Project Report

On

BASIC WEB SERVER DEVELOPER

Submitted in partial fulfilment of the requirements for the award of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE & ENGINEERING

(Artificial Intelligence & Machine Learning)

by

Ms. AIMAN RAZIA (22WH1A6614)

Ms. LAKSHMI INDU KOSURI (22WH1A6619)

Ms. JAHNVI KAKKAR (22WH1A6640)

Ms. MUSKAAN(22WH1A6650)

Under the esteemed guidance of

Ms. P Anusha

Assistant Professor, CSE(AI&ML)

BVRIT HYDERABAD College of Engineering for Women

(UGC Autonomous Institution | Approved by AICTE | Affiliated to JNTUH)

(NAAC Accredited - A Grade | NBA Accredited B.Tech. (EEE, ECE, CSE and IT)

Bachupally, Hyderabad – 500090

2024-25

ABSTRACT

The **Basic Web Server Developer** project implements a simple, lightweight web server using C. It handles HTTP requests and delivers HTML responses, making it suitable for learning and experimenting with server-side programming. The server supports essential features like socket creation, binding, and client connection handling. Designed for efficiency, it serves a basic "Hello, World!" webpage and can be extended for more complex applications. This project provides a foundation for understanding web server functionality and networking concepts.

PROBLEM STATEMENT

In the modern digital era, web servers play a crucial role in delivering content over the internet. However, understanding the underlying mechanisms of web servers can be challenging for learners and developers new to networking and server-side programming. Most existing server solutions are complex, resource-intensive, or require advanced knowledge of frameworks, making them unsuitable for educational purposes or lightweight applications.

This project aims to address these challenges by providing a **Basic Web Server Developer** implementation using C. It enables users to:

- Understand fundamental web server concepts, such as socket creation, binding, and HTTP response handling.
- Develop a lightweight server capable of delivering basic content over HTTP.
- Gain hands-on experience with low-level networking operations without the overhead of complex frameworks.

The goal is to offer a straightforward, easily extensible, and resource-efficient solution for developers to learn and experiment with web server functionality.

FUNCTIONAL REQUIREMENTS

- **Network Statistics**:

The application must fetch real-time network statistics, including the number of bytes sent (upload) and received (download), using the psutil library. These statistics should be precise.

- **Human-readable Outputs**:

To enhance usability, the application must convert raw network statistics, such as the number of bytes transferred, into more human-readable formats, such as kilobytes (KB), megabytes (MB), gigabytes (GB), or terabytes (TB).

- **Real-time Updates**:

The tool should update network statistics in real time to ensure users always have access to the latest information. In the GUI mode, updates should occur at an interval of 1.5 seconds to balance performance and usability.

- **Flexible User Interface**:

The application must provide a user-friendly and intuitive interface in both GUI and console modes. The GUI should have a clear layout with organized labels for displaying upload speeds, download speeds, and total network usage.

- **Error Handling**:

The application must handle edge cases, such as counter wraparounds, gracefully. For example, if the system's network counters reset due to hardware or software limitations, the tool should detect and account for this to avoid displaying incorrect statistics.

NON-FUNCTIONAL REQUIREMENTS

1. **Performance**:

The application must be lightweight and efficient, minimizing CPU and memory usage even when running continuously. Real-time updates should occur without introducing significant delays, ensuring a smooth user experience.

2. **Usability**:

The tool should be simple and intuitive for users of all technical levels. The GUI version must have a clean and organized layout, displaying all key statistics clearly and without unnecessary clutter.

3. **Reliability**:

The application must ensure that the displayed network statistics are accurate and up-to-date in both GUI and console modes. It should handle unexpected situations, such as network counter wraparounds, network disconnections, or hardware changes, gracefully and without crashing.

4. **Portability**:

The tool should be compatible with multiple platforms, including Windows, Linux, and macOS, requiring minimal or no changes to the source code. It must use widely available Python libraries, such as psutil and tkinter, ensuring easy installation and setup across different operating systems.

5. **Scalability**:

While the application is designed primarily for individual users, it should be adaptable for larger-scale deployments. For example, it could be extended to monitor multiple network interfaces or support multi-user environments on servers.

6. **Maintainability**:

The codebase should be modular and well-documented, with clear comments and consistent naming conventions. Each functionality, such as network data fetching, UI updates, and error handling, should be implemented as separate modules to facilitate debugging, future updates, and scalability.

7. **Security**:

Although the application does not directly interact with sensitive data, it must ensure that no system or network data is leaked or mishandled during operation. Additionally, the application should operate with minimum permissions, accessing only the necessary system resources to avoid potential vulnerabilities.

8. Extensibility:

The design of the application should allow for future enhancements. For instance, developers should be able to add new features, such as logging historical data or monitoring specific applications, without requiring significant modifications to the existing structure.

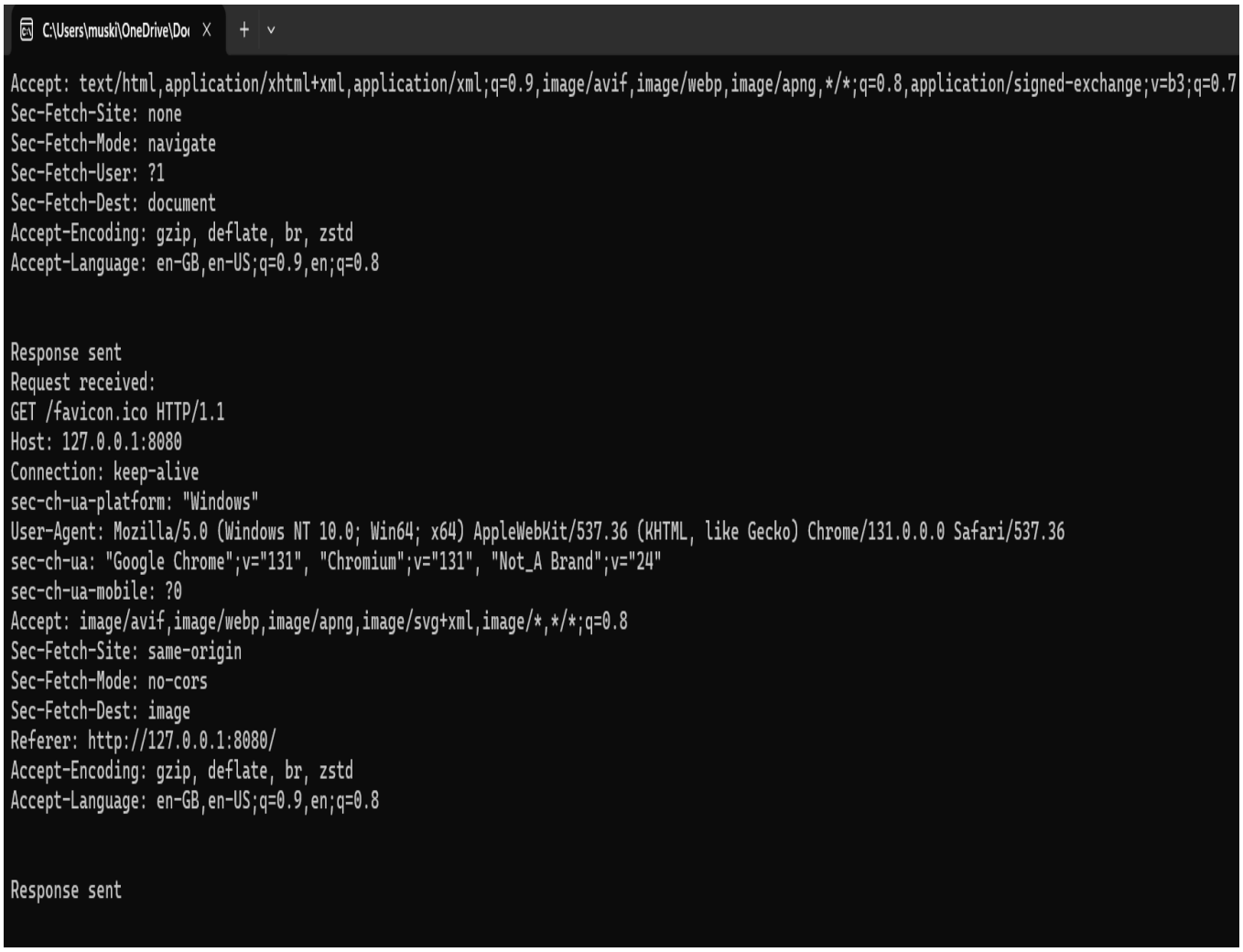
SOURCE CODE

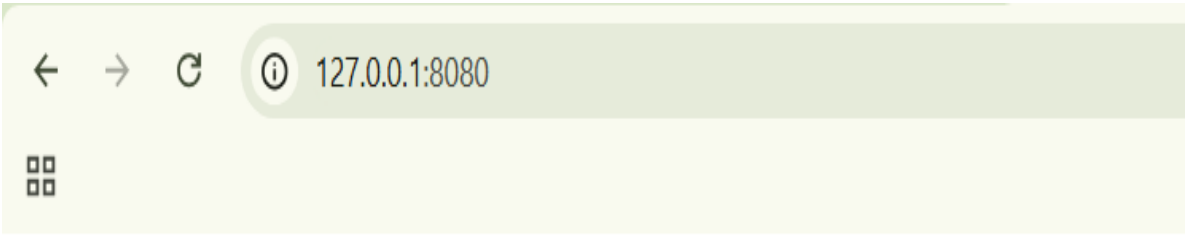
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>

#define PORT 8080
#define BUFFER_SIZE 1024

int main() {
    int server_fd, new_socket;
    struct sockaddr_in address;
    int addrlen = sizeof(address);
    char buffer[BUFFER_SIZE] = {0};
    const char *response =
        "HTTP/1.1 200 OK\n"
        "Content-Type: text/html\n\n"
        "<html><body><h1>Hello, World!</h1></body></html>";

    // Step 1: Create socket
    if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
        perror("Socket failed");
        exit(EXIT_FAILURE);
    }
```

```
    // Step 2: Bind the socket to the address and port

    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY;
    address.sin_port = htons(PORT);

    if (bind(server_fd, (struct sockaddr *)&address, sizeof(address)) < 0) {
        perror("Bind failed");
        close(server_fd);
        exit(EXIT_FAILURE);
    }

    // Step 3: Listen for incoming connections
    if (listen(server_fd, 3) < 0) {
        perror("Listen failed");
        close(server_fd);
        exit(EXIT_FAILURE);
    }
    printf("Server is listening on port %d\n", PORT);

    // Step 4: Accept and handle incoming connections
    while (1) {
        if ((new_socket = accept(server_fd, (struct sockaddr *)&address, (socklen_t *)&addrlen))
< 0) {
            perror("Accept failed");
```

```
            continue;
        }

        // Read the request (not used here, but for HTTP parsing if needed)
        read(new_socket, buffer, BUFFER_SIZE);
        printf("Request received:\n%s\n", buffer);

        // Send response
        send(new_socket, response, strlen(response), 0);
        printf("Response sent\n");

        // Close connection
        close(new_socket);
    }

    // Cleanup
    close(server_fd);
    return 0;
}
```

OUTPUT


C:\Users\muski\OneDrive\Do
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: en-GB,en-US;q=0.9,en;q=0.8
Response sent
Request received:
GET /favicon.ico HTTP/1.1
Host: 127.0.0.1:8080
Connection: keep-alive
sec-ch-ua-platform: "Windows"
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/131.0.0.0 Safari/537.36
sec-ch-ua: "Google Chrome";v="131", "Chromium";v="131", "Not_A Brand";v="24"
sec-ch-ua-mobile: ?0
Accept: image/avif,image/webp,image/apng,image/svg+xml,image/*,*/*;q=0.8
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: no-cors
Sec-Fetch-Dest: image
Referer: http://127.0.0.1:8080/
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: en-GB,en-US;q=0.9,en;q=0.8
Response sent



Hello, World!