



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»

---

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

---

# РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

## *К КУРСОВОЙ РАБОТЕ*

### *НА ТЕМУ:*

*«Разработка сервера для отдачи статического  
содержимого с диска»*

Студент ИУ7-71Б  
(Группа)

\_\_\_\_\_  
(Подпись, дата)

Постнов С. А.  
(Фамилия И. О.)

Руководитель курсовой работы

\_\_\_\_\_  
(Подпись, дата)

Клочков М. С.  
(Фамилия И. О.)

*2024 г.*

# СОДЕРЖАНИЕ

<b>ВВЕДЕНИЕ</b>	<b>3</b>
<b>1 Аналитический раздел</b>	<b>4</b>
1.1 Архитектура thread-pool . . . . .	4
1.2 Системный вызов pselect . . . . .	5
<b>2 Конструкторский раздел</b>	<b>7</b>
2.1 Схема алгоритма работы сервера . . . . .	7
<b>3 Технологический раздел</b>	<b>8</b>
3.1 Требования к разрабатываемой программе . . . . .	8
3.2 Средства реализации . . . . .	8
3.3 Реализация сервера . . . . .	9
3.4 Примеры работы программы . . . . .	15
<b>4 Исследовательский раздел</b>	<b>18</b>
4.1 Технические характеристики . . . . .	18
4.2 Описание исследования . . . . .	18
4.3 Результаты исследования . . . . .	18
<b>ЗАКЛЮЧЕНИЕ</b>	<b>20</b>
<b>СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ</b>	<b>21</b>

# ВВЕДЕНИЕ

Статический веб-сервер — сервер, который обслуживает статические файлы по запросу клиента. Статические файлы — файлы, содержимое которых не изменяется динамически на стороне сервера. Статический сервер предназначен для чтения файлов из диска и отправления их клиенту, как правило, по протоколу HTTP.

Целью курсовой работы является разработка сервера для отдачи статического содержимого с диска. Архитектура сервера должна быть основана на пуле потоков (thread-pool) совместно `cpselect()`.

Для достижения поставленной цели необходимо решить следующие задачи:

- 1) описать предметную область;
- 2) спроектировать схему алгоритма работы сервера;
- 3) выбрать средства реализации сервера;
- 4) провести сравнение реализованного сервера с известными аналогами.

# 1 Аналитический раздел

## 1.1 Архитектура thread-pool

Архитектура пул потоков (thread-pool) — модель управления потоками выполнения (threads), которая предусматривает предварительное создание фиксированного или динамически изменяемого пула потоков, используемых для обработки задач из общей очереди. Такая архитектура обеспечивает эффективное распределение вычислительных ресурсов и уменьшение накладных расходов, связанных с созданием и уничтожением потоков. Она является одной из самых распространенных архитектур многопоточности Object Request Broker Architecture (CORBA), используемых в реализациях Object Request Broker (ORB), и была принята веб-серверами, такими как Microsoft Internet Information Server (IIS) [1].

Основными компонентами архитектуры thread-pool являются:

- 1) пул потоков (thread-pool);
- 2) очередь задач (task queue);
- 3) менеджер пула (pool manager);
- 4) задачи (tasks).

Архитектура пула потоков представлена на рисунке 1.1.

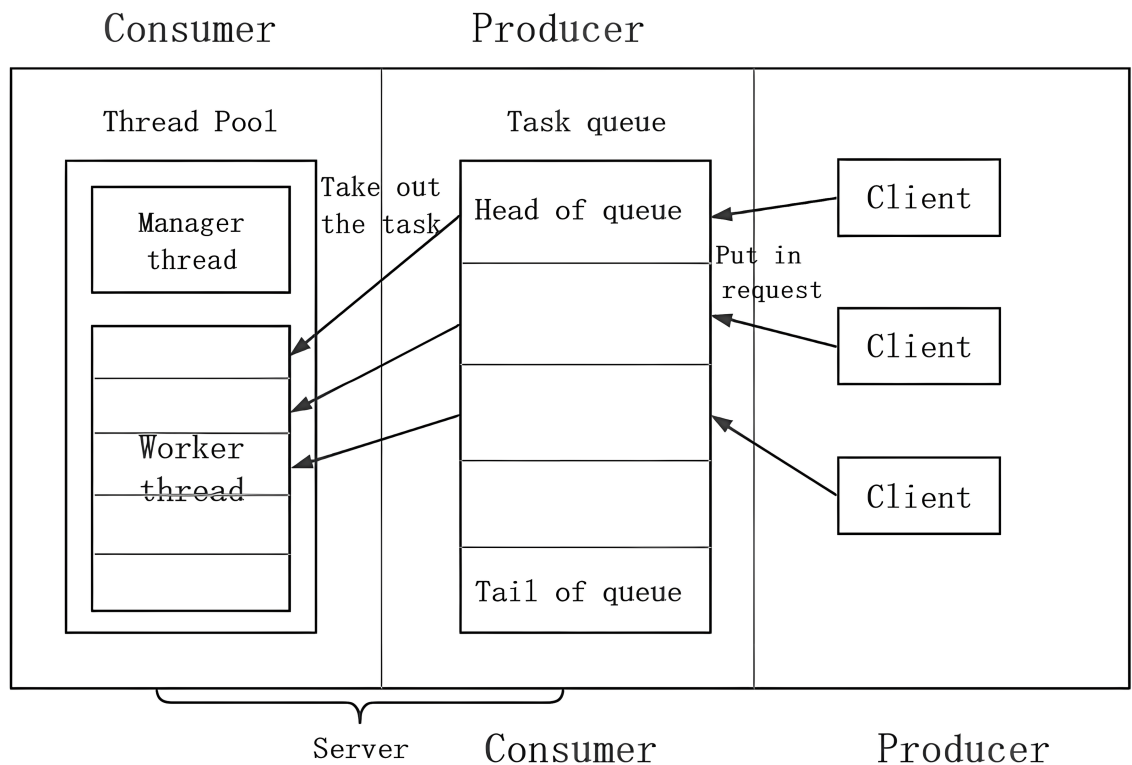


Рисунок 1.1 – Архитектура пула потоков

## 1.2 Системный вызов pselect

В Unix процесс выполняет ввод-вывод по одному файловому дескриптору за раз, поэтому происходит блокировка и снижение производительности программы. Чтобы избежать этой проблемы, необходимо использовать системный вызов **pselect()**, который позволяет программе отслеживать несколько файловых дескрипторов. Программа ожидает, пока один или несколько файловых дескрипторов не станут готовы к определённому классу операций ввода-вывода, не блокируя их и обеспечивая многопоточный синхронный ввод-вывод [2].

Модель неблокирующего синхронного ввода-вывода, в которой применяется `pselect()`, представлена на рисунке 1.2.

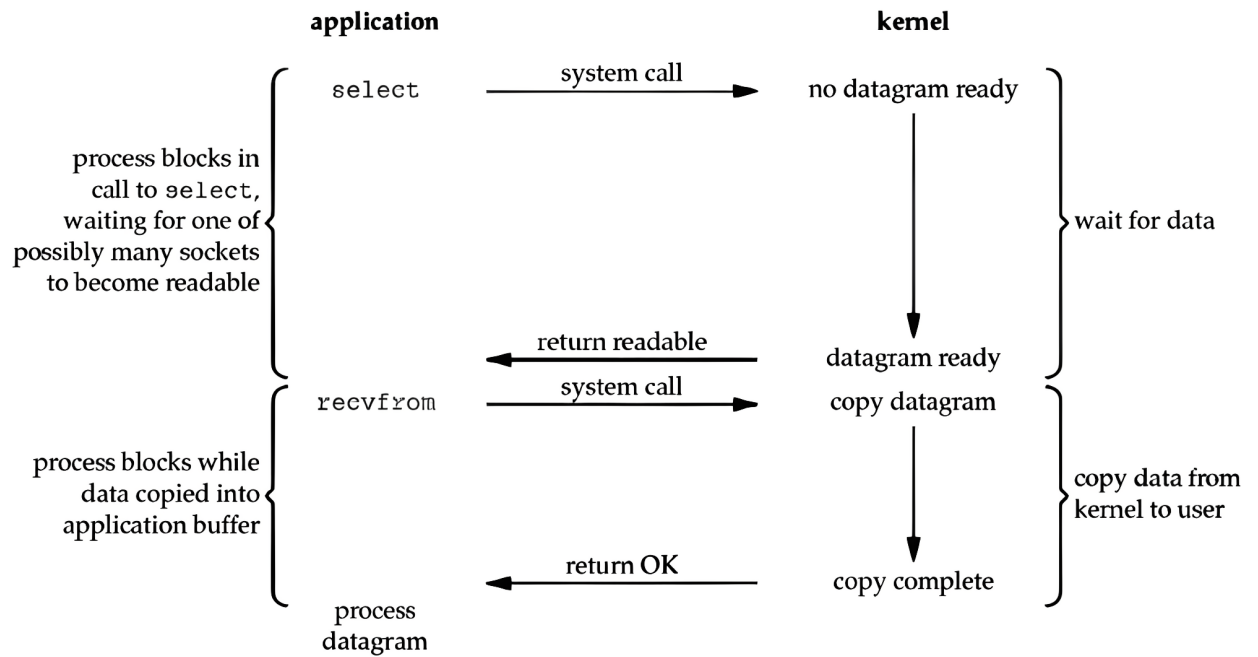


Рисунок 1.2 – Модель неблокирующего синхронного ввода-вывода

## 2 Конструкторский раздел

### 2.1 Схема алгоритма работы сервера

На рисунке 2.1 представлена схема алгоритма работы сервера.



Рисунок 2.1 – Схема алгоритма работы сервера

## **3 Технологический раздел**

### **3.1 Требования к разрабатываемой программе**

Разрабатываемое программное обеспечение должно удовлетворять следующим требованиям:

- 1) поддержка запросов **GET** и **HEAD**;
- 2) поддержка статусов 200, 403, 404 и 405 (на неподдерживаемые запросы);
- 3) поддержка корректной передачи файлов размером до 128 Мб;
- 4) возврат по умолчанию html-страницы с css-стилем;
- 5) запись информации о событиях;
- 6) минимальные требования к безопасности серверов статического содержимого.

### **3.2 Средства реализации**

В качестве языка программирования для реализации веб-сервера был выбран язык C [3].

В качестве среды для разработки была выбрана среда CLion [4].



### 3.3 Реализация сервера

Реализация веб-сервера представлена в листинге 3.1.

Листинг 3.1 – Реализация веб-сервера

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <unistd.h>
5  #include <fcntl.h>
6  #include <pthread.h>
7  #include <sys/socket.h>
8  #include <sys/types.h>
9  #include <netinet/in.h>
10 #include <arpa/inet.h>
11 #include <sys/stat.h>
12 #include <signal.h>
13 #include <time.h>
14
15 #define PORT 8080
16 #define MAX_THREADS 11
17 #define QUEUE_SIZE 64
18 #define ROOT_DIR "./static"
19 #define MAX_BUFFER 4096
20 #define LOG_BUFFER 512
21
22 pthread_mutex_t queue_mutex = PTHREAD_MUTEX_INITIALIZER;
23 pthread_cond_t queue_cond = PTHREAD_COND_INITIALIZER;
24
25 int task_queue[QUEUE_SIZE];
26 int queue_start = 0, queue_end = 0, queue_count = 0;
27
28 FILE *log_file = NULL;
29 int server_fd;
30
31 int endswith(const char *str, const char *suffix) {
32     if (!str || !suffix)
33         return 0;
34
35     const size_t str_len = strlen(str);
36     const size_t suffix_len = strlen(suffix);
37     if (suffix_len > str_len)
```

```

38         return 0;
39
40         return strncmp(str + str_len - suffix_len, suffix,
41             suffix_len) == 0;
42     }
43
44 void log_event(const char *message) {
45     pthread_mutex_lock(&queue_mutex);
46     if (log_file) {
47         const time_t now = time(NULL);
48         fprintf(log_file, "[%s] %s\n", strtok(ctime(&now),
49             "\n"), message);
50         fflush(log_file);
51     }
52     pthread_mutex_unlock(&queue_mutex);
53 }
54
55 void cleanup() {
56     if (log_file)
57         fclose(log_file);
58     if (server_fd > 0)
59         close(server_fd);
60
61     pthread_mutex_destroy(&queue_mutex);
62     pthread_cond_destroy(&queue_cond);
63 }
64
65 void handle_signal(const int sig) {
66     if (sig == SIGINT || sig == SIGKILL) {
67         log_event("Server shutting down...");
68         cleanup();
69         exit(0);
70     }
71 }
72
73 void enqueue_task(const int fd) {
74     pthread_mutex_lock(&queue_mutex);
75     if (queue_count == QUEUE_SIZE) {
76         pthread_mutex_unlock(&queue_mutex);
77         close(fd);
78         return;

```

```

77     }
78
79     task_queue[queue_end] = fd;
80     queue_end = (queue_end + 1) % QUEUE_SIZE;
81     queue_count++;
82     pthread_cond_signal(&queue_cond);
83     pthread_mutex_unlock(&queue_mutex);
84 }
85
86 int dequeue_task() {
87     pthread_mutex_lock(&queue_mutex);
88     while (queue_count == 0)
89         pthread_cond_wait(&queue_cond, &queue_mutex);
90
91     const int fd = task_queue[queue_start];
92     queue_start = (queue_start + 1) % QUEUE_SIZE;
93     --queue_count;
94     pthread_mutex_unlock(&queue_mutex);
95
96     return fd;
97 }
98
99 void send_response(const int fd, const int status, const char
    *status_text, const char *content_type, const char *body,
    const size_t body_length) {
100     char header[MAX_BUFFER];
101     const int header_length = snprintf(header, MAX_BUFFER,
102         "HTTP/1.1 %d %s\r\n"
103         "Content-Length: %zu\r\n"
104         "Content-Type: %s\r\n"
105         "Connection: close\r\n\r\n",
106         status, status_text, body_length, content_type);
107
108     write(fd, header, header_length);
109     if (body && body_length > 0)
110         write(fd, body, body_length);
111     close(fd);
112
113     char log_msg[LOG_BUFFER];
114     snprintf(log_msg, sizeof(log_msg), "Response: %d %s",
        status, status_text);

```

```

115     log_event(log_msg);
116 }
117
118 void serve_static_file(const int fd, const char *file_path) {
119     struct stat file_stat;
120     if (stat(file_path, &file_stat) == -1 ||
121         S_ISDIR(file_stat.st_mode)) {
122         send_response(fd, 404, "Not Found", "text/html",
123             "<h1>404 Not Found</h1>", 22);
124         return;
125     }
126
127     if (access(file_path, R_OK) == -1) {
128         send_response(fd, 403, "Forbidden", "text/html",
129             "<h1>403 Forbidden</h1>", 22);
130         return;
131     }
132
133     const int file_fd = open(file_path, O_RDONLY);
134     if (file_fd == -1) {
135         send_response(fd, 500, "Internal Server Error",
136             "text/html", "<h1>500 Internal Error</h1>", 27);
137         return;
138     }
139
140     char header[MAX_BUFFER];
141     snprintf(header, sizeof(header), "HTTP/1.1 200
142         OK\r\nContent-Length: %ld\r\nContent-Type:
143         %s\r\nConnection: close\r\n\r\n",
144         file_stat.st_size,
145         (endswith(file_path, ".html") ? "text/html" :
146             "text/plain"));
147     write(fd, header, strlen(header));
148
149     ssize_t bytes_read;
150     char buffer[MAX_BUFFER];
151     while ((bytes_read = read(file_fd, buffer, sizeof(buffer)))
152         > 0)
153         write(fd, buffer, bytes_read);
154
155     close(file_fd);

```

```

148     close(fd);
149
150     log_event("Response: 200 OK");
151 }
152
153 void *worker_thread(void *arg) {
154     while (1) {
155         const int client_fd = dequeue_task();
156
157         char buffer[MAX_BUFFER];
158         const ssize_t bytes_read = read(client_fd, buffer,
159             sizeof(buffer) - 1);
160         if (bytes_read <= 0) {
161             close(client_fd);
162             continue;
163         }
164
165         buffer[bytes_read] = '\0';
166
167         char method[16], path[256];
168         sscanf(buffer, "%15s %255s", method, path);
169
170         char log_msg[LOG_BUFFER];
171         snprintf(log_msg, sizeof(log_msg), "Request: %s on %s",
172             method, path);
173         log_event(log_msg);
174
175         if (strcmp(method, "GET") != 0 && strcmp(method, "HEAD")
176             != 0) {
177             send_response(client_fd, 405, "Method Not Allowed",
178                 "text/html", "<h1>405 Method Not Allowed</h1>",
179                 31);
180             continue;
181         }
182
183         if (strlen(path) == 1 && path[0] == '/')
184             strncpy(path, "/index.html", 11);
185
186         char file_path[512];
187         snprintf(file_path, sizeof(file_path), "%s%s", ROOT_DIR,
188             path);
189         serve_static_file(client_fd, file_path);

```

```

183     }
184 }
185
186 int main() {
187     signal(SIGINT, handle_signal);
188     log_file = fopen("server.log", "a");
189     if (!log_file) {
190         perror("Failed to open log file");
191         return 1;
192     }
193
194     server_fd = socket(AF_INET, SOCK_STREAM, 0);
195     if (server_fd == -1) {
196         perror("Socket creation failed");
197         return 1;
198     }
199
200     struct sockaddr_in server_addr = {0};
201     server_addr.sin_family = AF_INET;
202     server_addr.sin_port = htons(PORT);
203     server_addr.sin_addr.s_addr = INADDR_ANY;
204
205     if (bind(server_fd, (struct sockaddr *)&server_addr,
206             sizeof(server_addr)) == -1) {
207         perror("Bind failed");
208         return 1;
209     }
210
211     if (listen(server_fd, SOMAXCONN) == -1) {
212         perror("Listen failed");
213         return 1;
214     }
215
216     char log_msg[LOG_BUFFER];
217     snprintf(log_msg, sizeof(log_msg), "Server starting on :%d",
218             PORT);
219     log_event(log_msg);
220
221     pthread_t threads[MAX_THREADS];
222     for (int i = 0; i < MAX_THREADS; ++i)
223         pthread_create(&threads[i], NULL, worker_thread, NULL);

```

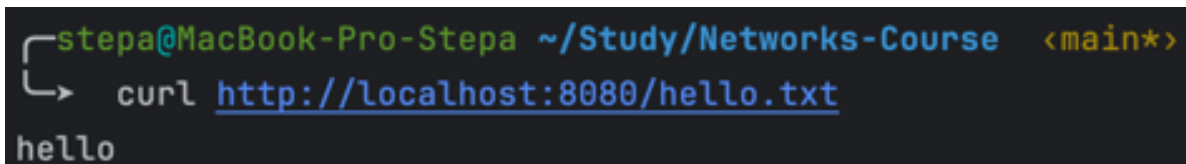
```

222
223     fd_set readfds;
224     FD_ZERO(&readfds);
225     FD_SET(server_fd, &readfds);
226     const int max_fd = server_fd;
227
228     while (1) {
229         fd_set temp_set = readfds;
230         if (pselect(max_fd + 1, &temp_set, NULL, NULL, NULL,
231                     NULL) > 0) {
232             if (FD_ISSET(server_fd, &temp_set)) {
233                 const int client_fd = accept(server_fd, NULL,
234                                             NULL);
235                 if (client_fd == -1) {
236                     perror("Accept failed");
237                     continue;
238                 }
239                 enqueue_task(client_fd);
240             }
241         }
242     }

```

### 3.4 Примеры работы программы

На рисунке 3.1 представлен пример ответа на GET-запрос.



```

stepa@MacBook-Pro-Stepa ~/Study/Networks-Course <main*>
└─ curl http://localhost:8080/hello.txt
hello

```

Рисунок 3.1 – Пример ответа на GET-запрос

На рисунке 3.2 представлен пример ответа на HEAD-запрос.

```
stepa@MacBook-Pro-Stepa ~/Study/Networks-Course <main*>  
└─> curl -I http://localhost:8080/index.html  
HTTP/1.1 200 OK  
Content-Length: 9336  
Content-Type: text/html  
Connection: close
```

Рисунок 3.2 – Пример ответа на HEAD-запрос

На рисунке 3.3 представлен пример ответа на GET-запрос несуществующего файла.

```
stepa@MacBook-Pro-Stepa ~/Study/Networks-Course <main*>  
└─> curl -I http://localhost:8080/indexxx.html  
HTTP/1.1 404 Not Found  
Content-Length: 22  
Content-Type: text/html  
Connection: close
```

Рисунок 3.3 – Пример ответа на GET-запрос несуществующего файла

На рисунке 3.4 представлен пример ответа на неразрешенный POST-запрос.

```
stepa@MacBook-Pro-Stepa ~/Study/Networks-Course <main*>  
└─> curl -X POST http://localhost:8080/1.png  
<h1>405 Method Not Allowed</h1>%
```

Рисунок 3.4 – Пример ответа на неразрешенный POST-запрос



На рисунке 3.5 представлен пример ответа на GET-запрос без прав на доступ.

```
stepa@MacBook-Pro-Stepa ~/Study/Networks-Course <main*>
└─> chmod 0 static/hello.txt

stepa@MacBook-Pro-Stepa ~/Study/Networks-Course <main*>
└─> curl http://localhost:8080/hello.txt
<h1>403 Forbidden</h1>%
```

Рисунок 3.5 – Пример ответа на GET-запрос без прав на доступ

На рисунке 3.6 представлен пример логирования событий в системе.

```
stepa@MacBook-Pro-Stepa ~/Study/Networks-Course <main*>
└─> cat server.log
[Sun Dec 8 20:39:45 2024] Server starting on :8080
[Sun Dec 8 20:39:59 2024] Request: GET on /
[Sun Dec 8 20:39:59 2024] Response: 200 OK
[Sun Dec 8 20:40:19 2024] Request: GET on /
[Sun Dec 8 20:40:19 2024] Response: 200 OK
[Sun Dec 8 20:40:23 2024] Request: GET on /hello.txt
[Sun Dec 8 20:40:23 2024] Response: 200 OK
[Sun Dec 8 20:40:41 2024] Request: GET on /hello.txt
[Sun Dec 8 20:40:41 2024] Response: 200 OK
[Sun Dec 8 20:40:49 2024] Request: GET on /hello.txt
[Sun Dec 8 20:40:49 2024] Response: 200 OK
[Sun Dec 8 20:41:23 2024] Request: HEAD on /1.png
[Sun Dec 8 20:41:23 2024] Response: 200 OK
[Sun Dec 8 20:41:32 2024] Request: HEAD on /index.html
[Sun Dec 8 20:41:32 2024] Response: 200 OK
[Sun Dec 8 20:42:03 2024] Request: HEAD on /indexxx.html
[Sun Dec 8 20:42:03 2024] Response: 404 Not Found
```

Рисунок 3.6 – Пример логирования событий в системе

## 4 Исследовательский раздел

### 4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование:

- 1) операционная система — macOS 15.1.1 (24B91) [5];
- 2) объем оперативной памяти — 18 Гбайт;
- 3) процессор — Apple M3 Pro, 11 ядер [5].

Во время тестирования ноутбук был подключен к сети электропитания и нагружен только встроенными приложениями и системой тестирования.

### 4.2 Описание исследования

В качестве стороннего веб-сервера для сравнения производительности был выбран **nginx**, так как является самым популярным и востребованным [6]. В качестве инструмента для генерации нагрузки и замеров производительности будет использоваться утилита **ab** (Apache Benchmark) [7]. Целью исследования является сравнение реализованного веб-сервера с известными аналогами по среднему времени ответа на получение файла размером 9.1 КБ в зависимости от количества запросов.

### 4.3 Результаты исследования

Результаты сравнения ве-серверов представлены в таблице 4.1.

Таблица 4.1 – Среднее время ответа на запрос для получения файла

Количество запросов	Среднее время ответа (разработанный веб- сервер), мс	Среднее время ответа (nginx), мс
100	0.209	0.739
1000	0.091	0.460
5000	0.079	0.440
10000	0.077	0.444
50000	0.077	0.444
100000	0.077	0.444

## Вывод

В исследовательском разделе было проведено сравнение реализованного статического веб-сервера и веб-сервера `nginx` по времени получения файла размером 9.1 КБ. Исходя из полученных в таблице 4.1 результатов, был сделан вывод, что время ответа для небольшого количества запросов (до 1000) примерно в 2 раза больше, чем для большого количества запросов (больше 1000). При этом разница во времени ответа между разным количеством запросов уменьшается с повышением самого числа запросов для обоих серверов. Реализованный веб-сервер в среднем превосходит `nginx` по скорости ответа примерно в 3.5 раза до 1000 запросов и примерно в 6 раз для количества запросов, превышающего 1000.

## ЗАКЛЮЧЕНИЕ

Цель работы, заключающаяся в разработке сервера для отдачи статического содержимого с диска, была достигнута.

Были решены следующие задачи:

- 1) описана предметная область;
- 2) спроектирована схема алгоритма работы сервера;
- 3) выбраны средства реализации сервера;
- 4) проведено сравнение реализованного сервера с известными аналогами.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Yibei Ling, Tracy Mullen, Xiaola Lin*. Analysis of Optimal Thread Pool Size. — 2000.
2. MAN pselect. — [Электронный ресурс]. — Режим доступа: <https://www.opennet.ru/man.shtml?topic=pselect&category=2&russian=0> (дата обращения: 23.11.24).
3. C Language Reference. — [Электронный ресурс]. — Режим доступа: <https://learn.microsoft.com/en-us/cpp/c-language/c-language-reference?view=msvc-170> (дата обращения: 24.11.24).
4. A cross-platform IDE for C and C++. — [Электронный ресурс]. — Режим доступа: <https://www.jetbrains.com/clion/> (дата обращения: 24.11.24).
5. MacOS. — [Электронный ресурс]. — Режим доступа: <https://www.apple.com/macos/macos-sequoia/> (дата обращения: 25.11.24).
6. NGINX. — [Электронный ресурс]. — Режим доступа: <https://nginx.org/ru/> (дата обращения: 26.11.24).
7. Apache Benchmark. — [Электронный ресурс]. — Режим доступа: <https://httpd.apache.org/docs/current/programs/ab.html> (дата обращения: 26.11.24).