



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. Н.
(Фамилия И. О.)

2024 г.

РЕФЕРАТ

Расчетно-пояснительная записка 20 с., 9 рис., 1 табл., 1 источн., 1 прил.
БЕБ-СЕРВЕР, THREAD-POOL, PSELECT, NGINX, C.

Цель работы — разработка сервера для отдачи статического содержимого с диска. Архитектура сервера должна быть основана на пуле потоков (thread-pool) совместно с `pselect()`.

В результате работы был проведен анализ предметной области, спроектирована схема алгоритма работы веб-сервера. Разработан статический веб-сервер, поддерживающий обработку запросов на получение различных типов файлов. Проведено сравнение реализованного веб-сервера с существующими аналогами.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	5
1 Аналитический раздел	6
1.1 Архитектура thread-pool	6
1.2 Системный вызов pselect	7
2 Конструкторский раздел	9
2.1 Схема алгоритма работы сервера	9
3 Технологический раздел	10
3.1 Требования к разрабатываемой программе	10
3.2 Реализация веб-сервера	11
3.3 Примеры работы программы	13
4 Исследовательский раздел	16
4.1 Технические характеристики и описание исследования	16
4.2 Результаты исследования	17
ЗАКЛЮЧЕНИЕ	18
ПРИЛОЖЕНИЕ А	19
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	20

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 с `pselect()`.

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 [**pselect**].

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

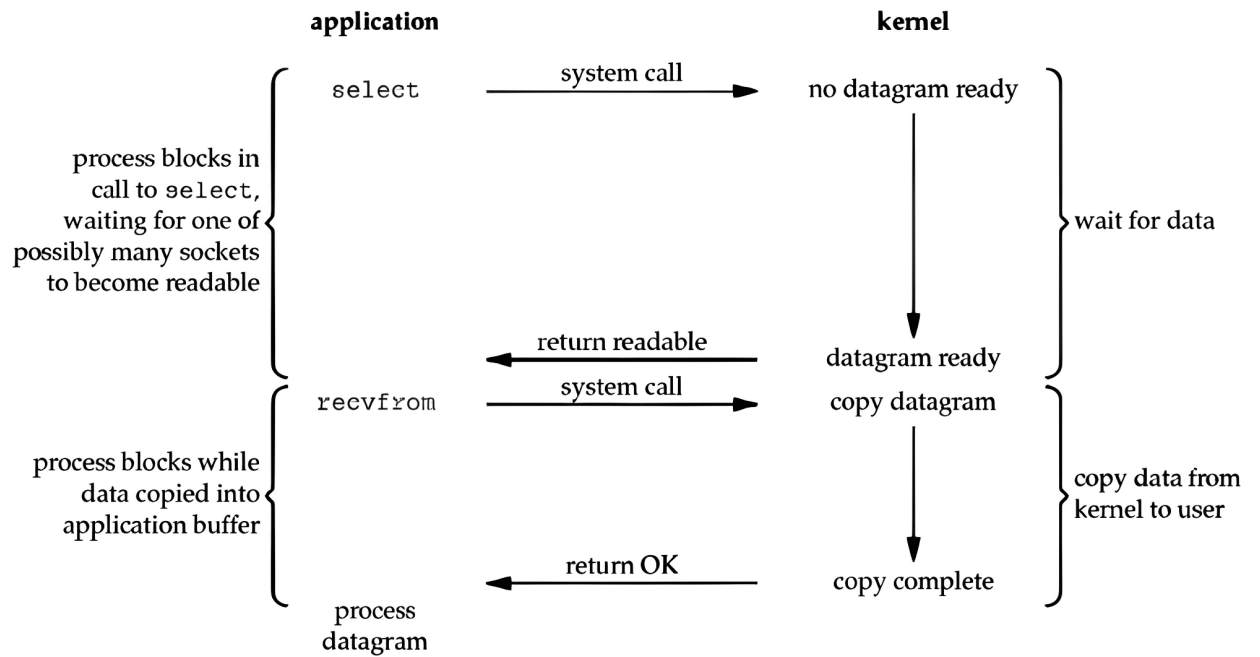


Рисунок 1.2 – Модель неблокирующего синхронного ввода-вывода

2 Конструкторский раздел

2.1 Схема алгоритма работы сервера

На рисунке 2.1 представлена схема алгоритма работы сервера.

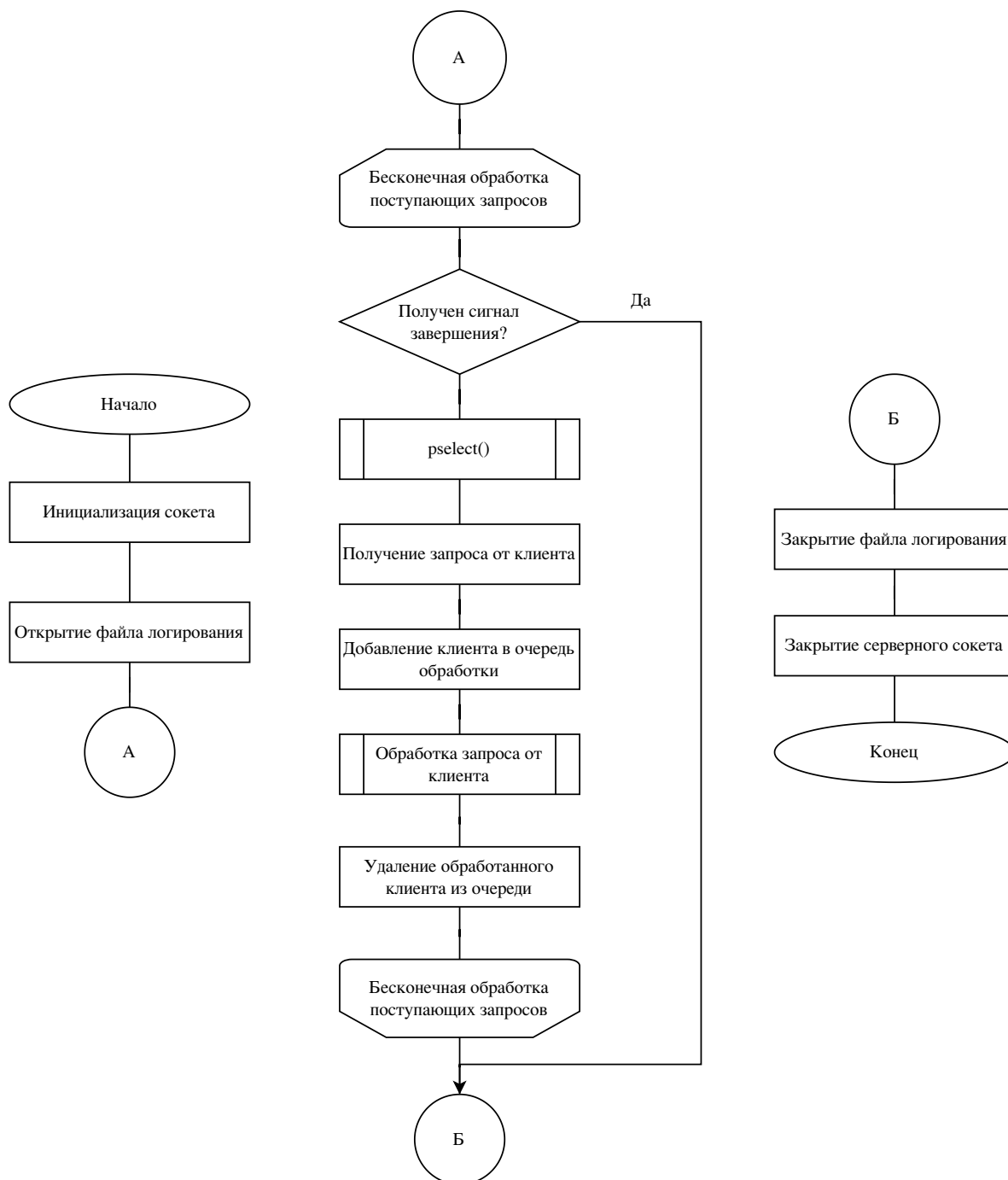


Рисунок 2.1 – Схема алгоритма работы сервера

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.

В качестве языка программирования для реализации веб-сервера был выбран язык C. В качестве среды для разработки была выбрана среда CLion [с; clion].

3.2 Реализация веб-сервера

Реализация функции логирования представлена в листинге 3.1.

Листинг 3.1 – Реализация функции логирования

```
void log_event(const char *message) {
    pthread_mutex_lock(&queue_mutex);
    if (log_file) {
        const time_t now = time(NULL);
        fprintf(log_file, "[%s] %s\n", strtok(ctime(&now),
            "\n"), message);
        fflush(log_file);
    }
    pthread_mutex_unlock(&queue_mutex);
}
```

Реализация функции добавления задачи в очередь представлена в листинге 3.2.

Листинг 3.2 – Реализация функции добавления задачи в очередь

```
void enqueue_task(const int fd) {
    pthread_mutex_lock(&queue_mutex);
    if (queue_count == QUEUE_SIZE) {
        pthread_mutex_unlock(&queue_mutex);
        close(fd);
        return;
    }

    task_queue[queue_end] = fd;
    queue_end = (queue_end + 1) % QUEUE_SIZE;
    queue_count++;
    pthread_cond_signal(&queue_cond);
    pthread_mutex_unlock(&queue_mutex);
}
```

Реализация функции удаления задачи из очереди представлена в листинге 3.3.

Листинг 3.3 – Реализация функции удаления задачи в очередь

```
int dequeue_task() {
    pthread_mutex_lock(&queue_mutex);
    while (queue_count == 0)
        pthread_cond_wait(&queue_cond, &queue_mutex);

    const int fd = task_queue[queue_start];
    queue_start = (queue_start + 1) % QUEUE_SIZE;
    --queue_count;
    pthread_mutex_unlock(&queue_mutex);

    return fd;
}
```

Реализация функции отправки ответа представлена в листинге 3.4.

Листинг 3.4 – Реализация функции отправки ответа

```
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) {
    char header[MAX_BUFFER];
    const int header_length = snprintf(header, MAX_BUFFER,
        "HTTP/1.1 %d %s\r\n"
        "Content-Length: %zu\r\n"
        "Content-Type: %s\r\n"
        "Connection: close\r\n\r\n",
        status, status_text, body_length, content_type);

    write(fd, header, header_length);
    if (body && body_length > 0)
        write(fd, body, body_length);
    close(fd);

    char log_msg[LOG_BUFFER];
    snprintf(log_msg, sizeof(log_msg), "Response: %d %s",
        status, status_text);
    log_event(log_msg);
}
```

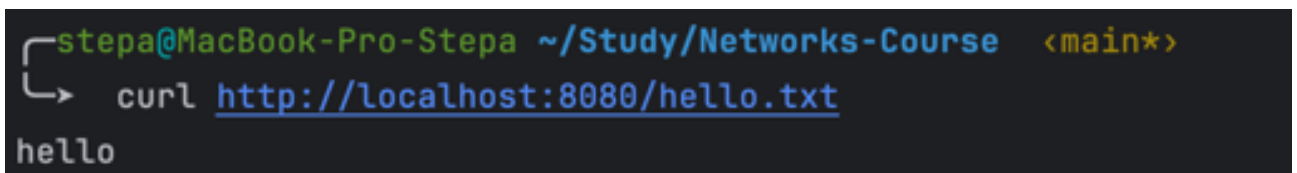
Реализация цикла обработки клиентов представлена в листинге 3.5.

Листинг 3.5 – Реализация цикла обработки клиентов

```
while (1) {
    fd_set temp_set = readfds;
    if (pselect(max_fd + 1, &temp_set, NULL, NULL, NULL, NULL) >
        0)
        if (FD_ISSET(server_fd, &temp_set)) {
            const int client_fd = accept(server_fd, NULL, NULL);
            if (client_fd == -1) {
                perror("Accept failed");
                continue;
            }
            enqueue_task(client_fd);
        }
}
```

3.3 Примеры работы программы

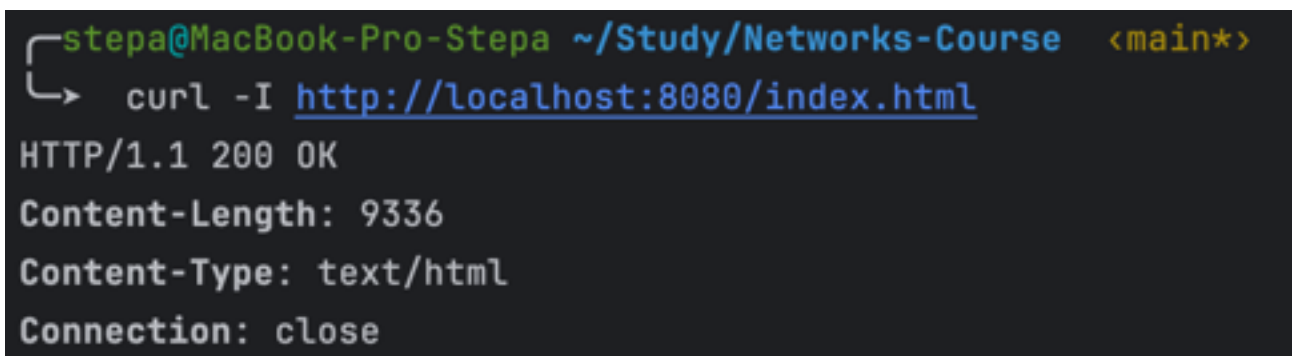
На рисунке 3.1 представлен пример ответа на GET-запрос.



```
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 для HTML файла.



```
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 и описание исследования

Технические характеристики устройства, на котором выполнялось тестирование:

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

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

В качестве стороннего веб-сервера для сравнения производительности был выбран **nginx**, так как является самым популярным и востребованным. В качестве инструмента для генерации нагрузки и замеров производительности будет использоваться утилита **ab** (Apache Benchmark). 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 [**nginx**; **ab**].

4.2 Результаты исследования

Результаты сравнения веб-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 с использованием архитектуры, основанной на пуле потоков (thread-pool) совместно с `pselect()`, 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.

ПРИЛОЖЕНИЕ А

Презентация к курсовой работе состоит из 9 слайдов

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Yibei Ling, Tracy Mullen, Xiaola Lin*. Analysis of Optimal Thread Pool Size. — 2000.