SIMPLE PROXY SERVER

21CSC302J /COMPUTER NETWORKS

MINI PROJECT

Submitted by

Sarathi V **(RA2211042020026)**

Aribalan I **(RA2211042020028)**

Shakthivel K **(RA2211042020060)**

Manyam V Avinash **(RA2211042020069)**

Under the guidance of

Dr. G. Gangadevi

(Assistant Professor, Department of Computer Science and Engineering)

V SEMESTER/ III YEAR

COMPUTER SCIENCE AND ENGINEERING

WITH SPECIALIZATION IN BUSINESS SYSTEMS

FACULTY OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
RAMAPURAM, CHENNAI

October 2024

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
(Deemed to be University U/S 3 of UGC Act, 1956)

BONAFIDE CERTIFICATE

Certified that this project report titled "**Simple Proxy Server**"is the bonafide work of **Sarathi V (RA2211042020026),Aribalan I (RA2211042020028), Shakthivel K (RA2211042020060),Manyam V Avinash (RA2211042020069)** who carried out the project work under my supervision. This project work confirms to **21CSC302J /Computer Networks** , V Semester, III year, 2024.

SIGNATURE

Dr. G. Gangadevi, M. Tech, Ph.D.,
Assistant Professor

Computer Science and Engineering, SRM Institute of Science and Technology, Ramapuram, Chennai.

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

RAMAPURAM, CHENNAI

DECLARATION

We hereby declare that the entire work contained in this mini project titled **"Simple Proxy Server"** has been carried out by **Sarathi V (RA2211042020026), Aribalan I (RA2211042020028), Shakthivel K (RA2211042020060), Manyam V Avinash (RA2211042020069)** at SRM Institute of Science and Technology, Ramapuram, Chennai, under the guidance of Dr. G. Gangadevi, Assistant professor, Department of Computer Science and Engineering.

Place: Chennai

Date:

Sarathi V
Aribalan I
Shakthivel K
Manyam V Avinash

ABSTRACT

This project, titled "Simple Proxy Server," explores the implementation and functionality of a basic proxy server using Python. A proxy server acts as an intermediary between a client and a server, facilitating indirect network connections by handling requests and forwarding them to the target server. This project focuses on the essential components of a proxy server, including connection handling, request forwarding, and response relay.

The primary aim of the project is to design a proxy server capable of managing HTTP requests from clients and forwarding them to external web servers while returning the corresponding responses. The implementation includes socket programming in Python, multithreading to handle multiple client connections, and basic request logging to track communication between clients and servers.

The project also discusses the challenges of designing a proxy server, such as connection handling, timeouts, and data parsing. Through the development of this proxy server, students gain hands-on experience with network protocols, understand the role of proxies in privacy and security, and appreciate the complexities involved in network communication.

This project offers valuable insights into computer networks by demonstrating the practical use of a proxy server in managing and securing network traffic, making it a crucial component of modern web communication and security practices.

TABLE OF CONTENTS

INTRODUCTION

The growth of the internet has brought an increased need for secure and efficient data transmission across networks. Proxy servers play a pivotal role in managing and optimizing these communications. A proxy server acts as an intermediary between a client requesting resources and a server providing those resources. By handling requests and forwarding them to the target server, proxy servers offer numerous benefits, such as improved security, load balancing, anonymity, and traffic control.

This project, "Simple Proxy Server," is designed to implement a basic proxy server using Python, which facilitates communication between clients and external servers. It enables users to access web resources indirectly, allowing the proxy server to act as a gateway. The project aims to provide students with hands-on experience in socket programming, network protocols, and multithreading concepts, while demonstrating how proxy servers can be used to manage, monitor, and secure network traffic.

The use of proxy servers is fundamental in various applications, including caching, content filtering, and access control. Additionally, proxy servers play a vital role in improving privacy by hiding the client's IP address, thus enhancing online anonymity. This project highlights these key aspects, offering an understanding of proxy server operations and their significance in modern-day network communication.

PROBLEM STATEMENT

In today's digital landscape, the need for enhanced network security, data privacy, and efficient communication across the internet has become increasingly critical. Proxy servers are essential tools that manage these needs by acting as intermediaries between clients and external servers. However, implementing a proxy server requires a solid understanding of networking concepts such as connection handling, HTTP protocols, and client-server communication, which can be challenging for beginners.

This project aims to address the problem of building a basic yet functional proxy server that can manage and relay HTTP requests and responses efficiently. The primary issues include how to handle multiple client connections concurrently, how to process and forward HTTP requests correctly, and how to ensure secure and reliable data transmission between the client and the external server. Additionally, the solution needs to provide transparency in its operations while maintaining flexibility to handle different types of web requests.

By developing a simple proxy server using Python, this project seeks to offer a learning platform for understanding socket programming, network protocols, and the role of proxy servers in modern networking. The solution will provide a practical approach to managing network traffic while demonstrating the value of proxy servers in improving privacy, security, and network performance.

OBJECTIVE

The main objective of the **"Simple Proxy Server" project is to design and implement a basic proxy server using Python to enable indirect communication between clients and external web servers. This project focuses on developing a proxy server capable of handling HTTP requests from clients, forwarding them to the appropriate web servers, and retrieving and delivering the corresponding responses. Additionally, it aims to enhance understanding of essential concepts such as socket programming, network protocols, and the practical functionality of proxy servers.

To optimize the performance of the proxy, multithreading is implemented, allowing the server to efficiently manage multiple client connections simultaneously. The project also explores how proxy servers contribute to network security, privacy, and performance optimization by acting as intermediaries, offering opportunities to learn about practical applications in areas like load balancing, content filtering, and access control. Furthermore, the project provides hands-on experience with monitoring and analyzing network traffic through request logging, helping troubleshoot network communication issues. Overall, this project serves as a valuable learning opportunity for understanding the core concepts and practical applications of proxy servers within computer networks.

SCOPE OF PROJECT

The "Simple Proxy Server" project focuses on the development of a basic proxy server using Python to handle HTTP requests from clients and forward them to external web servers. This project demonstrates essential functionalities such as receiving requests, forwarding them, and relaying responses back to the client. The project aims to provide practical insight into the operation of proxy servers and their role in managing network traffic, enhancing security, and optimizing performance. One key aspect is the implementation of multithreading, which enables the server to handle multiple client connections concurrently, ensuring scalability and efficiency. The proxy server will also log client requests, allowing for the monitoring and analysis of network communication. In addition, the project highlights how proxy servers can enhance privacy by masking client IP addresses and securing communication channels. While this project covers fundamental proxy server operations, it can be extended to include advanced features such as caching, SSL/TLS encryption, and content filtering. The project offers students a hands-on learning experience in socket programming, HTTP protocols, and multithreaded network application development, laying the groundwork for understanding more complex network systems and security protocols.

EXISTING SYSTEM

In the current landscape of network communication, proxy servers play an essential role in managing and optimizing data transmission between clients and servers. Existing proxy systems are sophisticated, offering features such as caching, content filtering, SSL/TLS encryption, load balancing, and secure access control. These systems serve various purposes, including enhancing privacy by masking user IP addresses, improving network performance through caching of frequently accessed data, and allowing administrators to control and monitor network traffic.

Commercial proxy servers, such as Squid Proxy, NGINX, and HAProxy, are widely used across industries to manage web traffic and ensure secure communication. These servers support large-scale, high-performance environments, providing robust load-balancing features, session management, and advanced security mechanisms. Moreover, existing systems are integrated with caching techniques that reduce bandwidth consumption and improve the speed of content delivery.

However, these systems can be complex, requiring significant expertise to configure and manage. They often come with numerous advanced features that may be overwhelming for those just beginning to learn about proxy servers and their role in network communication. Additionally, they are designed for large-scale environments and may not be suitable for small-scale or educational purposes where basic functionality is required.

This project aims to simplify the concept of proxy servers by implementing a basic version, offering a streamlined approach to handling HTTP requests, forwarding them to external servers, and returning the responses. The proposed system focuses on providing a learning platform for beginners, introducing key concepts like socket programming, multithreading, and basic network management without the complexity of commercial solutions.

SOFTWARE AND HARDWARE REQUIREMENTS

HARDWARE REQUIREMENTS

2 GB RAM (basic) or 4 GB RAM (for better performance)
Dual-core or Quad-core processor
Ethernet or Wi-Fi capability for network connectivity
4 GB RAM for improved performance or 8 GB RAM for higher workloads
Multi-core processor for optimized multitasking
Stable network connectivity through Ethernet for consistent communication

SOFTWARE REQUIREMENTS

Operating System:
Windows 10 or higher, Linux, or macOS.
Programming Language:
Python 3.x (Recommended version: Python 3.7 or higher).
Python Libraries:
socket – for handling network connections and communications.
threading – for managing multiple client connections concurrently.
Integrated Development Environment (IDE):
Visual Studio Code, PyCharm, or any text editor with Python support.
Web Browser:
Any modern browser (Google Chrome, Microsoft Edge, Firefox) to test the proxy server.

PROPOSED SYSTEM

The proposed system is a Simple Proxy Server designed to facilitate indirect communication between clients and external web servers. This proxy server aims to intercept and forward HTTP requests from clients, retrieve the necessary responses from the web servers, and return those responses to the clients. Unlike complex commercial proxies, this system provides a basic implementation using Python, focusing on functionality, efficiency, and educational value for those new to networking concepts.

The proposed proxy server is built with the following core features:

Request Handling and Forwarding:

The system will accept HTTP requests from multiple clients, forward these requests to the relevant web servers, and retrieve the respective responses. It acts as an intermediary, ensuring seamless communication between the client and the external server.

Multithreading:

To handle multiple client connections simultaneously, the proposed system will incorporate multithreading. This will allow the proxy to manage and serve multiple clients at once without compromising performance.

Connection Transparency:

The system will forward requests and responses transparently, allowing users to interact with external web services without any noticeable interference from the proxy.

Request Logging:

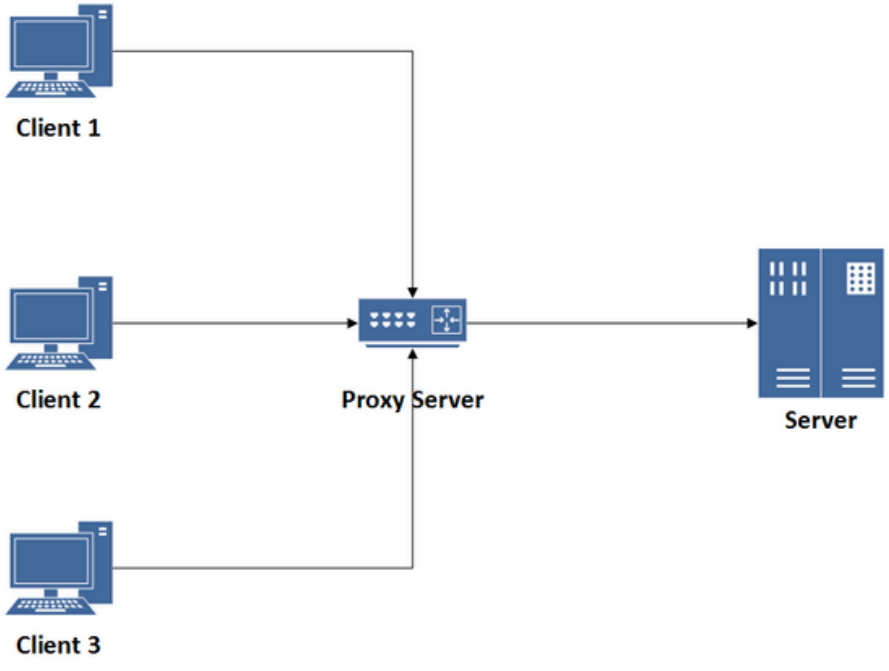
The proxy server will log each client's request, providing a record of the traffic passing through the server. This will help in analyzing network performance and troubleshooting communication issues.

Educational Focus:

This system provides a simplified model of a proxy server, emphasizing core networking concepts such as socket programming, HTTP protocol handling, and client-server communication. Its primary goal is to serve as a learning tool for students and beginners in the field of computer networks.

By implementing this simple proxy server, the proposed system offers a practical, hands-on understanding of how proxy servers work, their role in network communication, and their applications in enhancing privacy, security, and performance. The project can be further extended to include advanced features like caching, content filtering, and SSL encryption in future iterations.

Architecture Diagram



MODULES DESCRIPTION

The Simple Proxy Server project consists of several interconnected modules that work together to achieve its core functionality. These modules handle client requests, forward them to external servers, receive responses, and return the data to the client while maintaining logs of the interactions. The key modules of the project are as follows:

Client Connection Module:

This module is responsible for accepting incoming client requests. It uses Python's socket library to establish a TCP connection between the client and the proxy server. Once a connection is established, the module waits for the client's HTTP request and passes it on for further processing. It ensures that the proxy can handle multiple clients by creating a separate thread for each connection.

Request Processing Module:

After receiving the client's request, this module processes the HTTP request header, parses the requested URL, and forwards the request to the external web server. This step includes managing the connection to the web server, ensuring that the proper request format is maintained, and handling both GET and POST request types.

Forwarding and Response Module:

Once the request is sent to the external server, this module is responsible for receiving the server's response. It retrieves the content (such as an HTML page or other media) and forwards it back to the client. This module also handles the buffering of data to ensure smooth communication between the proxy and both the client and the server.

Multithreading Module:

The multithreading module ensures that the proxy server can handle multiple client requests simultaneously. Using Python's threading library, this module creates a new thread for each client connection, allowing the server to serve several clients concurrently. It manages thread lifecycles and ensures that each request is handled independently without blocking other connections.

Logging Module:

This module logs all incoming client requests and outgoing server responses. It records essential information such as client IP address, requested URL, and response status. This log can be used for debugging, monitoring, or analyzing network traffic patterns.

Error Handling Module:

The error handling module ensures that the proxy server can manage unexpected issues like broken client connections, server timeouts, or incorrect request formats. It provides error messages to the client when a request cannot be fulfilled and ensures that the server continues to function smoothly despite errors.

Each of these modules works in conjunction to create a fully functional proxy server, enabling users to send and receive data through an intermediary server. The modular design allows for future improvements and extensions, such as adding SSL support, caching, or advanced security features.

CODE

Python -

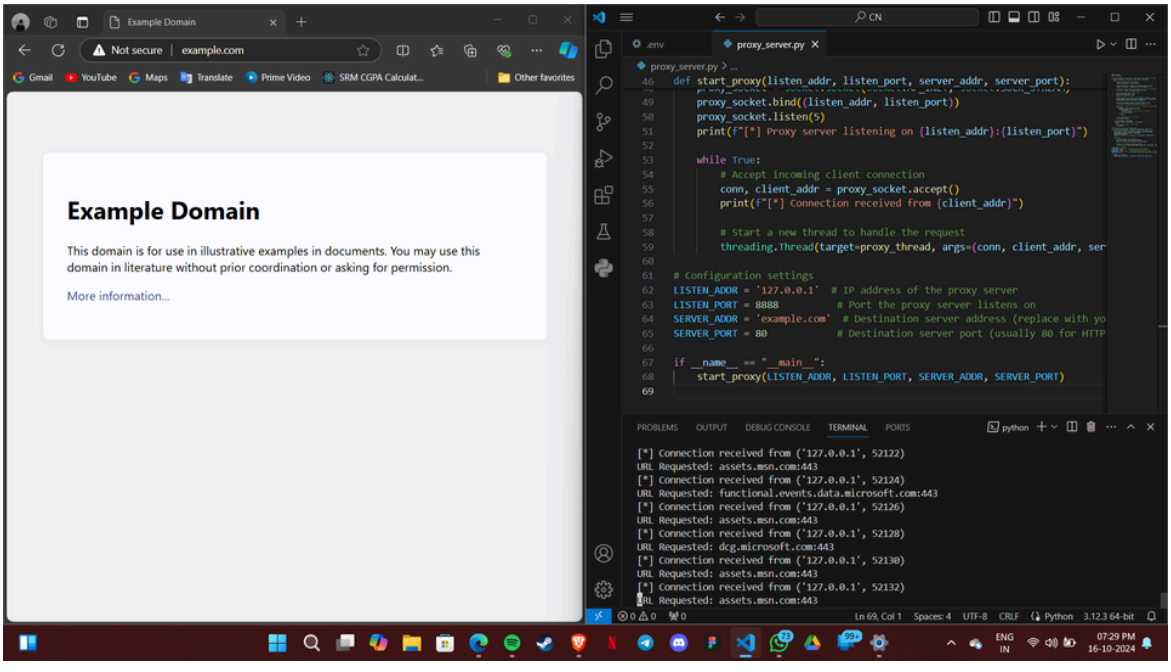
```
import socket
import threading

# Function to handle requests from clients and forward them to the server
def proxy_thread(conn, client_addr, server_addr, server_port):
  try:
    # Receive request from the browser
    request_from_browser = conn.recv(4096)

    # Decode the bytes received from the browser into a string
    request_from_browser = request_from_browser.decode('utf-8')

    # Extract the first line of the request (this contains the method and URL)
    first_line = request_from_browser.split('\n')[0]

    # Extract the URL from the first line
    url = first_line.split(' ')[1]
    print(f"URL Requested: {url}")
```

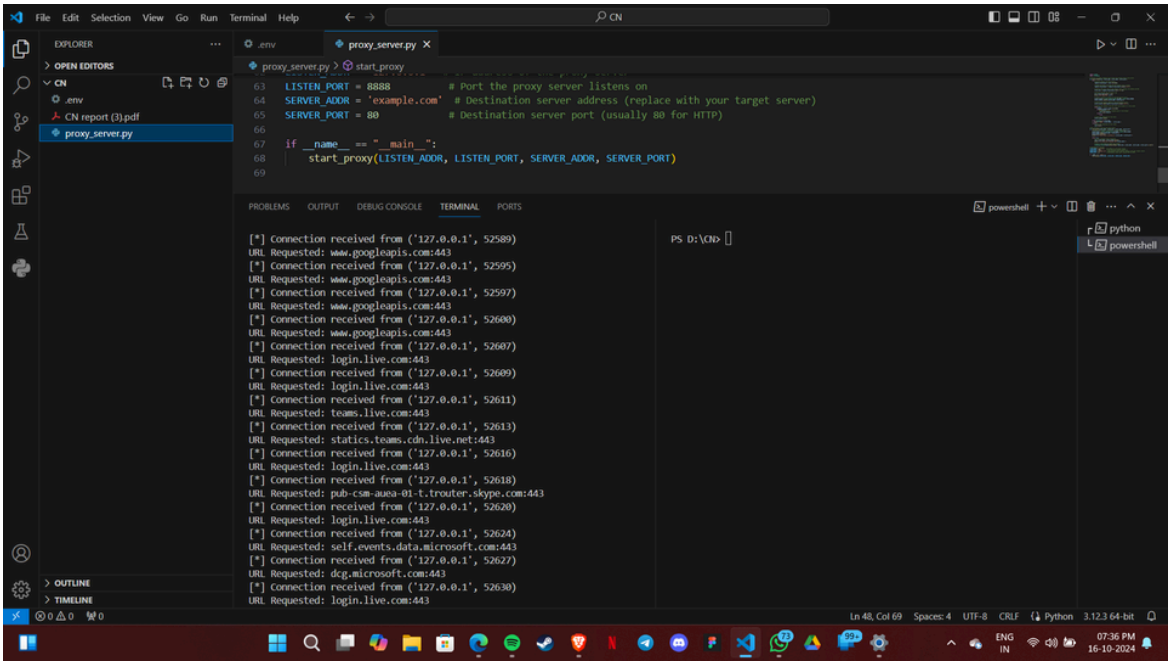
```
# Create a socket to connect to the destination server
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.connect((server_addr, server_port))

    # Forward the browser request to the destination server
    server_socket.send(request_from_browser.encode('utf-8'))

    # Receive the response from the server and forward it to the client
    while True:
      response = server_socket.recv(4096)
      if len(response) > 0:
        # Send the server's response back to the browser
        conn.send(response)
      else:
        break

    # Close the connection to the server
    server_socket.close()
```

```
except Exception as error_msg:
print(f'ERROR: {client_addr} - {error_msg}')
finally:
# Close the connection to the client
conn.close()

# Main function to set up the proxy server
def start_proxy(listen_addr, listen_port, server_addr, server_port):
# Create a TCP socket to listen for incoming connections
proxy_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
proxy_socket.bind((listen_addr, listen_port))
proxy_socket.listen(5)
print(f"[*] Proxy server listening on {listen_addr}:{listen_port}")

while True:
# Accept incoming client connection
conn, client_addr = proxy_socket.accept()
print(f"[*] Connection received from {client_addr}")
```

```
# Start a new thread to handle the request
        threading.Thread(target=proxy_thread, args=(conn, client_addr, server_addr, server_port)).start()

# Configuration settings
LISTEN_ADDR = '127.0.0.1' # IP address of the proxy server
LISTEN_PORT = 8888 # Port the proxy server listens on
SERVER_ADDR = 'example.com' # Destination server address (replace with your target server)
SERVER_PORT = 80 # Destination server port (usually 80 for HTTP)

if __name__ == "__main__":
 start_proxy(LISTEN_ADDR, LISTEN_PORT, SERVER_ADDR, SERVER_PORT)
```

OUTPUT


Example Domain
Not secure | example.com
Gmail YouTube Maps Translate Prime Video SRM CGPA Calculat... Other favorites
Example Domain
This domain is for use in illustrative examples in documents. You may use this domain in literature without prior coordination or asking for permission.
More information...
proxy_server.py
def start_proxy(listen_addr, listen_port, server_addr, server_port):
proxy_socket.bind((listen_addr, listen_port))
proxy_socket.listen(5)
print(f"[*] Proxy server listening on {listen_addr}:{listen_port}")
while True:
# Accept incoming client connection
conn, client_addr = proxy_socket.accept()
print(f"[*] Connection received from {client_addr}")
# Start a new thread to handle the request
threading.Thread(target=proxy_thread, args=(conn, client_addr, ser
# Configuration settings
LISTEN_ADDR = '127.0.0.1' # IP address of the proxy server
LISTEN_PORT = 8888 # Port the proxy server listens on
SERVER_ADDR = 'example.com' # Destination server address (replace with yo
SERVER_PORT = 80 # Destination server port (usually 80 for HTTP
if __name__ == "__main__":
start_proxy(LISTEN_ADDR, LISTEN_PORT, SERVER_ADDR, SERVER_PORT)
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
[*] Connection received from ('127.0.0.1', 52122)
URL Requested: assets.msn.com:443
[*] Connection received from ('127.0.0.1', 52124)
URL Requested: functional.events.data.microsoft.com:443
[*] Connection received from ('127.0.0.1', 52126)
URL Requested: assets.msn.com:443
[*] Connection received from ('127.0.0.1', 52128)
URL Requested: dcg.microsoft.com:443
[*] Connection received from ('127.0.0.1', 52130)
URL Requested: assets.msn.com:443
[*] Connection received from ('127.0.0.1', 52132)
URL Requested: assets.msn.com:443
Ln 69, Col 1 Spaces: 4 UTF-8 CRLF Python 3.12.3 64-bit
07:29 PM
16-10-2024


File Edit Selection View Go Run Terminal Help
CN
EXPLORER
OPEN EDITORS
CN
.env
CN report (3).pdf
proxy_server.py
.env
proxy_server.py
proxy_server.py > start_proxy
63 LISTEN_PORT = 8888 # Port the proxy server listens on
64 SERVER_ADDR = 'example.com' # Destination server address (replace with your target server)
65 SERVER_PORT = 80 # Destination server port (usually 80 for HTTP)
66
67 if __name__ == "__main__":
68 start_proxy(LISTEN_ADDR, LISTEN_PORT, SERVER_ADDR, SERVER_PORT)
69
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
powershell
python
powershell
[*] Connection received from ('127.0.0.1', 52589)
URL Requested: www.googleapis.com:443
[*] Connection received from ('127.0.0.1', 52595)
URL Requested: www.googleapis.com:443
[*] Connection received from ('127.0.0.1', 52597)
URL Requested: www.googleapis.com:443
[*] Connection received from ('127.0.0.1', 52600)
URL Requested: www.googleapis.com:443
[*] Connection received from ('127.0.0.1', 52607)
URL Requested: login.live.com:443
[*] Connection received from ('127.0.0.1', 52609)
URL Requested: login.live.com:443
[*] Connection received from ('127.0.0.1', 52611)
URL Requested: teams.live.com:443
[*] Connection received from ('127.0.0.1', 52613)
URL Requested: statics.teams.cdn.live.net:443
[*] Connection received from ('127.0.0.1', 52616)
URL Requested: login.live.com:443
[*] Connection received from ('127.0.0.1', 52618)
URL Requested: pub-csm-auea-01-t.trouter.skype.com:443
[*] Connection received from ('127.0.0.1', 52620)
URL Requested: login.live.com:443
[*] Connection received from ('127.0.0.1', 52624)
URL Requested: self.events.data.microsoft.com:443
[*] Connection received from ('127.0.0.1', 52627)
URL Requested: dcg.microsoft.com:443
[*] Connection received from ('127.0.0.1', 52630)
URL Requested: login.live.com:443
PS D:\CN>
OUTLINE
TIMELINE
Ln 48, Col 69 Spaces: 4 UTF-8 CRLF Python 3.12.3 64-bit
ENG
IN
07:36 PM
16-10-2024

CONCLUSION

The development of the Simple Proxy Server demonstrates a clear understanding of how network communication is handled between clients and external web servers. By implementing a basic proxy using Python, this project provides a practical introduction to key networking concepts such as socket programming, multithreading, and HTTP protocol handling.

The project serves as an excellent educational tool, bridging the gap between theoretical knowledge and hands-on application. It allows beginners to gain insight into how a proxy server functions, including the forwarding of client requests and server responses, managing concurrent connections, and logging network activity. Despite its simplicity, the project highlights the importance of intermediary servers in enhancing privacy, security, and performance in network communication.

Furthermore, this project lays a solid foundation for future enhancements. The basic structure can be extended with features such as caching, content filtering, or support for HTTPS connections, providing further learning opportunities. Ultimately, the Simple Proxy Server not only fulfills its initial objectives but also opens doors to deeper exploration of network infrastructure and cybersecurity concepts.

FUTURE ENHANCEMENTS

The Simple Proxy Server project offers significant potential for future enhancements that can extend its functionality and make it more robust. One of the primary areas for improvement is the implementation of a caching mechanism, which would allow the server to store frequently accessed content, reducing latency and bandwidth consumption. Another enhancement is to incorporate HTTPS support, which would enable the proxy to handle encrypted traffic, essential for secure communications in modern web environments.

Additional features like content filtering can be introduced to control access to certain types of content, making the proxy suitable for environments such as schools or offices. Furthermore, incorporating load balancing would allow the proxy to distribute client requests across multiple servers, improving overall performance and ensuring higher availability.

Other useful enhancements include detailed logging and analytics to track and monitor server performance and user behavior, as well as authentication and access control mechanisms to ensure only authorized users can access resources through the proxy. Adding support for additional protocols, such as FTP or WebSocket, would further expand the system's capabilities. Finally, implementing IP blocking and throttling features would strengthen security by preventing abuse or denial-of-service (DoS) attacks. These future enhancements would elevate the project from a simple learning tool to a more sophisticated, feature-rich proxy server capable of meeting various modern networking needs.

REFERENCES

- Stevens, W. R., & Fenner, B. (2004). UNIX Network Programming: Networking APIs: Sockets and XTI. Prentice Hall. This book provides an in-depth understanding of socket programming, which is a core concept used in the development of the proxy server.
- Tanenbaum, A. S., & Wetherall, D. J. (2011). Computer Networks (5th ed.). Pearson Education. This comprehensive textbook covers essential networking concepts and protocols, which form the foundation of the project.
- Python Software Foundation. (n.d.). Python Documentation: socket — Low-level networking interface. Retrieved from https://docs.python.org/3/library/socket.html. This documentation provides detailed explanations of Python's socket library, used for handling network connections in the project.
- HTTP/1.1: Hypertext Transfer Protocol. (1999). Retrieved from https://www.w3.org/Protocols/rfc2616/rfc2616.html. The official specification of the HTTP/1.1 protocol, used for client-server communication in the proxy server.
- Gupta, S., & Shroff, R. (2016). Introduction to Networking and Network Design. Springer. This book provides an overview of networking architectures and designs, relevant to the proxy server's network routing functionalities.