python language to realize the connection between the server and the user.

##### Standard file storage format:

```
open('htdocs' + filename)
```

Refer to the file storage method of LAB5, and store all accessible files in the server in a folder (htdocs), which is convenient for management.

## Implementation logic:



**Logic flow chart**

The logic flow diagram above briefly describes how a request from a client is processed after it is received.

After the server receives a client request, it will first split the client request to obtain the required information (for example, the type of request, the requested file name, file type, the time of last acquisition, etc.), and respond based on this information. In this step, if the request type cannot be parsed, the server will directly return 400-bad-request.

For GET type requests, the server will further judge the requested file type (because the way of reading text files and image files is different) and check whether the file exists. If not, it will return 404-not-found. If the file exists, the if-modified-since time will be compared with the file last modified time and return 200-OK with required file or 304-not-modified accordingly.

For HEAD type requests, the server does not need to return the file returned by this command, but still needs to detect whether the file exists and if-modified-since time (similar to GET requests), and then return 304-not-modified or 200-OK and some necessary entries (e.g., date, file length, etc.).

When the server finishes processing a request, it will record the request in the log file

and return to the original state, waiting for the next request.

## 1.2 Details of design and Implementation

### 1.2.1 Implementation of multithreading

```
import threading
```

```
client_connection, client_address = server_socket.accept()
thread = threading.Thread(target=Single_thread, args=(client_connection, client_address,))
thread.start()
```

```
1 usage
def Single_thread(client_connection, client_address):
```

#### The use of threading package

After calling the threading package in the python language, encapsulate the modules that require multi-threading into functions (the function is named single\_thread in the figure, and then the multi-threading of the program can be realized.

### 1.2.2 Time dealer

```
import time import datetime
```

```
4 usages
def time_compare(last_time, file_name):
    f = time.strftime("%a, %d %b %Y %I:%M:%S GMT", time.gmtime(os.path.getmtime('htdocs' + file_name)))
    print(f)
    print(last_time)
    f1 = datetime.datetime.strptime(f, "%a, %d %b %Y %I:%M:%S GMT").timestamp()
    f2 = datetime.datetime.strptime(last_time, "%a, %d %b %Y %I:%M:%S GMT").timestamp()
    return f1 > f2
```

#### Function to compare last-time and if-modified-since-time

With the use of time and datetime packages, this function could receive a file name and a time with standard time format which the browser is use. Then, get the last\_modified time from the requested file name and then change both time to another format by use the function *timestamp()* to make comparable. This function will return a bool value.

```
Wed, 5 Apr 2023 01:22:03 GMT
```

Example standard time format which browser use

### 1.2.3 Output dealer

```

12 usages
def output(res_type, file_name, file_type):
    file_len = os.path.getsize('htdocs' + file_name)
    date = time.strftime("%a, %d %b %Y %I:%M:%S GMT", time.localtime())
    date = date + '\n'
    last_m = time.strftime("%a, %d %b %Y %I:%M:%S GMT", time.gmtime(os.path.getmtime('htdocs' + file_name)))
    if file_type == 'html':
        content_type = 'Content_Type: text/html\n\n'
    else:
        content_type = 'Content_Type: application/x-png\n\n'
    return res_type + 'Date: ' + date + 'Last_Modified: ' + last_m + '\n' + 'Content_Length: ' \
        + str(file_len) + '\n' + content_type

```

### function to produce output

In the standard service request response, there are not only the required files and the return status, but also other necessary information (such as date, last modification time, file length, file type, etc.). After receiving the necessary information, the function, will process and return a formatted output that makes the output clean and nice.

```

Server response:

HTTP/1.1 200 OK
Date: Sat, 15 Apr 2023 02:01:10 GMT
Last_Modified: Tue, 11 Apr 2023 02:04:33 GMT
Content_Length: 195
Content_Type: text/html

<html>
<head>
    <link rel="icon" href="data:,">
    <title>Hello World Title</title>
</head>
<body>
    <h1>HELLO WORLD!</h1>
    <h2>HELLO WORLD!!</h2>
    <h3>HELLO WORLD!!!</h3>
</body>
</html>

```

### Sample output

#### 1.2.4 The use of try-and-except

```

try:
    headers = request.split("\n")
    fields = headers[0].split()
    request_type = fields[0]
    filename = fields[1]
except IndexError:
    response = 'HTTP/1.1 400 Bad Request\n\nRequest Not Supported'.encode()
    response_type = 'HTTP/1.1 400 Bad Request\n'

```

## IndexError

```
except FileNotFoundError:
    response = not_found().encode()
    response_type = 'HTTP/1.1 404 Not Found\n'
```

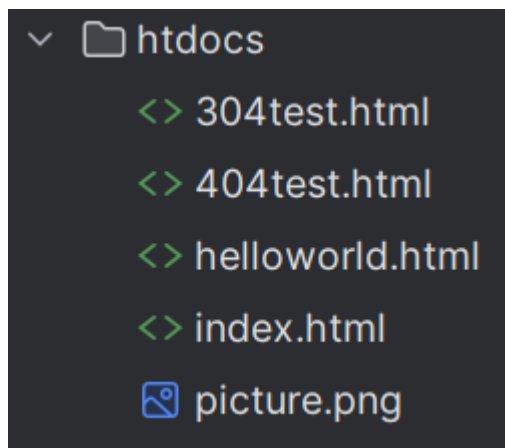
## FileNotFoundError

### Use “try” to catch some errors and send right response

In the face of unknown types of requests and malformed requests, errors will be generated in the segmentation of requests. Using this, it is just possible to use the try structure to capture some errors and make correct responses.

When the file requested by the user does not exist, the server will also throw an error. We can also use the try structure to catch this error and make a reasonable response.

## 2 Demonstration



On the left are all files owned by the server (the 304test.html and 404test.html files have no practical significance, just to make the browser interface beautiful)

When displaying, not only the browser will be used for display, but also the code client will be used for display, to facilitate the display of the server's response.

```
SERVER_HOST = '127.0.0.1'
SERVER_PORT = 8000

client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket.connect((SERVER_HOST, SERVER_PORT))
request = input('Input HTTP request command:\n')
client_socket.send(request.encode())
response = client_socket.recv(1024)
print('Server response:\n')
print(response.decode())
client_socket.close()
```

Code client referenced from LAB5

## 2.1 GET type request

### 2.1.1 Normal access

Request: <http://localhost:8000/> or <http://localhost:8000/index.html> (the same because the default page of the server is index.html)

Screen capturing:



# Welcome to the index.html web page.

Here's a link to [hello world](#).

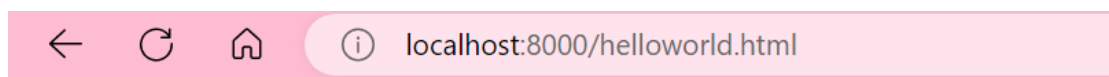
Here's a link to [picture of 3](#).

browser page

```
Listening on port 8000 ...
request:
GET / HTTP/1.1
Host: localhost:8000
Connection: keep-alive
sec-ch-ua: "Chromium";v="112", "Microsoft Edge";v="112", "Not:A-Brand";v="99"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/112.0.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/
Purpose: prefetch
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate, br
Accept-Language: zh-CN,zh;q=0.9,en;q=0.8,en-GB;q=0.7,en-US;q=0.6
```

server page

After click magnet link "hello world":



# HELLO WORLD!

## HELLO WORLD!!

## HELLO WORLD!!!

browser page

After click magnet link "picture of 3":



browser page

### 2.1.2 Abnormal access

Request: <http://localhost:8000/nofile.html> (there is no such file in sever)

Screen capturing:



## 404 Not Found

The requested resource could not be found on this server.

browser page

```
request:

GET /nofile.html HTTP/1.1
Host: localhost:8000
Connection: keep-alive
sec-ch-ua: "Chromium";v="112", "Microsoft Edge";v="112", "Not:A-Brand";v="99"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate, br
Accept-Language: zh-CN,zh;q=0.9,en;q=0.8,en-GB;q=0.7,en-US;q=0.6
```

server page

### 2.1.3 Normal access on code client

Request: GET /index.html HTTP/1.1\nHost: localhost:8000\nConnection: keep-alive\nif-modified-since: Wed, 5 Apr 2023 01:22:03 GMT



(the last-modified time of the requested file is Tue, 11 Apr 2023 04:22:03 GMT, which is later, so the response type should be 200 OK)

Screen capturing:

```
Listening on port 8000 ...
request:

GET /index.html HTTP/1.1\nHost: localhost:8000\nConnection: keep-alive\nif-mod
Tue, 11 Apr 2023 04:22:03 GMT
Wed, 5 Apr 2023 01:22:03 GMT
|
```

server page

```
Server response:

HTTP/1.1 200 OK
Date: Sat, 15 Apr 2023 02:52:51 GMT
Last_Modified: Tue, 11 Apr 2023 04:22:03 GMT
Content_length: 290
Content_Type: text/html

<html>
<head>
  <link rel="icon" href="data:,">
  <title>Index</title>
</head>
<body>
  <h1>Welcome to the index.html web page.</h1>
  <p>Here's a link to <a href="helloworld.html">hello world</a>.</p>
  <p>Here's a link to <a href="picture.png">picture of 3</a>.</p>
</body>
</html>
```

Client page

Request: **GET /index.html HTTP/1.1\nHost: localhost:8000\nConnection: keep-alive\nif-modified-since: Wed, 12 Apr 2023 01:22:03 GMT**

(the last-modified time of the requested file is Tue, 11 Apr 2023 04:22:03 GMT, which is earlier, so the response type should be 304 Not Modified)

Screen capturing:

```
request:

GET /index.html HTTP/1.1\nHost: localhost:8000\nConnection: keep-alive\nif-modifie
Tue, 11 Apr 2023 04:22:03 GMT
Wed, 12 Apr 2023 01:22:03 GMT
```

#### server page

Server response:

HTTP/1.1 304 Not Modified

Date: Sat, 15 Apr 2023 02:58:48 GMT

Last\_Modified: Tue, 11 Apr 2023 04:22:03 GMT

Content\_length: 290

Content\_Type: text/html

<html>

<head>

<link rel="icon" href="data:,">

<title>304 Not Modified</title>

</head>

<body>

<h1>304 Not Modified</h1>

<h1>The requested resource is not modified.</h1 >

</body>

</html>

Process finished with exit code 0

#### Client page

##### 2.1.4 Abnormal access on code client

Request: GET /no.html HTTP/1.1\nHost: localhost:8000\nConnection: keep-alive\nif-modified-since: Wed, 12 Apr 2023 01:22:03 GMT

(the requested file no.html is not find on server, so the response type should be 404 Not Find)

Screen capturing:

request:

GET /no.html HTTP/1.1\nHost: localhost:8000\nConnection: keep-alive

#### server page

```
Server response:
```

```
HTTP/1.1 404 Not Found
```

```
|
```

```
<html>
```

```
<head>
```

```
    <link rel="icon" href="data:,">
```

```
    <title>404 Not Found</title>
```

```
</head>
```

```
<body>
```

```
    <h1>404 Not Found</h1>
```

```
    <h1>The requested resource could not be found on this server.</h1 >
```

```
</body>
```

```
</html>
```

```
Process finished with exit code 0
```

Client page

Request: **CATCH /no.html HTTP/1.1**\nHost: localhost:8000\nConnection: keep-alive\nIf-Modified-Since: Wed, 12 Apr 2023 01:22:03 GMT

(The request type is unknown, the response type should be 400 Bad Request)

Screen capturing:

```
request:
```

```
CATCH /no.html HTTP/1.1\nHost: localhost:8000\nConnection: keep-alive
```

server page

```
Server response:
```

```
HTTP/1.1 400 Bad Request
```

```
Request Not Supported
```

```
Process finished with exit code 0
```

Client page

## 2.2 HEAD type request:

Since the HEAD type request is very similar to the GET request and it is difficult to display on

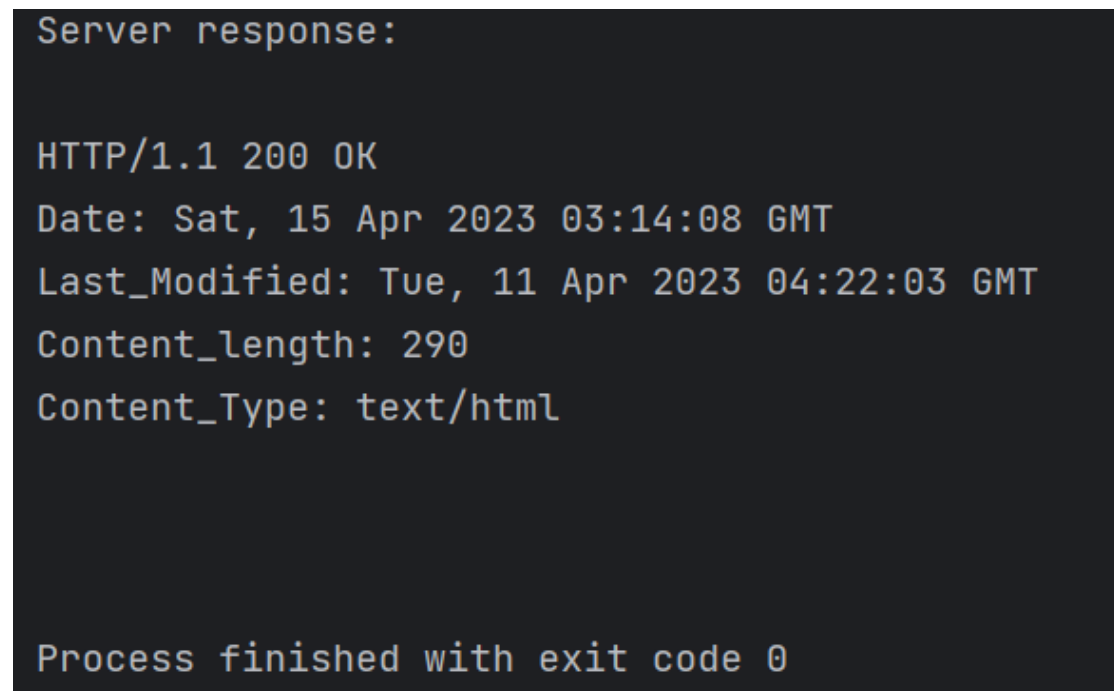
the browser, the following will use the code client to display and only display the screenshot of the client (the server-side screenshot is consistent with the display of the GET type).

### 2.2.1 Normal access on client

Request: **HEAD /index.html HTTP/1.1**\nHost: localhost:8000\nConnection: keep-alive\nif-modified-since: Wed, 5 Apr 2023 01:22:03 GMT

(the last-modified time of the requested file is Tue, 11 Apr 2023 04:22:03 GMT, which is later, so the response type should be 200 OK)

Screen capturing:



```
Server response:

HTTP/1.1 200 OK
Date: Sat, 15 Apr 2023 03:14:08 GMT
Last_Modified: Tue, 11 Apr 2023 04:22:03 GMT
Content_length: 290
Content_Type: text/html

Process finished with exit code 0
```

Client page

Request: **HEAD /picture.png HTTP/1.1**\nHost: localhost:8000\nConnection: keep-alive\nif-modified-since: Wed, 12 Apr 2023 01:22:03 GMT

(the last-modified time of the requested file is Thu, 17 Nov 2022 04:42:07 GMT, which is earlier, so the response type should be 304 Not Modified)

Screen capturing:

Server response:

HTTP/1.1 304 Not Modified

Date: Sat, 15 Apr 2023 03:15:29 GMT

Last\_Modified: Thu, 17 Nov 2022 04:42:07 GMT

Content\_length: 57649

Content\_Type: application/x-png

<html>

<head>

<link rel="icon" href="data:,">

<title>304 Not Modified</title>

</head>

<body>

<h1>304 Not Modified</h1>

<h1>The requested resource is not modified.</h1 >

</body>

</html>

Process finished with exit code 0

Client page

### 2.2.2 Abnormal access on client

Request: **HEAD /no.html HTTP/1.1**\nHost: localhost:8000\nConnection: keep-alive\nif-modified-since: Wed, 5 Apr 2023 01:22:03 GMT

(the requested file no.html is not find on server, so the response type should be 404 Not Find)

Screen capturing:

```

Server response:

HTTP/1.1 404 Not Found

<html>
<head>
  <link rel="icon" href="data:,">
  <title>404 Not Found</title>
</head>
<body>
  <h1>404 Not Found</h1>
  <h1>The requested resource could not be found on this server.</h1 >
</body>
</html>

Process finished with exit code 0

```

Client page

ALL the cases displayed.

### 3 Log file

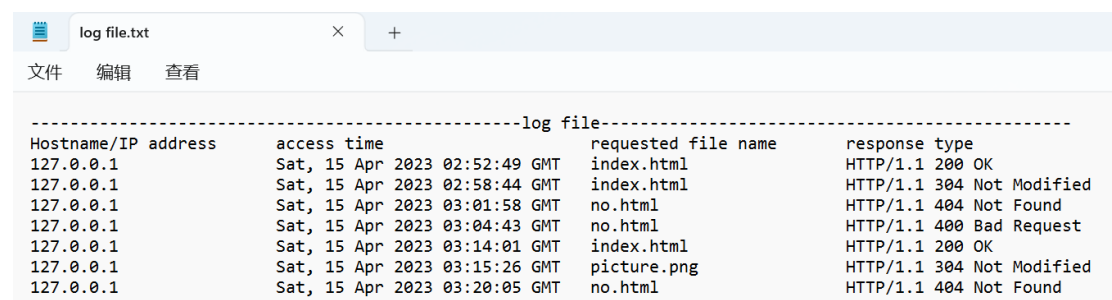
After the server successfully receives and processes a request, it will record the relevant information in a log file. The log files are in the same folder "Report" as this report. The content of the log file is briefly shown below.

```

1 -----log file-----
2 Hostname/IP address      access time      requested file name  response type
3 127.0.0.1                Sat, 15 Apr 2023 02:52:49 GMT  index.html          HTTP/1.1 200 OK
4 127.0.0.1                Sat, 15 Apr 2023 02:58:44 GMT  index.html          HTTP/1.1 304 Not Modified
5 127.0.0.1                Sat, 15 Apr 2023 03:01:58 GMT  no.html             HTTP/1.1 404 Not Found
6 127.0.0.1                Sat, 15 Apr 2023 03:04:43 GMT  no.html             HTTP/1.1 400 Bad Request
7 127.0.0.1                Sat, 15 Apr 2023 03:14:01 GMT  index.html          HTTP/1.1 200 OK
8 127.0.0.1                Sat, 15 Apr 2023 03:15:26 GMT  picture.png         HTTP/1.1 304 Not Modified
9 127.0.0.1                Sat, 15 Apr 2023 03:20:05 GMT  no.html             HTTP/1.1 404 Not Found

```

Open with the python compiler



```

-----log file-----
Hostname/IP address      access time      requested file name  response type
127.0.0.1                Sat, 15 Apr 2023 02:52:49 GMT  index.html          HTTP/1.1 200 OK
127.0.0.1                Sat, 15 Apr 2023 02:58:44 GMT  index.html          HTTP/1.1 304 Not Modified
127.0.0.1                Sat, 15 Apr 2023 03:01:58 GMT  no.html             HTTP/1.1 404 Not Found
127.0.0.1                Sat, 15 Apr 2023 03:04:43 GMT  no.html             HTTP/1.1 400 Bad Request
127.0.0.1                Sat, 15 Apr 2023 03:14:01 GMT  index.html          HTTP/1.1 200 OK
127.0.0.1                Sat, 15 Apr 2023 03:15:26 GMT  picture.png         HTTP/1.1 304 Not Modified
127.0.0.1                Sat, 15 Apr 2023 03:20:05 GMT  no.html             HTTP/1.1 404 Not Found

```

open directly

-----END-----