РЕФЕРАТ

Расчетно-пояснительная записка содержит 19 с., 5 рис..

В рамках данной курсовой работы была разработка статик — сервера на языке Си без использования сторонних библиотек, построенного при помощи паттерна prefork и использующего системный вызов select.

Ключевые слова: статик-сервер, prefork, select, http.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель работы — создать статический веб-сервер, способный обрабатывать GET и HEAD запросы. Статический веб-сервер — это программное решение, предназначенное для обслуживания и передачи статического контента через Интернет. Он обеспечивает доступ к различным ресурсам, таким как HTML-страницы, изображения, таблицы стилей и другие, которые не генерируются динамически на сервере.

С развитием веб-технологий статические веб-серверы стали играть важную роль в обеспечении быстрой и надежной обработки статических ресурсов. Они востребованы благодаря своей простоте и эффективности в предоставлении пользователям быстрого доступа к статическим компонентам веб-приложений.

Разработка статических веб-серверов включает в себя задачи по обеспечению безопасности, таких как предотвращение выхода за пределы определенной директории. Потенциальные угрозы могут привести к утечке конфиденциальных данных и другим негативным последствиям для сервера. В данной работе будет уделено внимание методам обеспечения безопасности статического веб-сервера, включая контроль доступа и предотвращение уязвимостей, связанных с управлением файловой системой сервера.

Для достижения поставленной цели, предполагается выполнение следующих задач:

- провести формализацию задачи и определить небоходимый функционал;
- исследовать предметную область веб-серверов;
- спроектировать приложение;
- реализовать приложение;
- протестировать приложение на предмет корректности.

1 Аналитический раздел

1.1 Требования к серверу

- поддержка запросов GET и HEAD (поддержка статусов 200, 403, 404);
- ответ на неподдерживаемые запросы статусом 405;
- выставление content type в зависимости от типа файла (поддержка .html, .css, .js, .png, .jpg, .jpeg, .swf, .gif);
- корректная передача файлов размером в 100мб;
- сервер по умолчанию должен возвращать html-страницу на выбранную тему с css-стилем;
- учесть минимальные требования к безопасности статик-серверов (предусмотреть ошибку в случае если адрес будет выходить за root директорию сервера);
- реализовать логгер;
- использовать язык Си, сторонние библиотеки запрещены;
- реализовать архитектуру prefork + select();
- статик сервер должен работать стабильно.

1.2 Протокол HTTP

Изучение современных веб-технологий требует глубокого понимания протокола HTTP (Hypertext Transfer Protocol). Этот стандарт является ключевым для взаимодействия между веб-клиентами и серверами, обеспечивая передачу данных в формате гипертекста. HTTP сыграл значительную роль в развитии интернета, служа основой для передачи ресурсов, таких как HTML-страницы, изображения и стили.

Протокол HTTP основан на принципе запроса-ответа, где клиент отправляет запрос на сервер для получения определенного ресурса, и сервер отвечает соответствующим ответом. Этот обмен сообщениями основан на простых принципах и стандартах, обеспечивая эффективное взаимодействие между веб-клиентами и серверами.

Существует несколько версий протокола HTTP, каждая из которых вносит улучшения и новшества. В данном контексте выбран HTTP 1.1 из-за его возможности многократного использования соединений (connection reuse). Эта характеристика позволяет снизить задержки при обмене данными, поскольку соединение может быть переиспользовано для последующих запросов, экономя трафик и улучшая производительность [1].

1.3 Сокеты UNIX

Сетевые сокеты представляют собой универсальный механизм для обмена данными между процессами на одной машине или по сети. В контексте создания веб-сервера, использование сетевых сокетов становится ключевым элементом, обеспечивая взаимодействие между сервером и клиентами.

Дуплексная связь, обеспечиваемая сетевыми сокетами, позволяет полнодуплексный обмен данными между сервером и клиентом. Это важный аспект в контексте веб-сервера, поскольку позволяет одновременно обрабатывать входящие и исходящие запросы, что способствует повышению производительности и отзывчивости сервера.

Файловые сокеты, также известные как доменные сокеты UNIX, играют важную роль в локальной сетевой коммуникации. Они предоставляют способ для процессов обмениваться данными напрямую, обеспечивая эффективную и безопасную передачу информации между различными частями приложения.

Выбор сетевых сокетов для реализации данного веб-сервера был обоснован их универсальностью и способностью обработки запросов как через сеть, так и локально. Сетевые сокеты обеспечивают надежное взаимодействие между сервером и клиентами, что является важным аспектом для корректной работы веб-сервера [2].

Вывод

На основании анализа были определены требования к статическому веб-серверу. Решение использовать протокол HTTP 1.1 было обосновано его возможностью эффективного переиспользования соединений. Также было обосновано предпочтение сетевым сокетам UNIX перед файловыми.

2 Конструкторский раздел

2.1 HTTP-запросы

Запрос в протоколе HTTP представляет собой структурированный набор строк. Начиная с первой строки, содержащей информацию о методе, URL и версии HTTP, последующие строки содержат заголовки запроса, которые определяют дополнительные параметры запроса. В некоторых случаях запрос может также содержать тело, которое используется для передачи дополнительных данных для более сложных веб-запросов.

Листинг 1 – Пример HTTP-запроса через Postman

```
GET /img/prison.jpeg HTTP/1.1
User-Agent: PostmanRuntime/7.35.0
Accept: */*
Postman-Token: 7af874dc-5a4a-4678-a088-65bc9ca2b09e
Host: localhost:5600
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
```

2.2 HTTP-ответы

Ответ в протоколе HTTP содержит информацию о версии HTTP, числовом представлении статуса, текстовом описании статуса, заголовках и теле ответа. Ответ может включать как текстовую, так и бинарную информацию. В случае передачи изображений или других бинарных файлов, которые требуется разбить на несколько буферов для эффективной передачи, особенно при больших размерах, данная операция выполняется с целью оптимизации передачи данных клиенту.

2.3 Асинхронность

Предлагается предоставить опцию использования серверной реализации, основанной на механизме prefork в сочетании с select для обработки входящих подключений. Этот подход позволяет создать множество процессов-обработчиков заранее (prefork) и использовать select для мониторинга активности сокетов и эффективной обработки ввода-вывода.

2.4 Безопасность

Большое внимание уделяется обеспечению безопасности статического веб-сервера. Основной угрозой является возможность несанкционированного доступа к файлам, расположенным за пределами директории, доступной для сервера. Для защиты от доступа за пределами статической директории предлагается использовать функцию realpath, которая помогает получить абсолютный путь к файлу или директории. Это позволяет проверить, что запрашиваемый

путь принадлежит к пределам статической директории сервера и предотвратить доступ к файлам за ее пределами.

Вывод

В данном разделе были сформулированы требования к безопасности, асинхронности, формату запросов и ответов.

3 Технологический раздел

Этот раздел охватит технические детали реализации приложения и продемонстрирует работу программы.

3.1 Средства реализации

Для написания программы был выбран язык программирования Си. Си является низкоуровневым языком, обеспечивающим полный контроль над сокетами. Большую часть функционала пришлось реализовать вручную. Для разработки программы использовалась среда Visual Studio Code [3].

3.2 Листинги кода

На листинге 2 представлена реализация алгоритма prefork + select.

Листинг 2 – Механизм prefork + select

```
void start_server(void) {
    int server_socket;
    struct sockaddr_in server_addr;

    if ((server_socket = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }

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

    if (bind(server_socket, (struct sockaddr*)&server_addr, sizeof(server_addr)) == -1) {
        perror("Bind failed");
        exit(EXIT_FAILURE);
    }

    if (listen(server_socket, SOMAXCONN) == -1) {
        perror("Listen failed");
        exit(EXIT_FAILURE);
    }

    log_message(LOG_INFO, "Server listening on port %d...\n", PORT);

    for (int i = 0; i < NUM_CHILDREN; ++i) {
        pid_t pid = fork();

        if (pid == 0) {
            int client_sockets[MAX_CLIENTS] = { 0 };
            int max_fd, activity, addr_len, client_socket;
            struct sockaddr_in client_addr;
            fd_set readfds;

            while (1) {
                FD_ZERO(&readfds);
                FD_SET(server_socket, &readfds);

                max_fd = server_socket;

                for (int j = 0; j < MAX_CLIENTS; ++j) {
                    if (client_sockets[j] > 0) {
                        FD_SET(client_sockets[j], &readfds);
                        if (client_sockets[j] > max_fd) {
                            max_fd = client_sockets[j];
                        }
                    }
                }

                activity = select(max_fd + 1, &readfds, NULL, NULL, NULL);
                if ((activity < 0) && (errno != EINTR)) {
                    perror("Select error");
                    exit(EXIT_FAILURE);
                }

                if (FD_ISSET(server_socket, &readfds)) {
                    addr_len = sizeof(client_addr);
                    if ((client_socket = accept(server_socket, (struct sockaddr*)&client_addr, (
socklen_t*)&addr_len)) == -1) {
```

```
                        perror("Accept failed");
                        continue;
                    }

                    printf("New connection, socket fd is %d, ip is : %s, port : %d\n",
    client_socket,
                        inet_ntoa(client_addr.sin_addr), ntohs(client_addr.sin_port));

                    for (int j = 0; j < MAX_CLIENTS; ++j) {
                        if (client_sockets[j] == 0) {
                            client_sockets[j] = client_socket;
                            break;
                        }
                    }
                }

                for (int j = 0; j < MAX_CLIENTS; ++j) {
                    if (client_sockets[j] > 0 && FD_ISSET(client_sockets[j], &readfds)) {
                        handle_client(client_sockets[j]);
                        close(client_sockets[j]);
                        client_sockets[j] = 0;
                    }
                }
            }
        }
        else if (pid > 0) {
        }
        else {
            perror("Fork failed");
            exit(EXIT_FAILURE);
        }
    }

    for (int i = 0; i < NUM_CHILDREN; ++i) {
        wait(NULL);
    }
}
```

На листинге 3 представлена функция обработки HTTP-запроса, парсинг его и проверка на валидность.

Листинг 3 – Обработка и парсинг HTTP-запроса

```
void handle_client(int client_socket) {
    char buffer[1024];
    char method[10], path[PATH_MAX], protocol[10];
    if (recv(client_socket, buffer, sizeof(buffer), 0) <= 0) {
        perror("Receive failed");
        return;
    }
    sscanf(buffer, "%s %s %s", method, path, protocol);

    if (strcmp(method, "GET") != 0 && strcmp(method, "HEAD") != 0) {
        send_error(client_socket, METHOD_NOT_ALLOWED);
        return;
    }
    char full_path[PATH_MAX];
    sprintf(full_path, "%s%s", STATIC_ROOT, path);

    char resolved_path[PATH_MAX];
    if (realpath(full_path, resolved_path) == NULL) {
        send_error(client_socket, NOT_FOUND);
        return;
    }
    if (strncmp(resolved_path, STATIC_ROOT, strlen(STATIC_ROOT)) != 0) {
        send_error(client_socket, FORBIDDEN);
        return;
    }

    if (strcmp(path, "/") == 0) {
        strcat(full_path, "index.html");
    }

    struct stat path_stat;
    if (stat(full_path, &path_stat) == 0 && S_ISDIR(path_stat.st_mode)) {
        if (path[strlen(path) - 1] != '/') { strcat(path, "/"); }
        send_directory_listing(client_socket, path, full_path);
        return;
    }

    send_file(client_socket, full_path, method);
}
```

На листинге 4 представлены функции обработки HTTP-ответов. Состав-

ление хедера и тела ответа в зависимости от ситуации.

Листинг 4 – Обработка HTTP-ответов

```
void send_directory_listing(int client_socket, const char* current_path, const char*
    directory_path) {
    DIR* directory = opendir(directory_path);

    if (directory == NULL) {
        send_error(client_socket, NOT_FOUND);
        return;
    }

    char headers[1024];

    sprintf(headers, "HTTP/1.1 200 OK\r\nContent-Type: text/html\r\n\r\n");
    send(client_socket, headers, strlen(headers), 0);

    char buffer[1024];
    sprintf(buffer, "<html><head><title>Directory Listing</title></head><body><h1>Directory
    Listing</h1><ul>");

    struct dirent* dir_entry;
    while ((dir_entry = readdir(directory)) != NULL)
    {
        if (strcmp(dir_entry->d_name, ".") == 0) { continue; }

        char full_entry_path[255];
        sprintf(full_entry_path, "%s/%s", directory_path, dir_entry->d_name);

        struct stat path_stat;
        if (stat(full_entry_path, &path_stat) == 0 && S_ISDIR(path_stat.st_mode))
        {
            sprintf(buffer + strlen(buffer), "<li><a href=\"%s%s/\">%s/</a></li>", current_path,
    dir_entry->d_name, dir_entry->d_name);
        }
        else
        {
            sprintf(buffer + strlen(buffer), "<li><a href=\"%s%s\">%s</a></li>", current_path,
    dir_entry->d_name, dir_entry->d_name);
        }
    }

    strcat(buffer, "</ul></body></html>");

    send(client_socket, buffer, strlen(buffer), 0);

    closedir(directory);
}

void send_file(int client_socket, const char* file_path, const char* method) {
    FILE* file = fopen(file_path, "rb");
    if (file == NULL) {
        send_error(client_socket, NOT_FOUND);
        return;
    }

    fseek(file, 0, SEEK_END);
    long file_size = ftell(file);
    fseek(file, 0, SEEK_SET);

    char headers[1024];
    sprintf(headers, "HTTP/1.1 200 OK\r\nContent-Length: %ld\r\n", file_size);

    const char* content_type = "text/plain";
    const char* file_ext = strrchr(file_path, '.');
    if (file_ext != NULL) {
        if (strcmp(file_ext, ".html") == 0) {
            content_type = "text/html";
        }
        else if (strcmp(file_ext, ".css") == 0) {
            content_type = "text/css";
        }
        else if (strcmp(file_ext, ".js") == 0) {
            content_type = "application/javascript";
        }
        else if (strcmp(file_ext, ".png") == 0) {
            content_type = "image/png";
        }
        else if (strcmp(file_ext, ".jpg") == 0 || strcmp(file_ext, ".jpeg") == 0) {
            content_type = "image/jpeg";
        }
        else if (strcmp(file_ext, ".swf") == 0) {
            content_type = "application/x-shockwave-flash";
        }
        else if (strcmp(file_ext, ".gif") == 0) {
            content_type = "image/gif";
        }
    }
```

```
82
83      sprintf(headers + strlen(headers), "Content-Type: %s\r\n\r\n", content_type);
84
85
86      send(client_socket, headers, strlen(headers), 0);
87
88      if (strcmp(method, "HEAD") == 0) {
89          fclose(file);
90          return;
91      }
92
93      char buffer[1024];
94      size_t bytes_read;
95      while ((bytes_read = fread(buffer, 1, sizeof(buffer), file)) > 0) {
96          send(client_socket, buffer, bytes_read, 0);
97      }
98
99      fclose(file);
100 }
101
102 void send_error(int client_socket, int status_code) {
103     char response[1024];
104     sprintf(response, "HTTP/1.1 %d\r\nContent-Length: 0\r\nContent-Type: text/html\r\n\r\n",
        status_code);
105
106     log_message(LOG_INFO, "ERROR %s", response);
107     log_message(LOG_DEBUG, "Debug message: %d", 42);
108     // log_message(LOG_WARNING, "Something unexpected happened!");
109     // log_message(LOG_ERROR, "Error occurred: %s", "File not found");
110
111     send(client_socket, response, strlen(response), 0);
112     // log_message("Sent forbidden error response to client");
113 }
```

На листинге 5 представлена кастомная функция для логирования сервера и вывода всех логов в специально отведенный файл.

Листинг 5 – Логгер

```
1  void log_message(LogLevel level, const char* format, ...) {
2      time_t rawtime;
3      struct tm* timeinfo;
4      time(&rawtime);
5      timeinfo = localtime(&rawtime);
6      LogLevel current_log_level = LOG_LEVEL;
7
8      if (level > current_log_level) { return; }
9
10     FILE* log_file = fopen(LOG_FILE, "a");
11     if (log_file == NULL) {
12         perror("Error opening log file");
13         return;
14     }
15
16     char time_str[50];
17
18     strftime(time_str, sizeof(time_str), "[%Y-%m-%d %H:%M:%S]", timeinfo);
19
20     va_list args;
21     va_start(args, format);
22
23     fprintf(log_file, "%s ", time_str);
24
25     switch (level) {
26     case LOG_DEBUG:
27         fprintf(log_file, "[DEBUG] ");
28         break;
29     case LOG_INFO:
30         fprintf(log_file, "[INFO] ");
31         break;
32     case LOG_WARNING:
33         fprintf(log_file, "[WARNING] ");
34         break;
35     case LOG_ERROR:
36         fprintf(log_file, "[ERROR] ");
37         break;
38     }
39
40     vfprintf(log_file, format, args);
41
42     va_end(args);
43     fclose(log_file);
44 }
```

3.3 Демонстрация работы программы

На рисунке 3.1 показана работа home директории сервера.

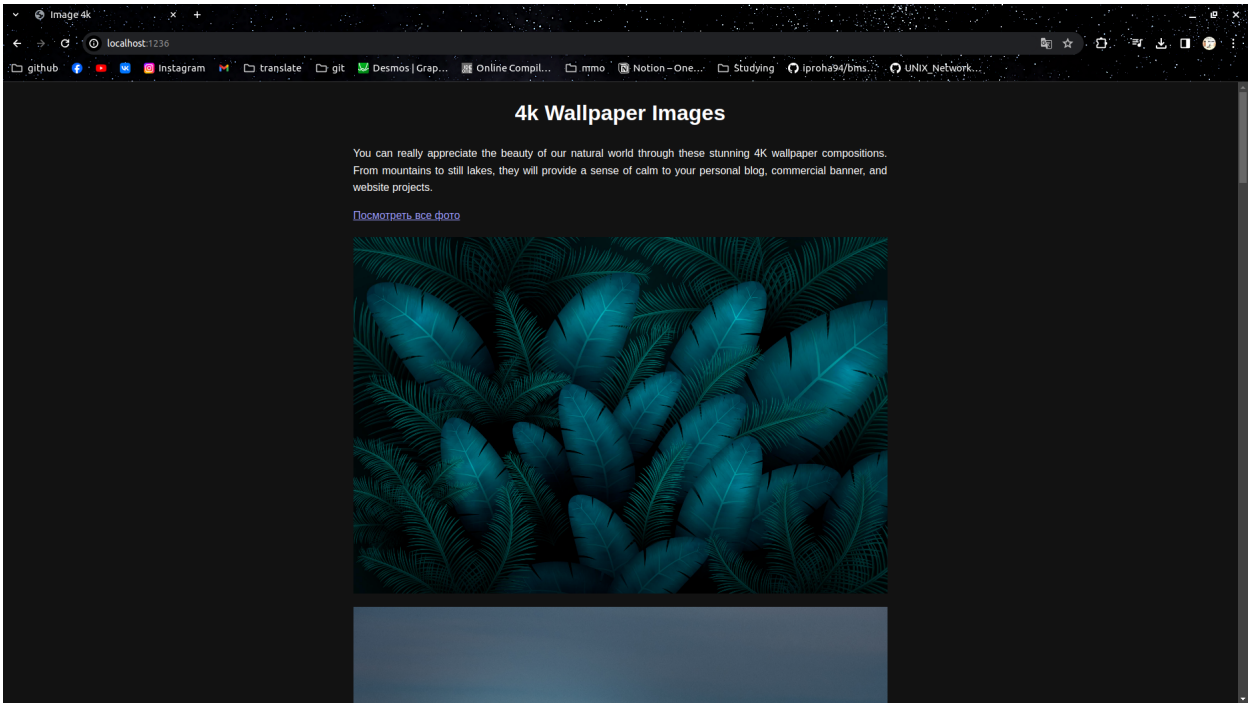

Рисунок 3.1 – Home директория сервера

На рисунке 3.2 показано корректное отображение списка файлов в запрашиваемой директории.

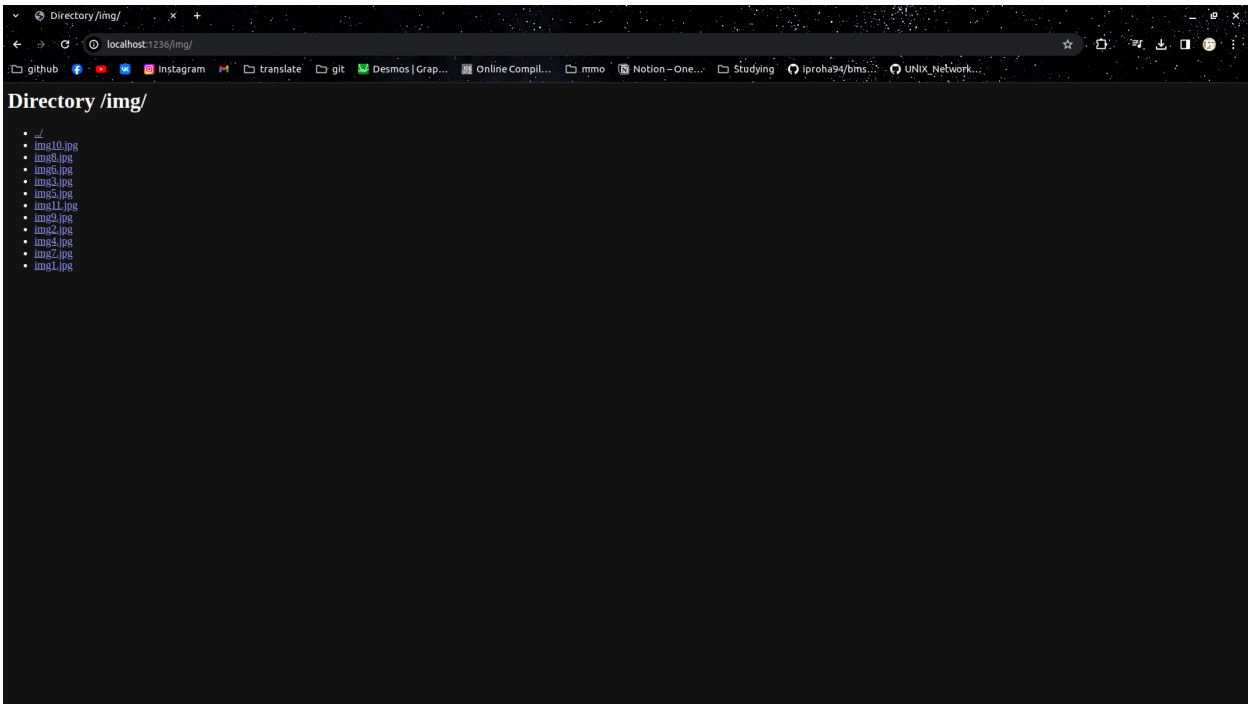

Рисунок 3.2 – Список файлов в запрашиваемой директории

На рисунке 3.3 показана отдача фото с сервера.

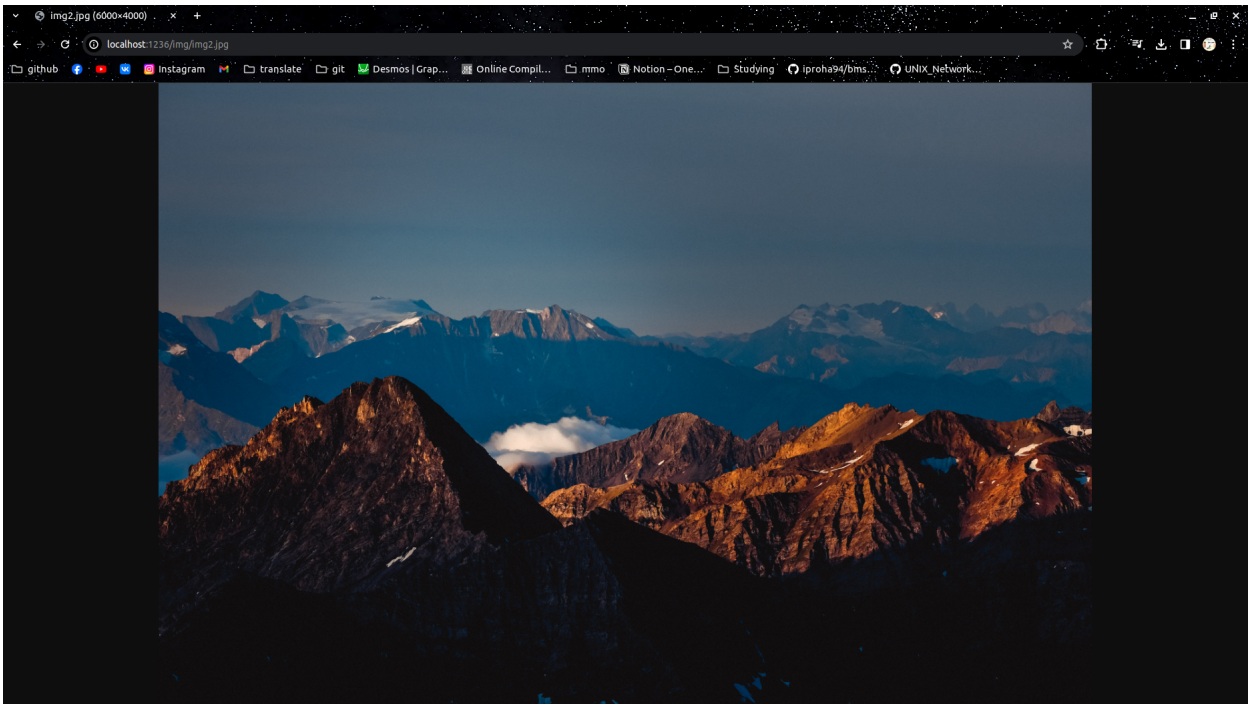

Рисунок 3.3 – Отдача фото с сервера

Вывод

Был реализован статик-сервер на языке Си, а также проведены проверки его функционала.

4 Исследовательский раздел

4.1 Описание эксперимента

В ходе исследования было выполнено нагрузочное тестирование разработанного статик-сервера и nginx с использованием ApacheBench. Тестирование было направлено на проверку отправки запросов к index.html.

4.2 Результаты эксперимента

На рисунках 4.1 – 4.2 рассматривались сценарии 500 и 1000 запросов пользователей каждую секунду в течение 5 секунд.

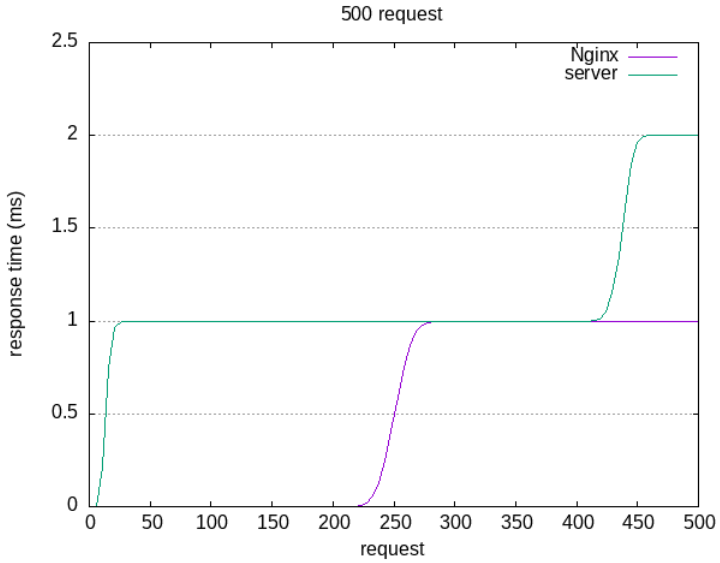

Рисунок 4.1 – 500 запросов

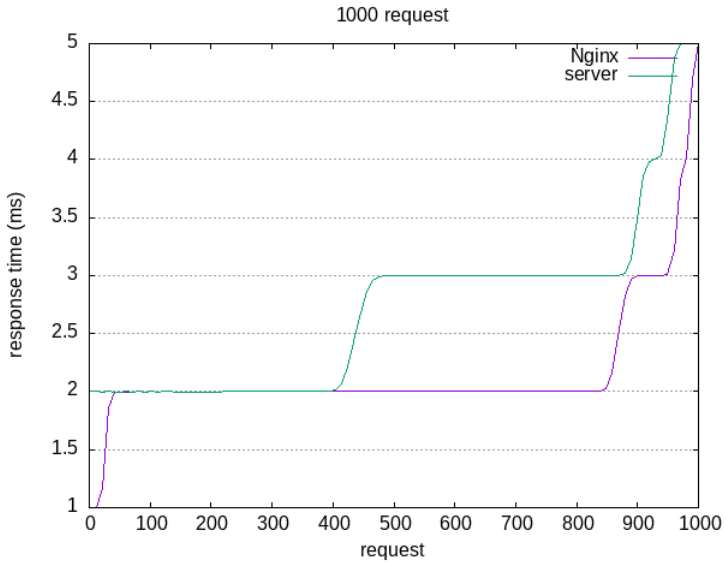

Рисунок 4.2 – 1000 запросов

Вывод

Проведено тестирование разработанного статик-сервера и nginx, в ходе которого выявлено, что nginx работает эффективнее.

ЗАКЛЮЧЕНИЕ

В процессе выполнения курсовой работы был выполнен анализ предметной области, выделены требования к разрабатываемому приложению. Было разработано приложение, которое прошло успешное тестирование на корректность работы программы. Также были проведены тесты на нагрузку для оценки поведения приложения при высокой нагрузке.

При написании данной работы:

- проведена формализация задачи и определен необходимый функционал;
- исследована предметная область веб-серверов;
- спроектировано приложение;
- реализовано приложение;
- приложение протестировано на предмет корректности.

Таким образом, поставленные задачи были выполнены, цель курсовой работы была достигнута.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Таненбаум Э. Компьютерные сети. 6 изд. Питер, 2024. с. 992.
2. Олифер В., Олифер Н. Компьютерные сети. Принципы, технологии, протоколы. Учебник. Питер, 2016. с. 996.
3. Visual Studio Code. URL: https://code.visualstudio.com/.