Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления (ИУ)»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии (ИУ7)»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

«Разработка статического сервера»

Студент ИУ7-73Б _____ С. М. Сабуров
(Группа) (Подпись, дата) (И. О. Фамилия)

Руководитель курсовой работы _____ _____
(Подпись, дата) (И. О. Фамилия)

2024 г.

РЕФЕРАТ

В рамках данной курсовой работы был реализован статик-сервер.

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

Рассчетно-пояснительная записка к курсовой работе содержит 23 страниц, 3 иллюстраций, 4 источника.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель проекта состоит в создании статического веб-сервера, способного обрабатывать GET и HEAD запросы. Статический веб-сервер — это программное решение, которое может обслуживать только статические веб-страницы. Статические страницы - это файлы HTML, CSS, JavaScript и другие файлы, которые не изменяются на сервере и не зависят от действий пользователя.

В современном мире компьютерные сети играют огромную роль в повседневной жизни. Они позволяют нам обмениваться информацией, работать удаленно и получать доступ к различным ресурсам в Интернете. Создание собственного статического веб-сервера является важным шагом в изучении компьютерных сетей и веб-технологий.

При разработке статического веб-сервера необходимо обеспечить базовую безопасность сервера, включая отслеживание и контроль доступа к файлам за пределами конкретной директории. Это поможет защитить сервер от несанкционированного доступа и сохранить целостность данных.

Для достижения поставленной цели, предполагается выполнение следующих задач:

— провести формализацию задачи и определить небоходимый функционал;

— провести анализ предметной области веб-серверов;

— разработать приложение;

— протестировать приложение;

— замерить работоспособность веб-сервера.

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 threadpool + 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.

HTTP является протоколом запроса и ответа. Браузер совершает запрос к веб-серверу, используя синтаксис HTTP. Веб-сервер в свою очередь отвечает сообщением, содержащим запрошенный ресурс.

В качестве версии HTTP была выбрана HTTP/1.1, поскольку она поддерживает режим «постоянного соединения», что позволяет в случае необходимости совершать сразу несколько запросов за одно соединение [1].

1.3 Сокеты UNIX

Сокеты UNIX - это механизм межпроцессного взаимодействия в операционных системах семейства UNIX. Они реализуются через файловую систему, где каждый сокет представлен в виде специального файла, и позволяют процессам обмениваться данными без использования сетевых интерфейсов [2].

Сокеты UNIX предоставляют способ для процессов обмениваться данными напрямую, обеспечивая эффективную и безопасную передачу информации между различными частями приложения.

Одной из причин выбора сокетов Unix при разработке статического веб-сервера на языке Си является их производительность. Кроме того, использование сокетов Unix позволяет упростить код сервера, так как не нужно работать с сетевыми интерфейсами напрямую.

1.4 Вывод

На основе проделанного 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.

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 2.1 – Пример HTTP-запроса через Postman

```
1 GET /img/prison.jpeg HTTP/1.1
2 User-Agent: PostmanRuntime/7.35.0
3 Accept: */*
4 Postman-Token: 7af874dc-5a4a-4678-a088-65bc9ca2b09e
5 Host: localhost:5600
6 Accept-Encoding: gzip, deflate, br
7 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 threadpool в сочетании с select для обработки входящих подключений. Этот подход позволяет создать множество потоков-обработчиков заранее (threadpool) 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.

2.5 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, который был рекомендован преподавателем. 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 3.1 представлена реализация алгоритма threadpool + select.

Листинг 3.1 – Механизм threadpool + select

```
int listen_and_serve(Web_server *server, int clients_cap){
    server_active = 1;
    int err = listen(server->listenfd, MAX_CONN);
    if (err != 0){
        log_message(ERROR_LEVEL,"listen_and_serve (listen)\n");
        return -1;
    }
    Clients_info *clients_info = new_clients_info(clients_cap);
    if (clients_info == NULL){
        log_message(ERROR_LEVEL,"listen_and_serve (clients_info)\n");
        return -1;
    }
    init_clients_info(clients_info, server->listenfd);
    while(server_active){
        log_message(DEBUG_LEVEL,"nready is %d\nSELECT UNBLOCKED\n", nready);
        if (FD_ISSET(server->listenfd, &rset)){
            log_message(DEBUG_LEVEL,"attempt to accept connection\n");
            int err = accept_connection(server, clients_info);
            if (err != 0){
                log_message(ERROR_LEVEL,"listen and serve (accept)\n");
                free_clients_info(clients_info);
                return -1;
            }
            if (--nready <= 0){
                log_message(DEBUG_LEVEL,"continuing\n");
                continue;
            }
```

```
28          }else{
29              log_message(DEBUG_LEVEL,"no new connections\n");
30          }
31
32          for (int i = 0; i <= clients_info->max_client_ind; i ++){
33              pthread_rwlock_rdlock(&clients_info->clients_RWmutex);
34
35              int cur_client_fd = clients_info->clients_fd[i];
36              log_message(DEBUG_LEVEL,"cur ind %d\n", i);
37              if (cur_client_fd != -1){
38                  if (FD_ISSET(cur_client_fd, &rset)){
39                      log_message(DEBUG_LEVEL,"%d is set\n", cur_client_fd);
40                      Conn_info *conn_info = new_conn_info(clients_info,
                            cur_client_fd, i);
41                      if (conn_info == NULL){
42                          log_message(ERROR_LEVEL,"listen and serve (new conn
                                info)\n");
43                          pthread_rwlock_unlock(&clients_info->clients_RWmutex
                                );
44                          free_clients_info(clients_info);
45                          return -1;
46                      }
47                      FD_CLR(cur_client_fd, &clients_info->clients_fdset);
48                      add_task(server->pool, new_task((void * (*)(void *))
                            handle_connection, (void *)conn_info));//
49                      if (--nready <= 0){
50                          pthread_rwlock_unlock(&clients_info->clients_RWmutex
                                );
51                          break;
52                      }
53                  }
54              }
55              pthread_rwlock_unlock(&clients_info->clients_RWmutex);
56          }
57      }
58      free_clients_info(clients_info);
59      return 0;
60  }
```

На листинге 3.2 представлена функция обработки HTTP-запроса, парсинг его и проверка на валидность.

Листинг 3.2 – Обработка и парсинг HTTP-запроса

```
1  int handle_connection(void *arg){
2      Conn_info *conn_info = (Conn_info *)arg;
3
4      char buf[BUF_SIZE];
5      log_message(DEBUG_LEVEL,"\t BEFORE READ from %d\n", conn_info->
```

```
        cur_client);
6       int err = read(conn_info->cur_client, buf, BUF_SIZE);
7       if (err <= 0){
8           log_message(WARNING_LEVEL,"READ NOTHING of %d (handler)\n",
                conn_info->cur_client);
9           disconnect_client(conn_info);
10          //free(sockfd);
11          return -1;
12      }
13      log_message(DEBUG_LEVEL,"\t AFTER READ from %d: \n%s\n", conn_info->
            cur_client, buf);
14
15      Http_request req = parse_request(buf);
16      char err_str[256];
17      if (strcmp(req.method, "GET")  && strcmp(req.method, "HEAD")){
18          log_message(INFO_LEVEL,"UNKNOWN REQUEST: %s is not GET\n", req.
                method);
19          resp_http_405(err_str);
20          err = write(conn_info->cur_client, err_str, strlen(err_str));
21          if (err <= 0){
22              log_message(WARNING_LEVEL,"WRITE ERROR (handler)\n");
23              disconnect_client(conn_info);
24              return 1;
25          }
26          disconnect_client(conn_info);
27          return 0;
28      }
29      char filepath[PATH_MAX];
30      if (strcmp(req.url, "/") == 0){
31          strcat(req.url, "index.html");
32      }
33      set_file_from_url(err_str, filepath, req.url);
34      if (strcmp(err_str, "OK")){
35          log_message(WARNING_LEVEL,"\t\t\tBAD URL\t\t\t\n");
36          err = write(conn_info->cur_client, err_str, strlen(err_str));
37          if (err <= 0){
38              log_message(WARNING_LEVEL,"WRITE ERROR (handler)\n");
39              disconnect_client(conn_info);
40              return 1;
41          }
42          disconnect_client(conn_info);
43          return 0;
44      }
45
46      struct stat path_stat;
47      //                  -
48      if (stat(filepath, &path_stat) == 0 && S_ISDIR(path_stat.st_mode)) {
49          if (req.url[strlen(req.url) - 1] != '/'){
```

```
50              strcat(req.url, "/");
51          }
52          log_message(DEBUG_LEVEL,"BEFORE DIR SEND\n");
53          err = send_dir(conn_info->cur_client, filepath, req.url);
54          if (err <= 0){
55              log_message(WARNING_LEVEL,"WRITE ERROR (handler)\n");
56              disconnect_client(conn_info);
57              return 1;
58          }
59          log_message(DEBUG_LEVEL,"AFTER DIR SEND\n");
60          disconnect_client(conn_info);
61          return 0;
62      }
63      log_message(DEBUG_LEVEL,"BEFORE FILE SEND\n");
64      err = send_file(conn_info->cur_client, filepath);
65      if (err <= 0){
66          log_message(WARNING_LEVEL,"WRITE ERROR (handler)\n");
67          disconnect_client(conn_info);
68          return 1;
69      }
70      log_message(DEBUG_LEVEL,"AFTER FILE SEND\n");
71      disconnect_client(conn_info);
72      log_message(INFO_LEVEL,"Successfully handled\n");
73      return 0;
74  }
```

На листинге 3.3 представлены функции обработки HTTP-ответов. Составление хедера и тела ответа в зависимости от ситуации.

Листинг 3.3 – Обработка HTTP-ответов

```
1   int send_dir(int client_fd, char *fname, char *url){
2       DIR *fd = opendir(fname);
3       if (fd == NULL){
4           log_message(WARNING_LEVEL,"send dir (opendir)\n");
5           return -1;
6       }
7       struct dirent *dir;
8       Http_response resp;
9       strcpy(resp.code, "200");
10      strcpy(resp.version, "HTTP/1.1");
11      strcpy(resp.status, "OK");
12      resp.headers[0] = '\0';
13      add_header(resp.headers, "text/html");
14      char resp_buf[RESPONSE_LEN];
15      response_to_string(resp_buf, resp);
16      int err = write(client_fd, resp_buf, strlen(resp_buf));
17      if (err <= 0){
18          log_message(WARNING_LEVEL, "send dir (write)\n");
```

```
19          closedir(fd);
20          return -1;
21      }
22      char body_buf[BUF_SIZE];
23      sprintf(body_buf, "<html><head><title>Dir %s</title></head><body><h1>Dir
            %s</h1><ul>", url, url);
24      while((dir = readdir(fd)) != NULL){
25          if ((strcmp(dir->d_name, ".") == 0) || (strcmp(dir->d_name, "..") ==
                0)){
26              continue;
27          }
28
29          char full_fname[FILENAME_MAX];
30          sprintf(full_fname, "%s/%s", fname, dir->d_name);
31          struct stat f_stat;
32          if (stat(full_fname, &f_stat) == 0 && S_ISDIR(f_stat.st_mode)){
33              sprintf(body_buf + strlen(body_buf), "<li><a href=\"%s%s/\">%s
                    /</a></li>", url, dir->d_name, dir->d_name);
34          }else{
35              sprintf(body_buf + strlen(body_buf), "<li><a href=\"%s%s\">%s</a
                    ></li>", url, dir->d_name, dir->d_name);
36          }
37      }
38      closedir(fd);
39      sprintf(body_buf + strlen(body_buf), "</ul></body></html>");
40      err = write(client_fd, body_buf, strlen(body_buf));
41      if (err <= 0){
42          log_message(WARNING_LEVEL, "send dir (write)\n");
43          return -1;
44      }
45      return 0;
46
47  }
48
49  long get_file_size(char *filename){
50      struct stat f_stat;
51      if (stat(filename, &f_stat) < 0){
52          return -1;
53      }
54
55      return f_stat.st_size;
56  }
57
58
59  int send_file(int client_fd, char *fname){
60      Http_response resp;
61      strcpy(resp.code, "200");
62      strcpy(resp.version, "HTTP/1.1");
```

```
63      strcpy(resp.status, "OK");
64      resp.headers[0] = '\0';
65
66      long file_size = get_file_size(fname);
67
68      char content_type[128];
69      strcpy(content_type, get_content_type(fname));
70      if (strcmp(content_type, "") == 0){
71          log_message(WARNING_LEVEL, "content type trouble\n");
72          add_header(resp.headers, "text/html");
73
74          char resp_buf[RESPONSE_LEN];
75          response_to_string(resp_buf, resp);
76          write(client_fd, resp_buf, strlen(resp_buf));
77          /*if (err <= 0){
78              log_message(DEBUG_LEVEL, "send dir (write)\n");
79              return -1;
80          }*/
81          char *body_buf = "<html>Unknown content-type</html>\r\n";
82          write(client_fd, body_buf, strlen(body_buf));
83          /*if (err <= 0){
84              printf("send dir (write)\n");
85              return -1;
86          }*/
87      }
88      add_header(resp.headers, content_type);
89      char resp_buf[RESPONSE_LEN];
90      response_to_string(resp_buf, resp);
91      int err = write(client_fd, resp_buf, strlen(resp_buf));
92      if (err <= 0){
93              log_message(WARNING_LEVEL,"send dir (write)\n");
94              return -1;
95      }
96      FILE *f = fopen(fname, "r");
97      if (f == NULL){
98          char err_str[128];
99          resp_http_404(err_str);
100         err = write(client_fd, err_str, strlen(err_str));
101         if (err <= 0){
102             log_message(WARNING_LEVEL,"send dir (write)\n");
103             return -1;
104         }
105     }
106     char body_buf[BUF_SIZE];
107     int bytes_read;
108     while((bytes_read = fread(body_buf, 1, BUF_SIZE, f)) > 0){
109         err = write(client_fd, body_buf, bytes_read);
110         if (err <= 0){
```

```
111             log_message(WARNING_LEVEL,"send dir (write)\n");
112             return -1;
113         }
114     }
115     fclose(f);
116     return 0;
117 
118 }
```

На листинге 3.4 представлена кастомная функция для логирования сервера и вывода всех логов в специально отведенный файл.

Листинг 3.4 – Логгер

```
1  int log_level = INFO_LEVEL;
2  
3  void log_message(int level, char* format, ...) {
4      FILE *f = fopen(LOG_FILE, "a");
5      if (f == NULL){
6          printf("cant open log file\n");
7          return;
8      }
9      va_list args;
10     va_start(args, format);
11     if (level >= log_level) {
12 
13         time_t current_time;
14         char* c_time_string;
15         current_time = time(NULL);
16         c_time_string = ctime(&current_time);
17         char* level_string;
18         switch (level) {
19             case DEBUG_LEVEL:
20                 level_string = "DEBUG";
21                 break;
22             case INFO_LEVEL:
23                 level_string = "INFO";
24                 break;
25             case WARNING_LEVEL:
26                 level_string = "WARNING";
27                 break;
28             case ERROR_LEVEL:
29                 level_string = "ERROR";
30                 break;
31         }
32          fprintf(f, "%s %s: ", c_time_string, level_string);
33          vfprintf(f, format, args);
34     }
35 
```

```
36      va_end(args);
37      fclose(f);
38  }
```

3.3 Демонстрация работы программы

На рисунке 3.1 показана работа home директории сервера. Сервер посвящен программному обеспечению Docker.

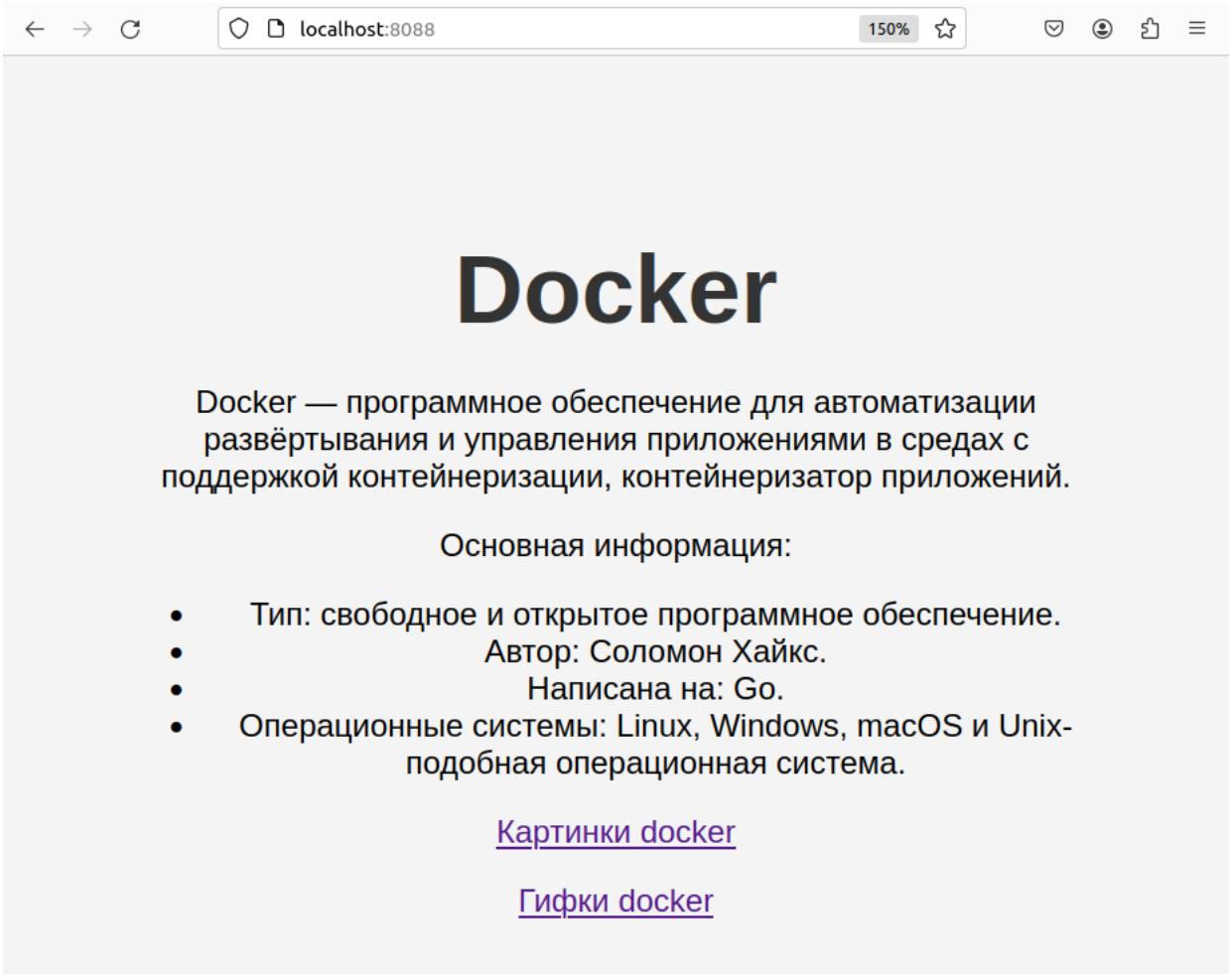

Рисунок 3.1 – index.html

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

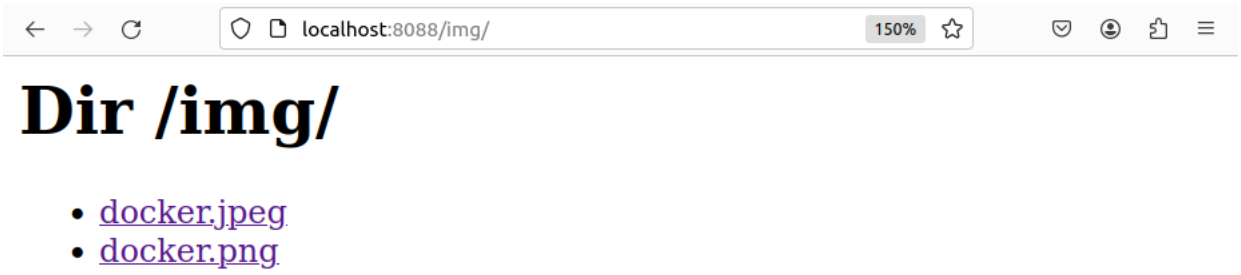

Рисунок 3.2 – Директория img

На рисунке 3.3 показана отдача фото с сервера.

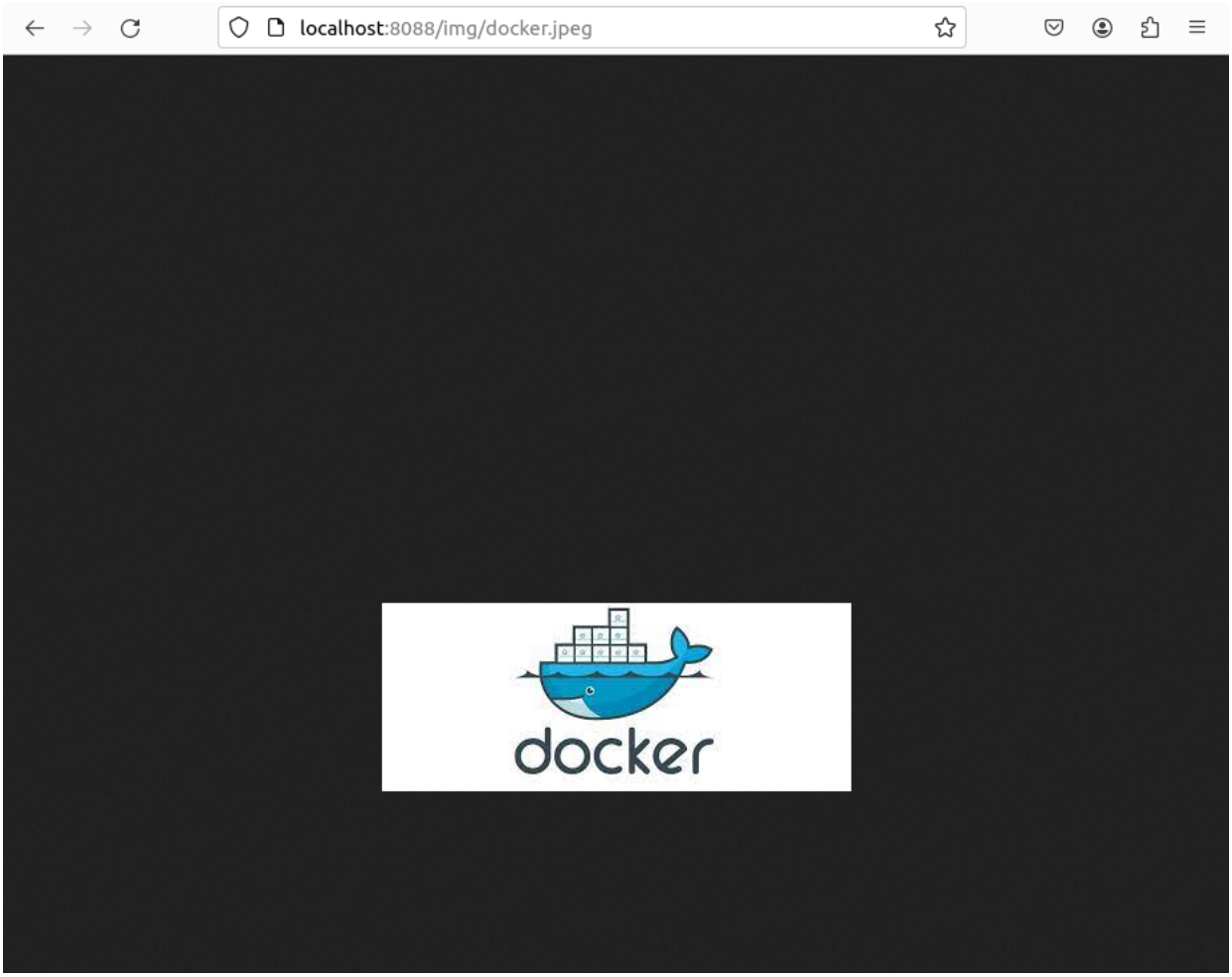

Рисунок 3.3 – Изображение с docker китом

3.4 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 программы wrk[4] . Тестирование было направлено на проверку отправки запросов к index.html.

Параметры тестирования:

— количество потоков - 12;

— количество подключений - 400;

— длительность тестирования - 30 секунд;

4.2 Результаты эксперимента

	Прочитано данных	Выполнено запросов	Время отклика (мс)
nginx	2.61 ГБ	1567563	7.9
сервер	235.82 МБ	155131	301.88

4.3 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 работы была достигнута.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Б. П.* HTTP/2 в действии. — 2-е изд. — ДМК пресс, 2021. — С. 425.
2. *Олифер В.*, *Олифер Н.* Компьютерные сети. Принципы, технологии, протоколы. Учебник. — Питер, 2016. — С. 996.
3. Visual Studio Code. — URL: `https://code.visualstudio.com/`.
4. Программа wrk. — URL: `https://github.com/wg/wrk`.