Project Report: Custom Network Proxy Server

Abstract

This project involves the design and implementation of a concurrent, multithreaded network proxy server using the C programming language. The server acts as an intermediary between client applications and the internet, facilitating HTTP forwarding, HTTPS tunneling via the CONNECT method, and rule-based traffic filtering. The system is built upon the POSIX socket API and utilizes a thread-per-connection concurrency model to ensure high performance and reliability under load.

Introduction

A proxy server is a critical component in modern network architecture, providing privacy, security, and traffic control. The objective of this project was to build a forward proxy from scratch that demonstrates fundamental systems programming concepts, including low-level socket manipulation, protocol parsing, and multithreading.

Objectives

- **Reliability:** Establish robust TCP connections between clients and upstream servers.
- **Concurrency:** Handle multiple simultaneous client requests without blocking.
- **Control:** Implement a blacklist mechanism to block access to specific domains.
- **Transparency:** Support encrypted HTTPS traffic using blind tunneling.

Functional Requirements

1. **Request Handling:** The server must accept incoming TCP connections on a configurable port (default: 8888).
2. **Parsing:** It must parse HTTP request lines to extract the Method, Host, and Port.
3. **Filtering:** Before forwarding, the target host must be checked against a blocked domain list.
4. **Forwarding:**

- **HTTP:** Forward the raw request and stream the response back.
- **HTTPS:** Establish a tunnel upon receiving a CONNECT request.

5. **Logging:** All transaction details (IP, URL, Status) must be logged to a file.

Non-Functional Requirements

- **Performance:** Low latency in request processing.
- **Scalability:** Ability to handle concurrent users via threading.
- **Modularity:** Codebase separated into logical modules (Network, Logic, Parsing).

Implementation Details

- **Technology Stack**
  - **Language:** C (Standard C99)
  - **Libraries:** pthread (Concurrency), sys/socket (Networking), netdb (DNS).
  - **Platform:** Linux/WSL (POSIX Compliant).
- **Key Modules**
  - **proxy.c:** The core server loop. It sets up the listener socket and spawns a new thread (pthread_create) for every accepted connection.
  - **proxy_parse.c:** A custom string parser that handles HTTP headers. It supports extracting hostnames from both absolute URLs (e.g., GET http://site.com) and relative URLs with Host headers.
  - **proxy_filter.c:** file I/O module that loads config/blocked_domains.txt and performs string matching to enforce access control.
- **HTTPS Tunneling Strategy**

  The project implements the CONNECT method for HTTPS.

  1. The proxy detects the CONNECT verb.
  2. It establishes a connection to the target server (port 443).
  3. It responds with 200 Connection Established.
  4. It enters a select() loop to blindly forward encrypted bytes between the client and server, maintaining the integrity of the SSL/TLS session.

Testing and Results

The system was tested using curl and automated scripts.

Test Case	Description	Result	Status
TC-01	Basic HTTP GET (example.com)	HTML Content received (200 OK)	**PASS**
TC-02	Blocked Domain (badsite.com)	403 Forbidden Error received	**PASS**
TC-03	HTTPS Tunneling (https://www.google.com/search?q=google.com)	Secure connection established	**PASS**
TC-04	Concurrent Load	Multiple clients served simultaneously	**PASS**

6. Conclusion

The Custom Network Proxy Server successfully meets all primary objectives outlined in the problem statement. It provides a reliable, concurrent mechanism for forwarding web traffic while enforcing access control policies. The addition of HTTPS tunneling makes it a viable tool for modern, encrypted web browsing.