Comp 2322 Computer Networking

Project: Multi-thread Web Server

Student name: Lai Ka Chung

Stduent ID: 22080062d

1. Introduction

This Multi-thread Web Server can receive the HTTP request message and display the content with multiple requests. Also, It has a log file to record the statistics of the client request. Each request corresponds to one line of record in the log. Write down client hostname/IP address, access time, requested file name and response type for each record. Moreover, it can handle some simple errors like web pages not found.

2. Function

2.1 Ports

The server's address will be 127.0.0.1, and the port will be 8000, which will be shown when the user launches the web server. Also, when a connection is received, the IP address and the client's port will be shown, as in the picture below.

For example:

```
socket successfully created
socket binded to 8000
Server running on 127.0.0.1:8000
Got connection from: 127.0.0.1:2836
```

[Website run on Port 8000]

2.2 Multi-threaded

Using multi-threaded at the server can allow parallel processing with the goal of increasing processing speed. Also, This can deal with asynchronous action; the server can wait for a new client connection and send the file to the client parallelly.

For example:

```python
while True:
    # Wait for client connections
    client_connection, client_address = server_socket.accept()

    print("Got connection from: " + client_address[0] + ":" + str(client_address[1]))
    # Handle client connection in a new thread
    client_handler = threading.Thread(
        target=handle_request,
        args=(client_connection,client_address)
    )
    client_handler.start()
```

[source code in run_server function]

2.3 HTTP Request Handling

The server is designed to handle HTTP requests by using the handle_request function. This function will be called each time when a new client connection is established. Also, if the reques_type is not expected, there will be error handling for it

For example:

```python
# Handle the HTTP request
def handle_request(client_socket,client_address):
    # Default to persistent connection
    connection_type = 'keep-alive'
    try:
        request = client_socket.recv(1024).decode()  # Receive the request from the client
        # Parse HTTP headers
        headers = request.split('\n')
        fields = headers[0].split()
        if len(fields) >= 2:
            request_type = fields[0]
            filename = fields[1]
        else:
            client_socket.close()
            print(f'Timeout, the client socket {client_address[0]}:{client_address[1]} has been closed')
            return

        if not request_type in ['GET','HEAD']:
            response_header = create_response_header(400, 'text/html', 'N/A',connection_type)
            client_socket.sendall(response_header.encode())
            print(response_header)
            # Log the response header
            log_header(response_header)
            return
```

[source code of handle_request]

```python
except Exception as e:
    print(f'Error handling request: {e}')
```

[source code of handle_request]

2.4 Response Status

There are four response statutes 200, 304, 400, and 404.

200 OK: This status code indicates that the request has succeeded and processes a GET or HEAD request.

304 Not Modified: This status code indicates that the requested resource has not been modified since the last request.

400 Bad Request: This status code indicates that when the request is made, the server cannot understand it due to invalid syntax.

404 File Not Found: This status code indicates that the server can not find the requested resource.

For example:

200 OK:

```
if status_code == 200:
    header += 'OK\r\n'
```

[source code of create_response_header]

304 Not Modified:

```
# Detect any if_modified_since
if if_modified_since:
    if_modified_since = time.mktime(time.strptime(if_modified_since, "%a, %d %b %Y %H:%M:%S"))
    if if_modified_since >= os.path.getmtime(file_path):
        # Send 304 Not Modified response
        response_header = create_response_header(304, 'text/html', last_modified,connection_type)
        client_socket.sendall(response_header.encode())
        print(response_header)
        # Log the response header
        log_header(response_header)
        return
```

[source code of handle_request]

400 Bad Request:

```
# Indicate request type is GET or HEAD
if not request_type in ['GET','HEAD']:
    response_header = create_response_header(400, 'text/html', 'N/A',connection_type)
    client_socket.sendall(response_header.encode())
    print(response_header)
    # Log the response header
    log_header(response_header)
    return
```

[source code of handle_request]

404 File Not Found:

```
# Check if the requested file exists
if not os.path.isfile(file_path):
    # Send 404 Not Found response
    response_header = create_response_header(404, 'text/html', 'N/A',connection_type)
    client_socket.sendall(response_header.encode())
```

[source code of handle_request]

2.5 File Serving

When responding to the GET request, the server will construct the full file path from the URL. Read it can send the content as part of the HTTP response.

For example:

```
# Send the file content
if filename.endswith('.html'):
    with open(file_path, 'r') as file:
        response_data = file.read()
        client_socket.sendall(response_data.encode()

else:
    with open(file_path, 'rb') as file:
        response_data = file.read()
        client_socket.sendall(response_data)
```

[source code of handle_request]

2.6 HTTP Headers

My server handles several HTTP headers:

Date: Including current date and time in the response header

Connection: Including the connection type in the response header

Keep-Alive: Specify the maximum number of requests that can be sent over the same connection and the timeout period

Last-Modified: Including the last modified time of the requested file in the response header.

Content-Type: Including the type of file

For example:

```python
# Function to create the HTTP response header
def create_response_header(status_code, content_type, last_modified, connection_type):
    header = f'HTTP/1.1 {status_code} '
    if status_code == 200:
        header += 'OK\r\n'
    elif status_code == 304:
        header += 'Not Modified\r\n'
    elif status_code == 400:
        header += 'Bad Request\r\n'
    elif status_code == 404:
        header += 'File Not Found\r\n'
        content_type = 'N/A'

    header += f'Date: {time.strftime("%a, %d %b %Y %H:%M:%S", time.localtime())}\r\n'
    header += f'Connection: {connection_type}\r\n'
    header += 'Keep-Alive: timeout=10, max=100\r\n'
    header += f'Last-Modified: {last_modified}\r\n'
    header += f'Content-Type: {content_type}\r\n\r\n'
    return header
```

[source code of create_response_header]

2.7 Logging

The server logs the response headers to the file which is the path specified by LOG_FILE_PATH. The function log_header is used to write the response header to the file and close the file.

For example:

```python
LOG_FILE_PATH = os.path.join(os.getcwd(), "log.txt")
```

[source code of define part]

```python
# Helper function to log the response header to a log file
def log_header(header):
    with open(LOG_FILE_PATH, "a") as log_file:
        # Split the header into fields, add brackets around each field, and join them back together
        log_file.write(''.join([f'[{field}]' for field in header.split('\r\n') if field]) + '\n')
```

[source code of log_header]

3. Structure

There will be three main parts to the program.

First, Server Initialization. Initialise the server by creating the socket and waiting for a response. After the response, handle the wait and requests from multiple clients simultaneously.

Second, Duel is used to meet the client's request using the handle_request function.

Third, log every request's header in the log file.

4. Demonstration

4.1 Run the command "python server.py"

```
PS C:\Project> & C:/Users/s1031/AppData/Local/Programs/Python/Python312/python.exe c:/Project/server.py
socket successfully created
socket binded to 8000
Server running on 127.0.0.1:8000
```

4.2 Visit the website "127.0.0.1:8000" on the web browser

Welcome to the index.html web page.

Here's a link to test.

Here's a link to image.

4.3 Here is the output after several client connections

```
Got connection from: 127.0.0.1:3521
Got connection from: 127.0.0.1:3522
HTTP/1.1 404 File Not Found
Date: Sun, 21 Apr 2024 04:02:00
Connection: keep-alive
Keep-Alive: timeout=10, max=100
Last-Modified: N/A
Content-Type: N/A


Got connection from: 127.0.0.1:3524
HTTP/1.1 404 File Not Found
Date: Sun, 21 Apr 2024 04:02:01
Connection: keep-alive
Keep-Alive: timeout=10, max=100
Last-Modified: N/A
Content-Type: N/A


Got connection from: 127.0.0.1:3526
HTTP/1.1 200 OK
Date: Sun, 21 Apr 2024 04:02:10
Connection: keep-alive
Keep-Alive: timeout=10, max=100
Last-Modified: Sun, 21 Apr 2024 02:51:37
Content-Type: text/html


Got connection from: 127.0.0.1:3532
HTTP/1.1 200 OK
Date: Sun, 21 Apr 2024 04:02:14
Connection: keep-alive
Keep-Alive: timeout=10, max=100
Last-Modified: Sat, 20 Apr 2024 03:16:50
Content-Type: text/html


Got connection from: 127.0.0.1:3538
HTTP/1.1 200 OK
Date: Sun, 21 Apr 2024 04:02:17
Connection: keep-alive
Keep-Alive: timeout=10, max=100
Last-Modified: Sun, 21 Apr 2024 00:57:47
Content-Type: image/jpeg
```

```
Got connection from: 127.0.0.1:4951
Got connection from: 127.0.0.1:4952
HTTP/1.1 200 OK
Date: Sun, 21 Apr 2024 04:39:39
Connection: keep-alive
Keep-Alive: timeout=10, max=100
Last-Modified: Sun, 21 Apr 2024 02:51:37
Content-Type: text/html


Got connection from: 127.0.0.1:4997
HTTP/1.1 200 OK
Date: Sun, 21 Apr 2024 04:39:41
Connection: keep-alive
Keep-Alive: timeout=10, max=100
Last-Modified: Sun, 21 Apr 2024 00:57:47
Content-Type: image/jpeg


Got connection from: 127.0.0.1:4999
HTTP/1.1 200 OK
Date: Sun, 21 Apr 2024 04:39:42
Connection: keep-alive
Keep-Alive: timeout=10, max=100
Last-Modified: Sun, 21 Apr 2024 00:57:47
Content-Type: image/jpeg


Timeout, the client socket 127.0.0.1:4999 has been closed
```

4.4 Timeout if the connection is idle for too long

```
Timeout, the client socket 127.0.0.1:3538 has been closed
```

4.5 The log file on the server

log.txt

```
[HTTP/1.1 404 File Not Found][Date: Sun, 21 Apr 2024 04:02:00][Connection: keep-alive][Keep-Alive: timeout=10, max=100][Last-Modified: N/A][Content-Type: N/A]
[HTTP/1.1 404 File Not Found][Date: Sun, 21 Apr 2024 04:02:01][Connection: keep-alive][Keep-Alive: timeout=10, max=100][Last-Modified: N/A][Content-Type: N/A]
[HTTP/1.1 200 OK][Date: Sun, 21 Apr 2024 04:02:10][Connection: keep-alive][Keep-Alive: timeout=10, max=100][Last-Modified: Sun, 21 Apr 2024 02:51:37][Content-Type: text/html]
[HTTP/1.1 200 OK][Date: Sun, 21 Apr 2024 04:02:14][Connection: keep-alive][Keep-Alive: timeout=10, max=100][Last-Modified: Sat, 20 Apr 2024 03:16:50][Content-Type: text/html]
[HTTP/1.1 200 OK][Date: Sun, 21 Apr 2024 04:02:17][Connection: keep-alive][Keep-Alive: timeout=10, max=100][Last-Modified: Sun, 21 Apr 2024 00:57:47][Content-Type: image/jpeg]
[HTTP/1.1 200 OK][Date: Sun, 21 Apr 2024 04:39:39][Connection: keep-alive][Keep-Alive: timeout=10, max=100][Last-Modified: Sun, 21 Apr 2024 02:51:37][Content-Type: text/html]
[HTTP/1.1 200 OK][Date: Sun, 21 Apr 2024 04:39:41][Connection: keep-alive][Keep-Alive: timeout=10, max=100][Last-Modified: Sun, 21 Apr 2024 00:57:47][Content-Type: image/jpeg]
[HTTP/1.1 200 OK][Date: Sun, 21 Apr 2024 04:39:42][Connection: keep-alive][Keep-Alive: timeout=10, max=100][Last-Modified: Sun, 21 Apr 2024 00:57:47][Content-Type: image/jpeg]

```

Web Server Readme:

readme.txt

```
# Project Title

    COMP2322 Project - Multi-thread Web Server

## Description

    This is a python program to implement a Web Service using the HTTP protocol

## Main function

    1. Multi-thread Web Server
    2. Proper request and response message exchanges
    3. GET command for both text files and image files
    4. HEAD command
    5. Four types of response statuses (200, 400, 404, 304)
    6. Handle Last-Modified and If-Modified-Since header fields
    7. Handle Connection header field for both HTTP persistent connection (keep-alive) and non-persistent connection (close)
    8. Logging

## Requirement

    Python 3
    build-in library socket, sys, threading, time, os

## Usage

    To start the server, run the following command:

        python server.py

    You can visit it by http://127.0.0.1:8000/

Author

    Lai Ka Chung (22080062d)
```