

# ACN Programming Assignment Report

## HTTP Client for Web Proxy and Web Server ( Client.py):

This Python script implements a simple HTTP client that can connect to either a web proxy server or a web server directly. It sends HTTP GET requests to the server and retrieves and displays the server responses. If the requested object is an HTML file, the client parses it for references to other objects and fetches them sequentially.

### Functions/ Modules:

#### **1. *send\_http\_request\_to\_proxy(proxy\_host, proxy\_port, server\_host, server\_port, path):***

- This function is responsible for connecting to the web proxy server and sending an HTTP GET request to the specified server via the proxy.
- It constructs the request, sends it, and receives the response. The response is then returned as a string.

#### **2. *send\_http\_request\_server(host, port, path):***

- This function connects directly to a web server and sends an HTTP GET request to it. It constructs the request, sends it, receives the response, and returns it as a string.

### **Main Function:**

- The **main()** function is the entry point for the program. It takes user input to determine whether to use a web proxy or connect directly to the web server.
- The user is prompted for relevant information such as proxy and server addresses, ports, and the file path. It then calls the appropriate function to send the HTTP request and displays the response.
- If the requested object is an HTML file, the script parses it for references to other objects (links) and fetches them one by one, displaying the responses.

### **Usage:**

1. Run the script, and it will prompt you for the following information:

- Whether you want to use a web proxy (Y/N).
- If you choose to use a web proxy:
  - Proxy Address

- Proxy Port
- Server Address
- Server Port
- File Path
- If you choose not to use a web proxy:
  - Server Address
  - Server Port
  - File Path

2. The script will connect to the specified server (either directly or via the proxy), send the HTTP GET request, and display the server's response.

3. If the requested object is an HTML file, the script will parse it for references to other objects (links) and fetch them sequentially, displaying the responses.

## Output:

**Test case 1:** The client requested an HTML File and the server responded the requested HTML file. Port no. and server IP is explicitly given as input to the client. This is using terminal

```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

[REQUEST SERVED]
[CONNECTED] Accepted connection from ('192.168.135.246', 34834)
[CONNECTED] Accepted connection from ('192.168.135.246', 34842)
[REQUEST SERVED]
[REQUEST SERVED]
[CONNECTED] Accepted connection from ('192.168.135.246', 34852)
[REQUEST SERVED]
[CONNECTED] Accepted connection from ('192.168.135.246', 58804)
[REQUEST SERVED]
[CONNECTED] Accepted connection from ('192.168.135.246', 58820)
[CONNECTED] Accepted connection from ('192.168.135.246', 58832)
[REQUEST SERVED]
[REQUEST SERVED]
[CONNECTED] Accepted connection from ('192.168.135.246', 58848)
[REQUEST SERVED]
[CONNECTED] Accepted connection from ('192.168.135.246', 33912)
[REQUEST SERVED]
[CONNECTED] Accepted connection from ('192.168.135.246', 33920)
[CONNECTED] Accepted connection from ('192.168.135.246', 33926)
[REQUEST SERVED]
[CONNECTED] Accepted connection from ('192.168.135.246', 33934)
[REQUEST SERVED]
[REQUEST SERVED]
[CONNECTED] Accepted connection from ('192.168.135.246', 33692)
[REQUEST SERVED]
[CONNECTED] Accepted connection from ('192.168.135.246', 33708)
[CONNECTED] Accepted connection from ('192.168.135.246', 33722)
[REQUEST SERVED]
[REQUEST SERVED]
[CONNECTED] Accepted connection from ('192.168.135.246', 33734)
[REQUEST SERVED]
[CONNECTED] Accepted connection from ('192.168.135.246', 36750)
[REQUEST SERVED]

● dnyan@dnyaneshwar-workstation:~/Project/Socket_Prog/prog ass$ python3 client.py
Do want to use Web Proxy Server (Y/N):N
Server Address:192.168.135.246
Server Port:15200
File Path:index.html
[CONNECTED] Connected To the Web Server
HTTP/1.1 200 OK
Content-Length: 652

<!DOCTYPE html>
<html>
<head>
  <title>Sample HTML</title>
  <link rel="stylesheet" type="text/css" href="styles.css">
</head>
<body>
  <h1>Welcome to My Web Page</h1>
  <p>This is a sample HTML page.</p>

  <p><a href="home.html">Home (index.html)</a></p>
  <p><a href="about.html">about (index.html)</a></p>
  <p><a href="services.html">services</a></p>

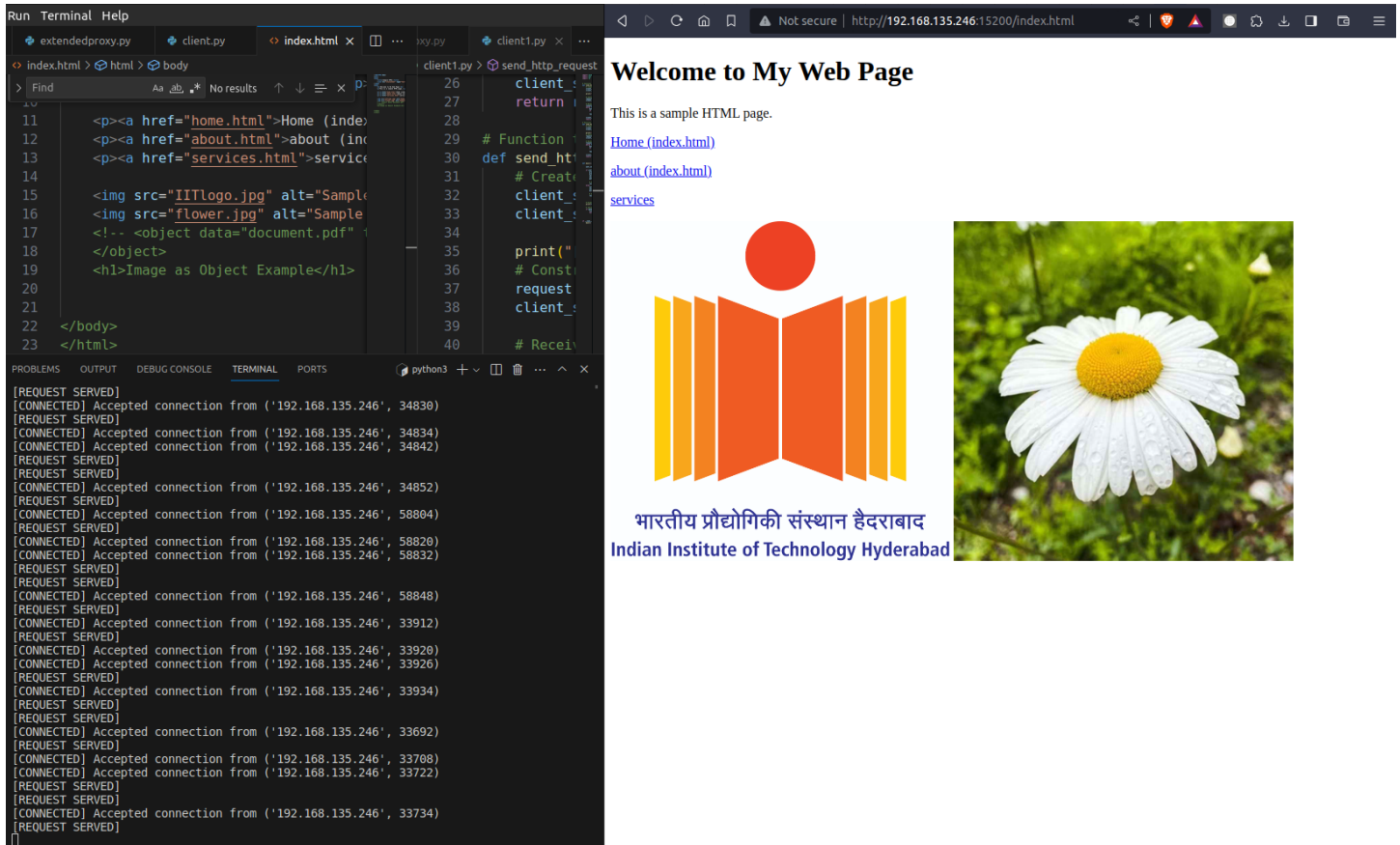
  
  
  <!-- <object data="document.pdf" type="application/pdf">
  </object>
  <h1>Image as Object Example</h1>

</body>
</html>

○ dnyan@dnyaneshwar-workstation:~/Project/Socket_Prog/prog ass$

```

**Test case 2:** The client requested an HTML File and the server responded the requested HTML file. Port no. and server IP is explicitly given as input to the client. This is using Web Browser.



## 2. Simple Web Proxy with Multi-Threaded Proxy Server (Proxy.py):

This Python script implements a basic web proxy that can intercept and forward HTTP requests and responses between a client and a target server. The web proxy performs the first role of proxying but does not involve caching. It is designed to handle "GET" requests, which specify the URL for retrieving objects. The proxy server itself is multi-threaded, allowing it to handle multiple client connections concurrently.

### **Key Modules:**

- **handle\_server(path, server\_add, server\_port, queue) Function:**
- This function is responsible for connecting to the target server specified in the client's HTTP request.

- It establishes a new socket connection to the server, sends an HTTP GET request to retrieve the requested object, and receives the server's response.
- The response is stored in a queue to be retrieved by the client handler.
- This function is run in a separate thread for each client request.

→ **handle\_client(client\_socket) Function:**

- The 'handle\_client' function is responsible for handling individual client connections.
- It receives the client's HTTP request, and parses the request to extract the requested object's path, server address, and server port.
- A new thread is created to connect to the target server using the 'handle\_server' function. This allows concurrent proxying of multiple client requests.
- The client handler waits for the thread connecting to the server to complete and retrieves the server's response from the queue.
- The response is then sent back to the client.

**3. start\_proxy\_server() Function:**

- The 'start\_proxy\_server' function is the main entry point of the script.
- It creates a socket, binds it to a specified IP address and port, and listens for incoming client connections.
- When a client connection is accepted, a new thread is spawned to handle the client's request, ensuring concurrent processing of multiple clients.

**Configuration:**

**Proxy Server's Address and Port:**

- The proxy server's IP address and port are configured as 'HOST' and 'PORT', respectively, at the beginning of the script.

**Usage:**

1. Run the proxy server script.
2. Configure your client (e.g., a web browser) to use the proxy server by specifying the proxy's IP address ('HOST') and port ('PORT') in the client's settings.
3. When the client sends HTTP requests, the proxy server intercepts them, forwards them to the target server specified in the request, and relays the responses back to the client.
4. The proxy handles various HTTP response codes and conditional requests transparently.

This code provides a foundation for a simple web proxy and multi-threaded proxy server, allowing concurrent handling of client connections and enhancing the responsiveness and scalability of the proxy. It can be extended to offer additional features in future iterations.

## Output:

**Test case 1:** The client requested an HTML File and the server responded to the requested HTML file. In between a web proxy server is used which takes the request from the client and then requests the server to send that particular file. This is using Terminal.

```
[REQUEST SERVED]
[CONNECTED] Accepted connection from ('192.168.135.246', 348
52)
[REQUEST SERVED]
[CONNECTED] Accepted connection from ('192.168.135.246', 588
04)
[REQUEST SERVED]
[CONNECTED] Accepted connection from ('192.168.135.246', 588
20)
[CONNECTED] Accepted connection from ('192.168.135.246', 588
32)
[REQUEST SERVED]
[REQUEST SERVED]
[CONNECTED] Accepted connection from ('192.168.135.246', 588
48)
[REQUEST SERVED]
[CONNECTED] Accepted connection from ('192.168.135.246', 339
12)
[REQUEST SERVED]
[CONNECTED] Accepted connection from ('192.168.135.246', 339
20)
[CONNECTED] Accepted connection from ('192.168.135.246', 339
26)
[REQUEST SERVED]
[CONNECTED] Accepted connection from ('192.168.135.246', 339
34)
[REQUEST SERVED]
[REQUEST SERVED]
[CONNECTED] Accepted connection from ('192.168.135.246', 336
92)
[REQUEST SERVED]
[CONNECTED] Accepted connection from ('192.168.135.246', 337
08)
[CONNECTED] Accepted connection from ('192.168.135.246', 337
22)
[REQUEST SERVED]
[REQUEST SERVED]
[CONNECTED] Accepted connection from ('192.168.135.246', 337
34)
[REQUEST SERVED]
[CONNECTED] Accepted connection from ('192.168.135.246', 367
50)
[REQUEST SERVED]
[CONNECTED] Accepted connection from ('192.168.135.246', 368
32)
[REQUEST SERVED]

dnyan@dnyaneshwar-workstation:~/Project/Socket_Prog/prog ass$ pyth
on3 webproxy.py
[LISTENING] Web ProxyServer is listening on 127.0.0.2:12347
[CONNECTED] Accepted connection from ('127.0.0.1', 45158)
[CONNECTED] Connected to the server on ('192.168.135.246,15200')
[
]

dnyan@dnyaneshwar-workstation:~/Project/Socket_Prog/
• prog ass$ python3 client.py
Do want to use Web Proxy Server (Y/N):Y
Proxy Address:127.0.0.2
Proxy Port:12347
Server Address:192.168.135.246
Server Port:15200
File Path:index.html
[CONNECTED] Connected To the Web Proxy Server
HTTP/1.1 200 OK
Content-Length: 652

<!DOCTYPE html>
<html>
<head>
  <title>Sample HTML</title>
  <link rel="stylesheet" type="text/css" href="sty
les.css">
</head>
<body>
  <h1>Welcome to My Web Page</h1>
  <p>This is a sample HTML page.</p>

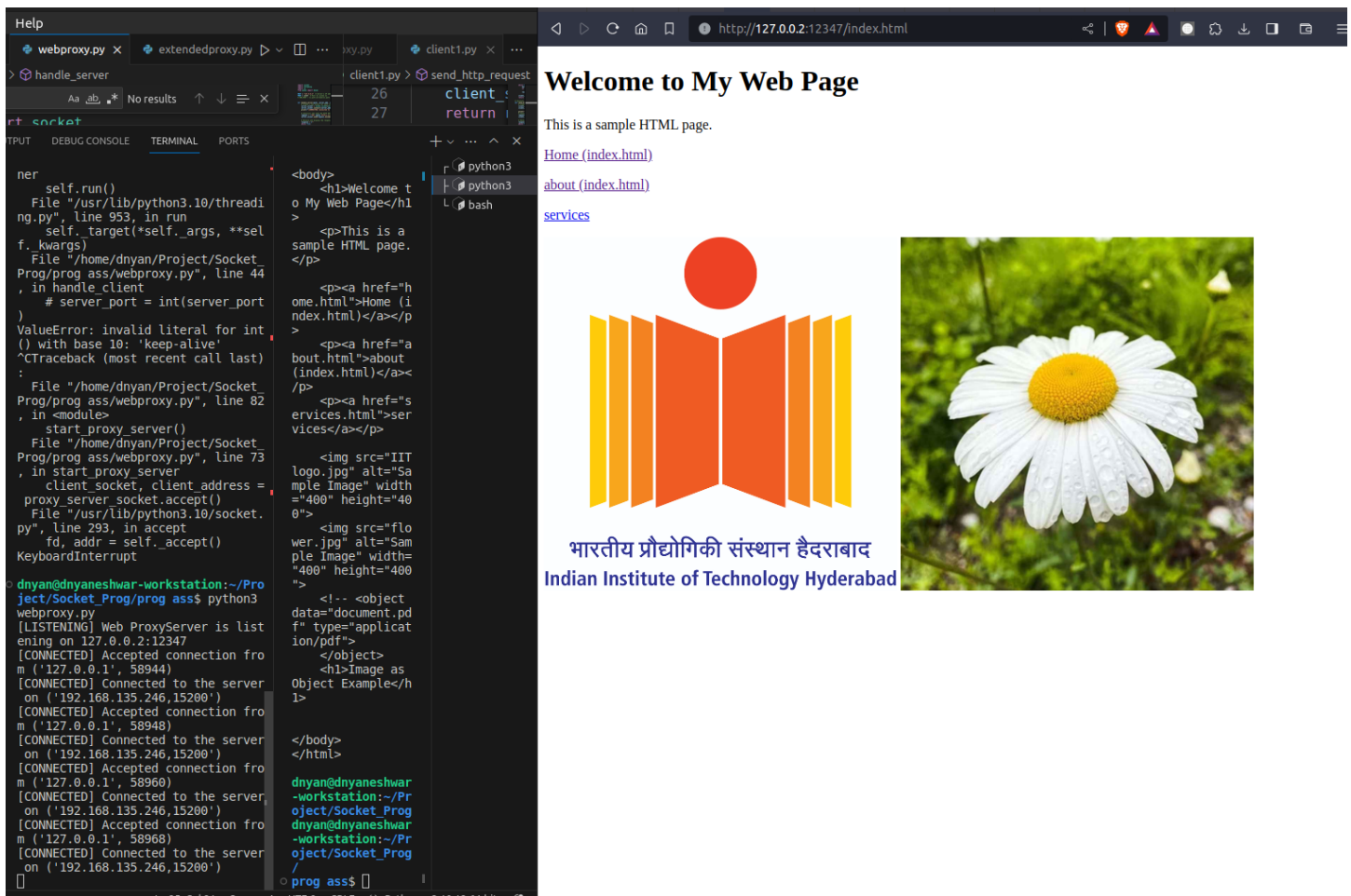
  <p><a href="home.html">Home (index.html)</a></p>
  <p><a href="about.html">about (index.html)</a></
p>
  <p><a href="services.html">services</a></p>

  
  
  <!-- <object data="document.pdf" type="applicati
on/pdf">
  </object>
  <h1>Image as Object Example</h1>

</body>
</html>

dnyan@dnyaneshwar-workstation:~/Project/Socket_Prog/
• prog ass$ [
```

**Test case 2:** The client requested an HTML File and the server responded to the requested HTML file. In between a web proxy server is used which takes the request from the client and then requests the server to send that particular file. This is using Web Browser.



The screenshot displays a web browser window on the right and a terminal window on the left. The browser shows a page titled "Welcome to My Web Page" with a URL of `http://127.0.0.2:12347/index.html`. The page content includes a greeting, a description of a sample HTML page, and three links: "Home (index.html)", "about (index.html)", and "services". Below the links are two images: a stylized orange and yellow logo of the Indian Institute of Technology Hyderabad, and a photograph of a white daisy flower with a yellow center. The terminal window on the left shows the execution of a Python script named `webproxy.py`. The script is running on `127.0.0.2:12347` and is handling multiple concurrent connections. The output shows the server listening, accepting connections from `127.0.0.1`, and successfully serving the requested HTML file to the client.

### 3. Multi-Threaded Web Server ( server.py )

- This Python script implements a multi-threaded web server capable of handling multiple HTTP requests concurrently. The server listens on a specified IP address and port, servicing client requests in separate threads.
- When a requested file is found, the server responds with an 'HTTP 200 OK' status code and sends the file content.

- If the requested file is not found, the server responds with an 'HTTP 404 Not Found' status code.

## **Thread Utilization:**

Threads are used to manage concurrent client connections, allowing multiple HTTP requests to be processed in parallel. The threading approach in this server involves two key functions:

### **→ handle\_client(client\_socket) Function**

- The 'handle\_client' function is responsible for handling individual client connections. Upon client connection, a new thread is created to manage the client's request.
- The function parses the HTTP request, checks if the requested file exists, and creates an HTTP response message with the requested file's content preceded by header lines.
- If the requested file is found, a "200 OK" response is sent with the file's content. In the case of a missing file, a "404 Not Found" response is sent.

### **→ start\_server() Function:**

- The 'start\_server' function serves as the main entry point for the script. It creates a socket, binds it to the specified IP address and port, and listens for incoming client connections.
- When a client connection is accepted, a new thread is spawned to handle that client's request.

By using threads, the server can efficiently process multiple client connections simultaneously, ensuring that the server remains responsive to new client connections. Each client connection is handled in its own thread, allowing for seamless concurrency without blocking other clients or the server's main functionality.

## **Usage:**

1. Run the server script.
2. Determine the IP address and port to which the server is bound.
3. Open a web browser or client application and provide the corresponding URL. For example:
  4. `http://127.0.0.3:15200/HelloWorld.html``
  5. Replace the IP address and port with the actual server's configuration.

6. 'HelloWorld.html' represents the name of the file placed in the server directory. You can request any file in the server's root directory.

4. The browser or client should display the contents of the requested file. If you request a file that is not present on the server, you will receive an "HTTP 404 Not Found" message.

This code provides a basic example of a multi-threaded web server for handling HTTP requests. You can expand its functionality, customize the server's root directory, or add additional features as needed for your specific application.

#### **4. Web Proxy with URL Prefetching(ExtendedProxy.py):**

This Python script implements a web proxy that not only forwards HTTP requests and responses between a client and a target server but also includes a URL prefetching mechanism to improve browsing speed. URL prefetching involves parsing requested pages for links and fetching those linked pages in the background, storing them in the cache for faster retrieval.

#### **Key Module/ Component:**

##### **1. Caching System:**

- The script uses a dictionary, `'html_cache'`, to store cached web pages with their full HTML content. This cache helps in storing and serving previously fetched web pages without contacting the remote server.

##### **2. Prefetching Web Pages:**

- The `fetch_web_page(url)` function is responsible for fetching web pages. It first checks if the page is already cached and returns it if found.
- If the page is not cached, it sends an HTTP GET request to the web server to fetch the page. The fetched page is then cached for future use.

##### **3. Extracting and Caching Links:**

- The `extract_links_and_cache(html_content, base_url)` function parses the HTML content of a page and extracts HTTP links. It then fetches and caches these linked pages in the background.
- The function uses the *BeautifulSoup* library for HTML parsing.

##### **4. Multi-Threaded Proxy Server:**

- The proxy server is multi-threaded, allowing it to handle multiple client connections concurrently.



- It listens for incoming client connections, and for each connection, it spawns a new thread to handle the client's request.
- The use of threads enhances the responsiveness and scalability of the proxy server.

### **5. Caching Linked Pages:**

- The `cache_linked_page(url)` function is called in a separate thread to fetch and cache linked pages.
- It uses the `fetch_web_page(url)` function to fetch linked pages and adds them to the cache.

### **Configuration:**

#### **Proxy Server's Address and Port:**

- The proxy server's IP address and port are configured as `'HOST'` and `'PORT'`, respectively, at the beginning of the script.

### **Usage:**

1. Run the proxy server script.
2. Configure your client (e.g., a web browser) to use the proxy server by specifying the proxy's IP address (`'HOST'`) and port (`'PORT'`) in the client's settings.
3. When the client sends HTTP requests, the proxy server intercepts them, forwards them to the target server specified in the request, and relays the responses back to the client.
4. The proxy server prefetches linked pages in the background, storing them in the cache for faster retrieval when requested by the client.

This code provides a web proxy with URL prefetching capabilities, enhancing the browsing experience by reducing page load times through the caching of linked pages. The multi-threaded proxy server efficiently handles multiple client connections, ensuring responsiveness and scalability.

## Output:

**Test case 1:** In the extended web proxy, there is caching of the web pages whose links are present on the index.html file. This is using terminal.

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

[REQUEST SERVED]
[CONNECTED] Accepted connection from ('192.168.135.246', 55866)
[REQUEST SERVED]
[CONNECTED] Accepted connection from ('192.168.135.246', 37408)
[REQUEST SERVED]
[CONNECTED] Accepted connection from ('192.168.135.246', 41642)
[REQUEST SERVED]
[CONNECTED] Accepted connection from ('192.168.135.246', 47886)
[REQUEST SERVED]
[CONNECTED] Accepted connection from ('192.168.135.246', 47888)
[REQUEST SERVED]
[CONNECTED] Accepted connection from ('192.168.135.246', 47904)
[REQUEST SERVED]
[CONNECTED] Accepted connection from ('192.168.135.246', 47916)
[REQUEST SERVED]
[CONNECTED] Accepted connection from ('192.168.135.246', 47928)
[REQUEST SERVED]
[CONNECTED] Accepted connection from ('192.168.135.246', 48064)
[REQUEST SERVED]
[CONNECTED] Accepted connection from ('192.168.135.246', 48070)

458, in prepare_request
  File "/usr/lib/python3/dist-packages/requests/models.py", line 316, in prepare
    return session.request(method=method, url=url, **kwargs)
  File "/usr/lib/python3/dist-packages/requests/sessions.py", line 530, in request
    self.prepare_url(url, params)
  File "/usr/lib/python3/dist-packages/requests/models.py", line 84, in prepare_url
    p.prepare()
  File "/usr/lib/python3/dist-packages/requests/models.py", line 16, in prepare
    raise InvalidURL(*e.args)
requests.exceptions.InvalidURL: Failed to parse: http://192.168.135.246:15200index.html/services.html
  self.prepare_url(url, params)
  File "/usr/lib/python3/dist-packages/requests/models.py", line 84, in prepare_url
    raise InvalidURL(*e.args)
requests.exceptions.InvalidURL: Failed to parse: http://192.168.135.246:15200index.html/home.html
  prep = self.prepare_request(req)
  File "/usr/lib/python3/dist-packages/requests/sessions.py", line 458, in prepare_request
    p.prepare()
  File "/usr/lib/python3/dist-packages/requests/models.py", line 16, in prepare
    self.prepare_url(url, params)
  File "/usr/lib/python3/dist-packages/requests/models.py", line 84, in prepare_url
    raise InvalidURL(*e.args)
requests.exceptions.InvalidURL: Failed to parse: http://192.168.135.246:15200index.html/about.html
[CONNECTED] Accepted connection from ('127.0.0.1', 44890)
[CONNECTED] Connected to the server on ('192.168.135.246,15200')
[CONNECTED] Accepted connection from ('127.0.0.1', 43724)
[CONNECTED] Connected to the server on ('192.168.135.246,15200')
[CONNECTED] Accepted connection from ('127.0.0.1', 36658)
[CONNECTED] Connected to the server on ('192.168.135.246,15200')

</body>
</html>

dnyan@dnyaneshwar-workstation:~/Project/Socket_Prog/prog ass$ python3 client6.py
Do want to use Web Proxy Server (Y/N):Y
File Path:services.html
[CONNECTED] Connected To the Web Proxy Server
HTTP/1.1 200 OK
Content-Length: 132

<!DOCTYPE html>
<html>
<head>
<title>About us</title>
</head>
<body>
<h1>This is the services page</h1>
</body>
</html>

dnyan@dnyaneshwar-workstation:~/Project/Socket_Prog/prog ass$ python3 client6.py
Do want to use Web Proxy Server (Y/N):Y
File Path:about.html
[CONNECTED] Connected To the Web Proxy Server
HTTP/1.1 200 OK
Content-Length: 131

<!DOCTYPE html>
<html>
<head>
<title>About us</title>
</head>
<body>
<h1>We are IITH NIS students</h1>
</body>
</html>

dnyan@dnyaneshwar-workstation:~/Project/Socket_Prog/prog ass$
```

**Test case 2:** In the extended web proxy, there is caching of the web pages whose links are present on the index.html file. This is using the web browser.

```
Run Terminal Help
webproxy6.py webproxy7.py client6.py extendedproxy.py
extendedproxy.py > handle_server
1 import socket
2 import threading
3 from queue import Queue
4
5 HOST = "127.0.0.2" # Server's IP address
6 PORT = 12348 # Port to listen on
7
8 # Create a dictionary to store cached pages
9 cache = {}
10
11 def handle_server(path, server_add, server_port, queue):
12     # Create a new socket for each request
13     server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
14     server_socket.connect((server_add, server_port))
15     print(f"[CONNECTED] Connected to the server on '{server_add}'")
16
17     # Construct and send the HTTP GET request
18     request = f"GET {path} HTTP/1.1\r\nHost: {server_add}\r\n"
19     server_socket.send(request.encode())
20
21     # Receive and display the response
22     response = b""
23     while True:
24         data = server_socket.recv(1024)
25         if not data:
26             break

Hello, World!

http://127.0.0.2:12348/home.html

<!DOCTYPE html>
<html>
<head>
<title>About us</title>
</head>
<body>
<h1>This is the services page</h1>
</body>
</html>

dnyan@dnyaneshwar-workstation:~/Project/Socket_Prog/prog ass$
```

We certify that this assignment/report is our own work, based on our personal study and/or research and that we have acknowledged all material and sources used in its preparation, whether they be books, articles, packages, datasets, reports, lecture notes, and any other kind of document, electronic or personal communication. We also certify that this assignment/report has not previously been submitted for assessment/project in any other course lab, except where specific permission has been granted from all course instructors involved, or at any other time in this course, and that we have not copied in part or whole or otherwise plagiarized the work of other students and/or persons. Additionally, we acknowledge that we may have used AI tools, such as language models (e.g., ChatGPT, Bard), for assistance in generating and refining my assignment, and we have made all reasonable efforts to ensure that such usage complies with the academic integrity policies set for the course. I pledge to uphold the principles of honesty and responsibility at CSE@IITH. In addition, we understand our responsibility to report honour violations by other students if we become aware of it.

Dnyaneshwar Shinde <cs23mtech12003>

Arjit Gupta <cs23mtech12001>

Banala Nishith Reddy <ee22mtech11001>

:

Date: 5/11/2023

Signatures: <Dnyaneshwar>