Project submitted to the

SRM University – AP, Andhra Pradesh

Submitted in partial fulfillment of the requirement for the award of the degreeof

Bachelor of Technologyin

Computer Science and Engineering

School of Engineering and Sciences

Submitted by:

BOGINENI KRISHNA	**AP21110010619**
VENNELA ROHAN	**AP21110010620**
NAYUDU RAVINDRANATH	**AP21110010621**

Under the guidance of: Krishna Siva Prasad Mudigonda

Department of Computer Science and Engineering SRM University,AP

[05-2023]

PROJECT TITLE:

Multi-threaded Proxy Server and Client.

TABLE OF CONTENTS:

1.AIM

The aim of this project is to develop a multi-threaded proxy server and client system that allows the client to request any URL, fetch the corresponding web content using the server, and save the output as an HTML page with an appropriate name. The system aims to provide an efficient and streamlined approach to fetching web content while ensuring concurrent handling of multiple client connections.

2. Description

This project is a simple implementation of a proxy server in Python. The proxy server acts as an intermediary between a client and a remote server, allowing the client to send HTTP requests to the server through the proxy.

The project consists of two parts: the proxy server and the client.

The proxy server:

The server listens on a specified host and port (in this case, '127.0.0.1' and port 8000).

It creates a socket object and binds it to the host and port.
The server listens for incoming connections and accepts client connections.

For each client connection, a new thread is created to handle the client.
The client's URL is received, and a request is made to the URL using the requests library.

The server determines if the request is valid or not and prepares a response containing the URL and validity.

The server logs the request details to a file and sends the response back to the client.

The client:

The client prompts the user to enter a URL.
It creates a socket object and connects to the proxy server.

The client sends the URL to the proxy server.
It receives the response from the proxy server and prints it.

The client logs the request details to a file.
The client repeats the process until the user decides to quit.

3.System Requirements

- 16-bit processor
- Windows 10 or above
- RAM- 8GB minimum
- Python preinstalled and Libraries, including socket, threading, requests, and datetime.

4. Individual members roles and responsibilities:

This project two code blocks:

i)Proxy server

→ Client Handling.

→ Server Setup.

ii)client server

→ NAYUDU RAVINDRANATH (621) : -

His role is to complete the ‘client server’ part and link to the proxy server.

The responsibilities of his role are the following:

Creating a socket object.
Connecting to the proxy server.
Sending the URL to the server.
Saving the request details to a file.
Sending a request to the proxy server.
Making requests until the user decides to quit.

→ VENNELA ROHAN (620): -

His role is to complete the client handling part in the proxy server.

The responsibilities of his role are the following:

Receive the URL and Fetching the output from it.
Preparing the response to send back to the client.
Saving the request details to a file.
Sending the response back to the client.

→ BOGINENI KRISHNA (619): -

His role is to design the server setup code block in the proxy server.

The responsibilities of his role are the following:

Start the proxy server.
Creating a socket object.
Binding the socket to a specific host and port.
Listening for incoming connections.
Accepting a client connection.
Finally Creating a new thread to handle the client.

5.CODE:

#Proxy server.py

```
import socket
import threading
import requests
from datetime import datetime


SERVER_HOST = '127.0.0.1'
SERVER_PORT = 8000
BUFFER_SIZE = 4096


def handle_client(client_socket):

    url = client_socket.recv(BUFFER_SIZE).decode('utf-8')
    try:
        response = requests.get(url)
        valid = True
    except requests.exceptions.RequestException:
        valid = False

    response_data = f"URL: {url}\nValid: {valid}\n"

    with open('requests.log', 'a') as log_file:
        log_file.write(f"Time: {datetime.now()}\nIP Address:
{client_socket.getpeername()[0]}\nURL: {url}\n\n")


    client_socket.sendall(response_data.encode('utf-8'))
    client_socket.close()

def start_proxy_server():

    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)


    server_socket.bind((SERVER_HOST, SERVER_PORT))
    server_socket.listen(5)
    print(f"Proxy server listening on {SERVER_HOST}:{SERVER_PORT}")

    while True:

        client_socket, client_address = server_socket.accept()
        print(f"Accepted connection from {client_address[0]}:{client_address[1]}")


        client_thread = threading.Thread(target=handle_client, args=(client_socket,))
        client_thread.start()


start_proxy_server()
```

#ProxyClient.py

```
import socket
from datetime import datetime

# Proxy server configuration
SERVER_HOST = '127.0.0.1'
SERVER_PORT = 8000
BUFFER_SIZE = 4096

# Send a URL request to the proxy server
def send_request(url):
    # Create a socket object
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    # Connect to the proxy server
    client_socket.connect((SERVER_HOST, SERVER_PORT))

    # Send the URL to the server
    client_socket.sendall(url.encode('utf-8'))

    # Receive and print the response from the server
    response = client_socket.recv(BUFFER_SIZE).decode('utf-8')
    print(response)

    # Save the request details to a file
    with open('requests.log', 'a') as log_file:
        log_file.write(f"Time: {datetime.now()}\nURL: {url}\nResponse: {response}\n\n")
    # Close the socket connection
    client_socket.close()

# Send a request to the proxy server
def make_request():
    url = input("Enter the URL: ")
    send_request(url)

# Make requests until the user decides to quit
# Make requests until the user decides to quit
while True:
    make_request()
    choice = input("Do you want to make another request? (y/n): ")
    if choice.lower() != 'y':
        break
```

6. Explanation

i)proxy server

This code implements a basic proxy server using the Python socket and threading modules. It allows clients to connect to it and fetch the contents of a given URL through the server.

The code can be divided into two main parts: the client handling and the server setup.

→Client Handling:

The handle_client function is responsible for processing client requests.

It receives the client socket as a parameter, representing the connection between the server and the client.

It receives the URL from the client using recv and decodes it as a UTF-8 string.

It then uses the requests library to fetch the output of the provided URL.

If the request is successful (no RequestException is raised), it sets the valid flag to True; otherwise, it sets it to False.

It prepares a response string containing the URL and validity status.

It appends the request details (timestamp, client IP address, and URL) to a log file.

It sends the response back to the client using sendall and closes the client socket.

→Server Setup:

The start_proxy_server function is responsible for setting up the proxy server.

It creates a socket object using socket.socket and specifies the address family (socket.AF_INET) and socket type (socket.SOCK_STREAM).

It binds the socket to a specific host and port using bind, using the values of SERVER_HOST and SERVER_PORT variables.

It starts listening for incoming connections using listen, with a maximum backlog of 5 pending connections.

It enters an infinite loop to continuously accept client connections.

When a client connection is accepted using accept, it prints the client's address.

It creates a new thread using threading.Thread and passes the handle_client function and the client socket as arguments.

The new thread is started with start, allowing the server to handle multiple clients concurrently.

ii)client sever:

This code block implements a simple proxy server that allows users to send URL requests through it.

The code defines the server host, port, and buffer size for communication.

The send_request() function creates a socket, connects to the proxy server, sends a URL request, receives and prints the response, and saves the request details to a log file.

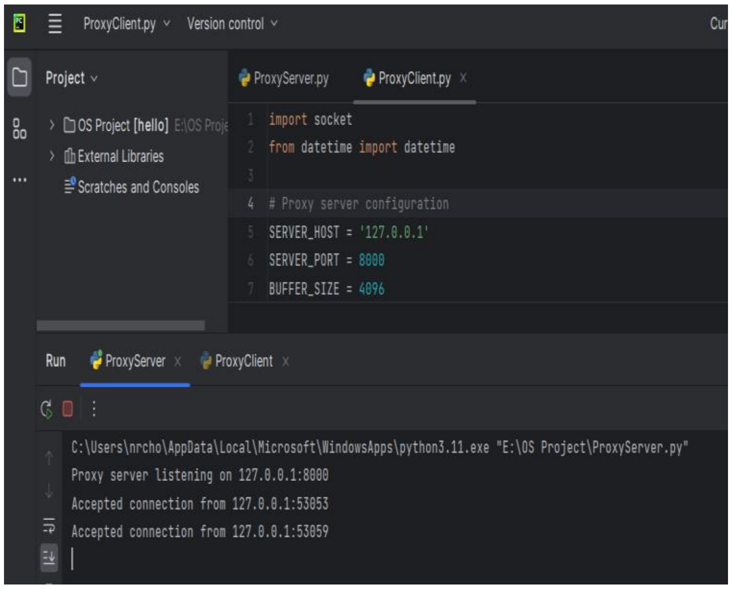
The make_request() function prompts the user to enter a URL and calls send_request() to send the request.

The code then enters a loop where it repeatedly makes requests based on user input and asks if the user wants to make another request.

The loop continues until the user enters 'n' to quit, at which point the program breaks out of the loop and terminates.

7.Output:

ProxyServer:

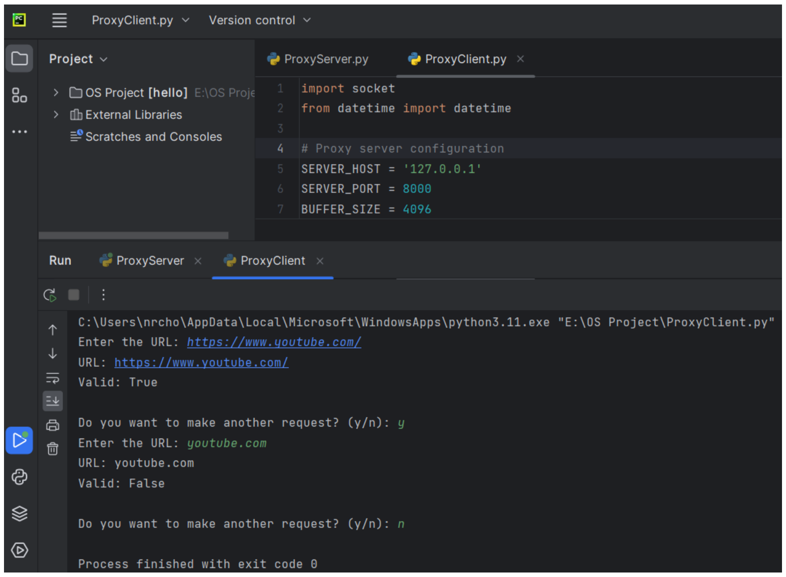


```
import socket
from datetime import datetime

# Proxy server configuration
SERVER_HOST = '127.0.0.1'
SERVER_PORT = 8000
BUFFER_SIZE = 4096
```

```
C:\Users\nrcho\AppData\Local\Microsoft\WindowsApps\python3.11.exe "E:\OS Project\ProxyServer.py"
Proxy server listening on 127.0.0.1:8000
Accepted connection from 127.0.0.1:53053
Accepted connection from 127.0.0.1:53059
```

ProxyClient:



```
import socket
from datetime import datetime

# Proxy server configuration
SERVER_HOST = '127.0.0.1'
SERVER_PORT = 8000
BUFFER_SIZE = 4096
```

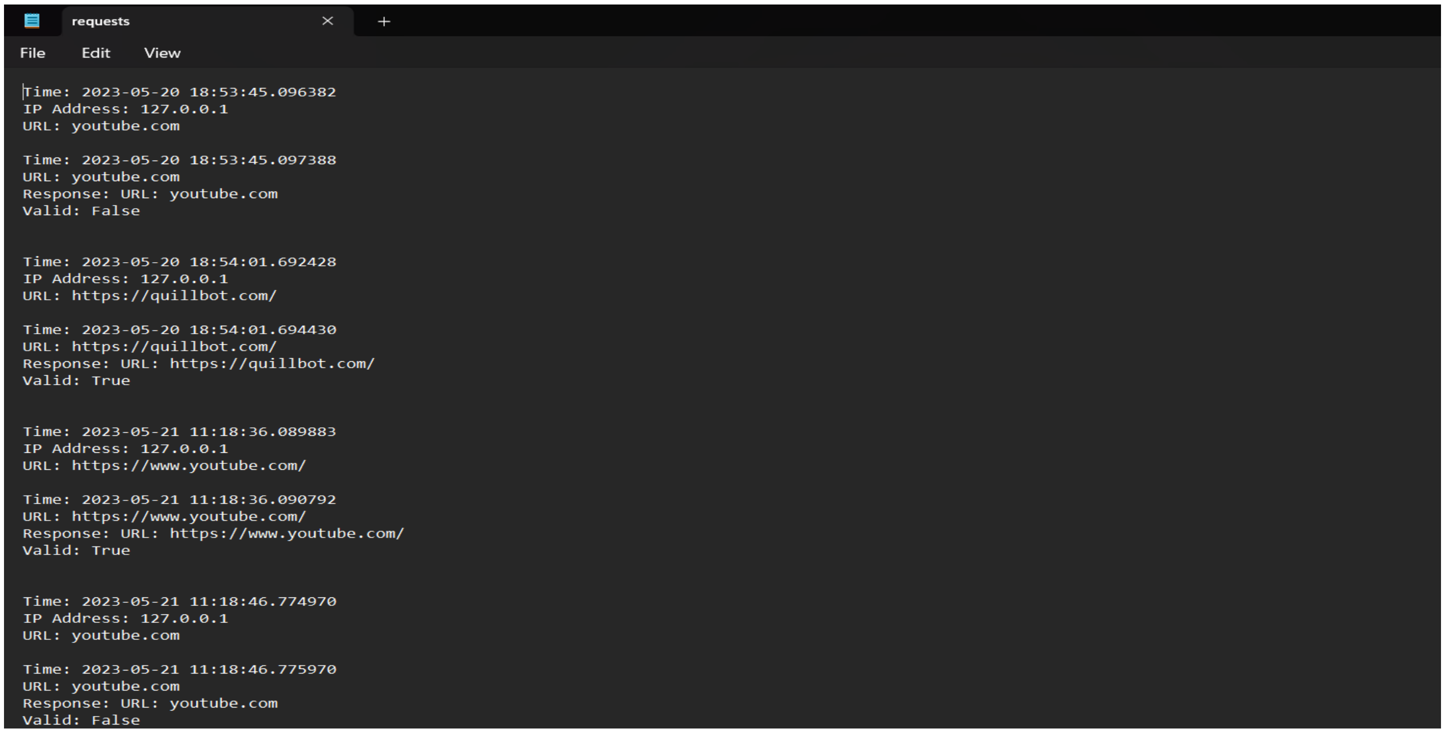
```
C:\Users\nrcho\AppData\Local\Microsoft\WindowsApps\python3.11.exe "E:\OS Project\ProxyClient.py"
Enter the URL: https://www.youtube.com/
URL: https://www.youtube.com/
Valid: True

Do you want to make another request? (y/n): y
Enter the URL: youtube.com
URL: youtube.com
Valid: False

Do you want to make another request? (y/n): n

Process finished with exit code 0
```

Output in textfile:



```
Time: 2023-05-20 18:53:45.096382
IP Address: 127.0.0.1
URL: youtube.com

Time: 2023-05-20 18:53:45.097388
URL: youtube.com
Response: URL: youtube.com
Valid: False


Time: 2023-05-20 18:54:01.692428
IP Address: 127.0.0.1
URL: https://quillbot.com/

Time: 2023-05-20 18:54:01.694430
URL: https://quillbot.com/
Response: URL: https://quillbot.com/
Valid: True


Time: 2023-05-21 11:18:36.089883
IP Address: 127.0.0.1
URL: https://www.youtube.com/

Time: 2023-05-21 11:18:36.090792
URL: https://www.youtube.com/
Response: URL: https://www.youtube.com/
Valid: True


Time: 2023-05-21 11:18:46.774970
IP Address: 127.0.0.1
URL: youtube.com

Time: 2023-05-21 11:18:46.775970
URL: youtube.com
Response: URL: youtube.com
Valid: False
```

8.Future enhancement of Project

→ Implementing caching mechanism to store and retrieve frequently accessed web resources, reducing response time.

→ Enhancing logging functionality to include additional information such as response status codes and client user-agent.

→ Implementing authentication and access control mechanisms to restrict access to the proxy server.

→ Improving error handling by providing detailed error messages and handling exceptions.

9.Conclusion

Finally, this proxy server project provides the implementation of a proxy server in Python. It demonstrates the functionality of handling client requests, making requests to servers using request library. The server logs request details and sends responses back to the clients. It offers potential for further enhancements, such as caching, SSL/TLS support, and access control. Overall, this project serves as a starting point for building more useful proxy server applications.

With further development and customization, this proxy server can be expanded to meet specific requirements and provide a more accurate solution for proxying and managing web traffic.