CN ASSIGNMENT 2

TCP-based Web Application Lab

(Group No 3)

Piyush Narula (2022354)

Aditya Aggrawal (2022028)

Part 1: Single-Threaded TCP Web Server

Functionality

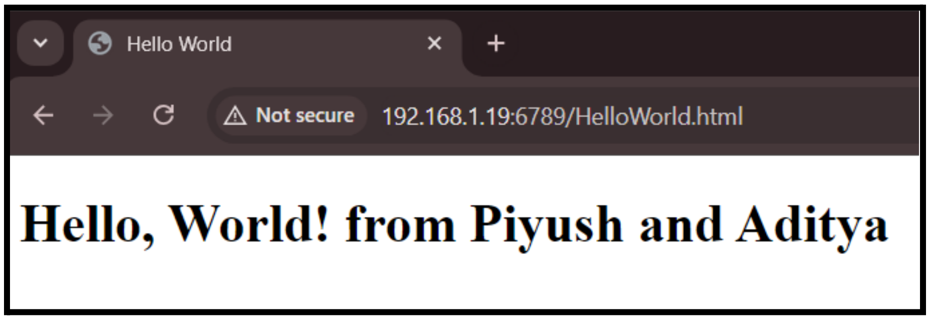
This task showcases creating a basic web server using Python's socket module. The server listens to a designated port, processes HTTP GET requests from clients retrieves the requested HTML file, and returns it to the client with a “200 OK” status. If the file is unavailable, the server replies with a "404 Not Found" error.

Implementation

The server creates a socket using *socket(AF_INET, SOCK_STREAM)*, which uses IPv4 and TCP protocols for establishing the connection. It then binds the socket to a port and listens for incoming client connections. When a client sends a request, the server accepts the connection and retrieves the HTTP request message. The server parses this request to extract the requested filename. If the file exists, the server reads its contents and sends the HTML text used for rendering the web page along with a *200 OK* response. If the file is not found, the server returns a *404 Not Found* error message. Once the server has either served the file or responded with the error, it closes the connection with the client and terminates the program.

Client Report

Below is a screenshot of the successfully established connection on the Chrome browser. The connection uses port number 6789.

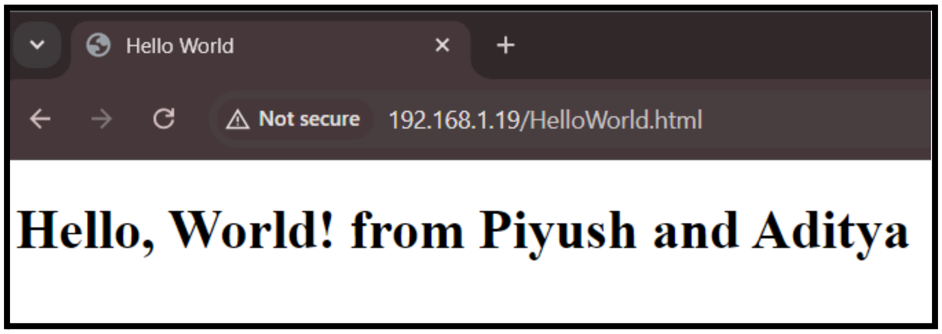


192.168.1.19:6789/HelloWorld.html

Hello, World! from Piyush and Aditya

Successful Connection on Port Number 6789

Below is a screenshot of a similar connection using port number 80, which is the default port number. Note that we established a successful connection without mentioning the port number in the URL this time.

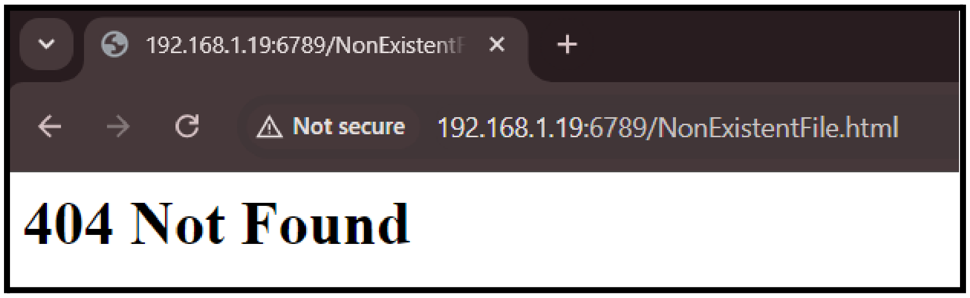


192.168.1.19/HelloWorld.html

Hello, World! from Piyush and Aditya

Successful Connection on Port Number 80

When we request to access a file that is not present in the system, we get an HTML page with a *404 Not Found* error message.



192.168.1.19:6789/NonExistentFile.html

404 Not Found

Unsuccessful Connection trying to access NonExistentFile.html

Part 2: Multi-Threaded TCP Web Server

Functionality

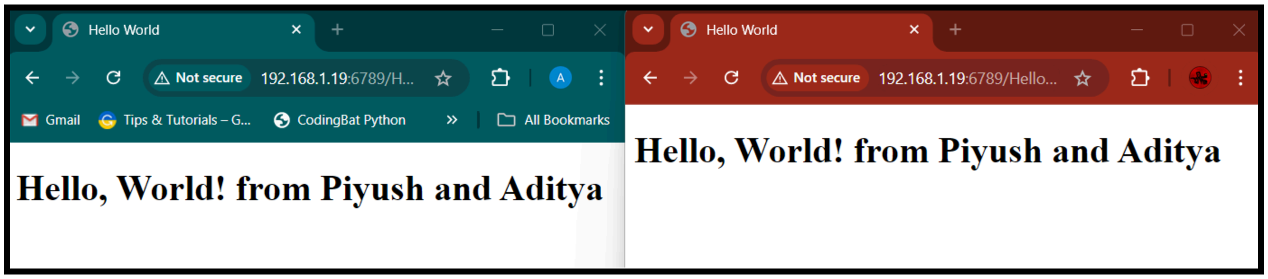
This part builds a multi-threaded web server using Python's `socket` and `threading` modules. The server is designed to handle multiple client requests concurrently by assigning a new thread for each client connection. This enhances the server's ability to manage multiple clients simultaneously without blocking other connections.

Implementation

The server side works in a similar fashion to the previous part. However, instead of operating on a single thread, it uses multi-threading. Once a client connects, the server accepts the connection and spawns a new thread using Python's *`threading.Thread()`* function. Each client request is handled by a dedicated thread, identified by its unique thread ID. We print the thread ID on the terminal to track which thread handles which request.

Client Report

Running a multi-threaded server helps us send multiple requests simultaneously on different Chrome IDs and browsers. A demonstration of the same is shown below.

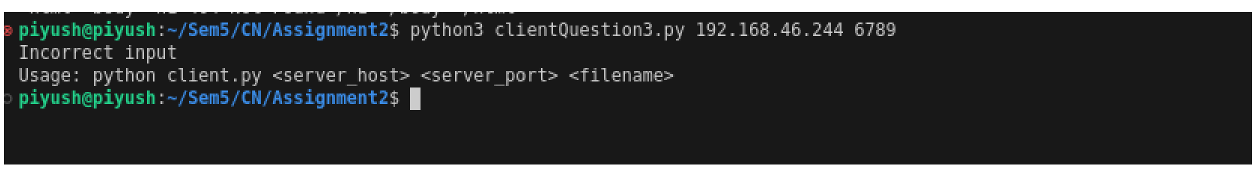


Hello, World! from Piyush and Aditya

Hello, World! from Piyush and Aditya

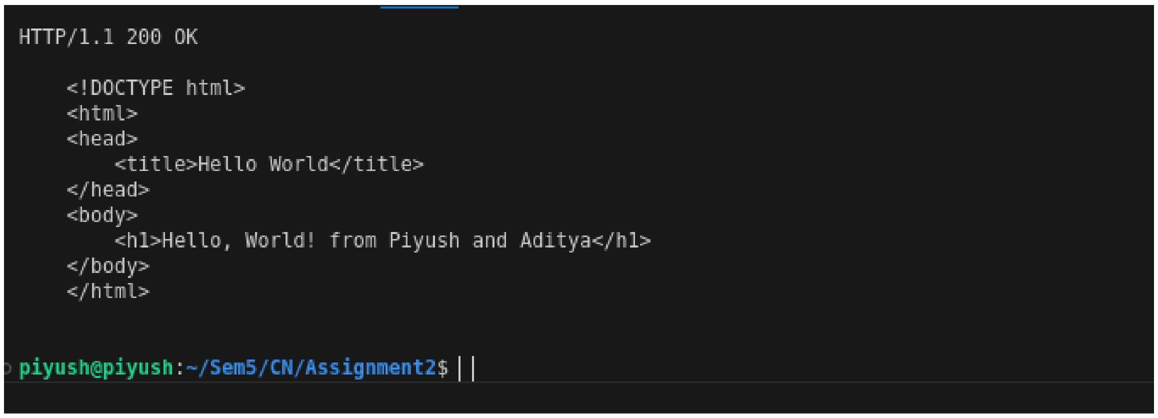
Part 3: HTTP Client to Test Server

The code below shows how, after the incorrect input format is sent to the terminal, it returns that the **incorrect input format is being sent.**



```
piyush@piyush:~/Sem5/CN/Assignment2$ python3 clientQuestion3.py 192.168.46.244 6789
Incorrect input
Usage: python client.py <server_host> <server_port> <filename>
piyush@piyush:~/Sem5/CN/Assignment2$
```

The code below uses the command **python3 clientQuestion3.py 192.168.46.244 6789 HelloWorld.html,** which is executed to run a Python script named clientQuestion3.py, which acts as an HTTP client. This client connects to a server located at the IP address 192.168.46.244 on port 6789, requesting the file HelloWorld.html. Upon successfully establishing the connection and sending the HTTP GET request, the client receives an HTTP response indicating a status of **200 OK**, signifying that the request was successful. The body of the response contains a simple HTML document, which includes a title of "**Hello World" and a heading that states, "Hello, World! from Piyush and Aditya.**" This output demonstrates the ability of the client to interact with a server, retrieve a specific resource, and display the returned HTML content, showcasing a basic implementation of HTTP communication.

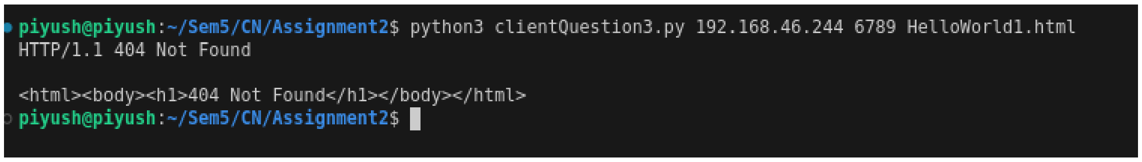


```
HTTP/1.1 200 OK

    <!DOCTYPE html>
    <html>
    <head>
        <title>Hello World</title>
    </head>
    <body>
        <h1>Hello, World! from Piyush and Aditya</h1>
    </body>
    </html>


piyush@piyush:~/Sem5/CN/Assignment2$ |
```

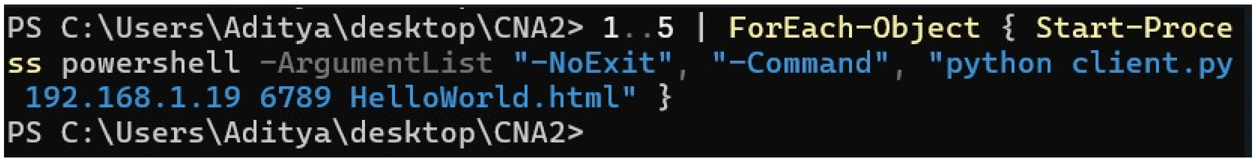
This Output demonstrates that on giving the wrong filename, it returns 404 Not Found. Showing that it couldn't find that filename.



```
piyush@piyush:~/Sem5/CN/Assignment2$ python3 clientQuestion3.py 192.168.46.244 6789 HelloWorld1.html
HTTP/1.1 404 Not Found

<html><body><h1>404 Not Found</h1></body></html>
piyush@piyush:~/Sem5/CN/Assignment2$
```

This command runs five terminals at once and runs the command below.

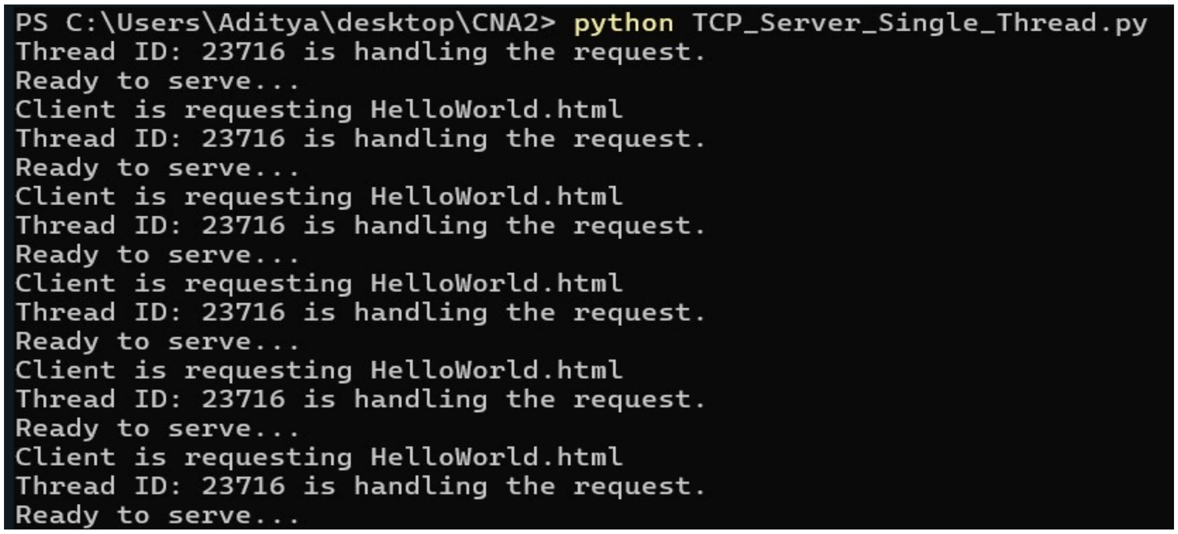


```
PS C:\Users\Aditya\desktop\CNA2> 1..5 | ForEach-Object { Start-Proce
ss powershell -ArgumentList "-NoExit", "-Command", "python client.py
 192.168.1.19 6789 HelloWorld.html" }
PS C:\Users\Aditya\desktop\CNA2>
```

For Part 1:

The below code checks the thread ID and confirms that all the terminals run on the same thread for a single thread server.

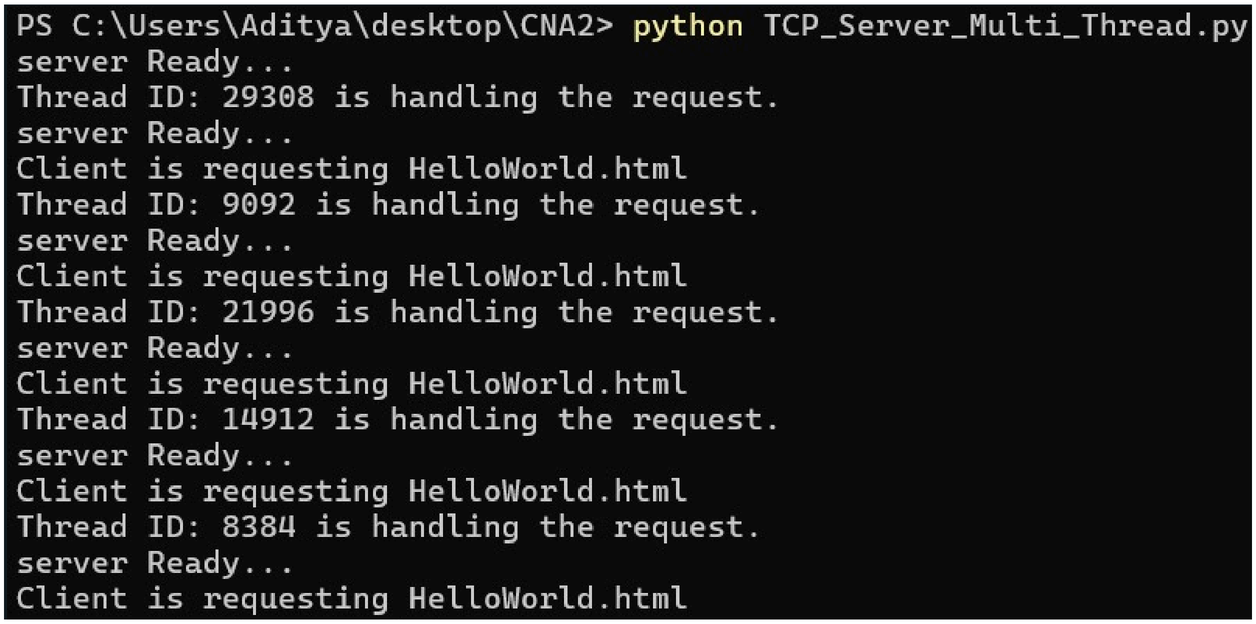
Since the server is single-threaded, it cannot handle multiple requests in parallel. Instead, it processes one request completely before moving on to the next. This ensures that the order of requests is maintained, but it may lead to delays if a request takes a long time to process.



```
PS C:\Users\Aditya\desktop\CNA2> python TCP_Server_Single_Thread.py
Thread ID: 23716 is handling the request.
Ready to serve...
Client is requesting HelloWorld.html
Thread ID: 23716 is handling the request.
Ready to serve...
Client is requesting HelloWorld.html
Thread ID: 23716 is handling the request.
Ready to serve...
Client is requesting HelloWorld.html
Thread ID: 23716 is handling the request.
Ready to serve...
Client is requesting HelloWorld.html
Thread ID: 23716 is handling the request.
Ready to serve...
Client is requesting HelloWorld.html
Thread ID: 23716 is handling the request.
Ready to serve...
```

For Part B:

Multi-threading involves running multiple threads simultaneously within a single process. Each thread represents a separate flow of control, allowing tasks to be executed concurrently.



```
PS C:\Users\Aditya\desktop\CNA2> python TCP_Server_Multi_Thread.py
server Ready...
Thread ID: 29308 is handling the request.
server Ready...
Client is requesting HelloWorld.html
Thread ID: 9092 is handling the request.
server Ready...
Client is requesting HelloWorld.html
Thread ID: 21996 is handling the request.
server Ready...
Client is requesting HelloWorld.html
Thread ID: 14912 is handling the request.
server Ready...
Client is requesting HelloWorld.html
Thread ID: 8384 is handling the request.
server Ready...
Client is requesting HelloWorld.html
```

In a multi-threaded server, when multiple clients connect simultaneously, each request can be handled by a different thread. For example:

- Client A connects and is assigned Thread 1.
- Client B connects and is assigned Thread 2.
- Client C connects and is assigned Thread 3.

Multi-threading allows a server to handle many requests simultaneously, improving responsiveness and throughput. This is particularly beneficial for I/O-bound tasks, where the server can perform other operations while waiting for I/O operations (like file reads or network requests) to complete.

Conclusion

The provided client is designed to send a GET request to a server and is compatible with both the single-threaded server implemented in Part 1 and the multi-threaded server implemented in Part 2.

- **Part 1 (Single-Threaded Server)**: The client will work effectively, as it will establish a connection to the server, send the GET request, and receive the response. However, since the server can handle only one request at a time, subsequent requests from other clients will be queued until the current request is processed and completed.

- **Part 2 (Multi-Threaded Server)**: The client will also function properly with the multi-threaded server, which can process multiple requests concurrently. In this scenario, each GET request the client sends will be handled in a separate thread, allowing for improved responsiveness and performance. Different thread IDs will be assigned to each request, enabling simultaneous handling of multiple clients.