



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 ИУ7-73Б
(Группа)

(Подпись, дата)

Шишмир Э. О.
(И. О. Фамилия)

Руководитель курсовой работы

(Подпись, дата)

Клочков М. Н.
(И. О. Фамилия)

2024 г.

РЕФЕРАТ

Расчетно-пояснительная записка 21 с., 3 рис., 1 табл., 8 источн., 4 прил.
ВЕБ-СЕРВЕР, PREFORK, EPOLL, NGINX, C.

Цель работы — разработка сервера для отдачи статического содержимого с диска. Архитектура сервера должна быть основана на предварительном создании процессов (prefork) совместно с `epoll`.

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 известными аналогами.

СОДЕРЖАНИЕ

РЕФЕРАТ	3
ВВЕДЕНИЕ	5
1 Аналитический раздел	6
1.1 Архитектура prefork	6
1.2 Epoll	7
2 Конструкторский раздел	8
2.1 Схема алгоритма работы сервера	8
3 Технологический раздел	10
3.1 Требования к разрабатываемой программе	10
3.2 Реализация веб-сервера	11
3.3 Тестирование программного обеспечения	15
4 Исследовательский раздел	17
4.1 Технические характеристики и описание исследования	17
4.2 Результаты исследования	18
ЗАКЛЮЧЕНИЕ	19
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	20
ПРИЛОЖЕНИЕ А	21

ВВЕДЕНИЕ

Статический веб-сервер — сервер, который передает статические файлы в ответ на запросы от клиента. Статические файлы — файлы, содержимое которых не изменяется динамически на сервере. Такой сервер предназначен для чтения файлов с диска и их отправки клиенту, обычно через протокол HTTP.

Целью курсовой работы является разработка сервера для отдачи статического содержимого с диска. Архитектура сервера должна быть основана на предварительном создании процессов (prefork) совместно с epoll.

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 prefork

Архитектура prefork — модель многозадачности, при которой для обработки входящих запросов заранее создается множество дочерних процессов. В отличие от моделей с использованием потоков, prefork использует процессы, что исключает проблемы, связанные с синхронизацией потоков в рамках одного процесса.

Эта архитектура широко используется в веб-серверах, таких как Apache HTTP Server [1], где она обеспечивает изоляцию запросов и повышенную стабильность за счет независимости процессов друг от друга. Каждый процесс запускается заранее и ждет подключения клиента, что снижает накладные расходы на создание новых процессов при каждом запросе [2].

Основными компонентами архитектуры prefork являются:

- 1) пул процессов;
- 2) менеджер процессов.

Менеджер процессов создает множество дочерних процессов (пул процессов) и занимается мониторингом, для их восстановления при независимости.

1.2 Epoll

Epoll — это механизм ввода-вывода в Linux, предназначенный для эффективного управления большим количеством файловых дескрипторов. Он предоставляет интерфейс для мониторинга множества событий на различных файловых дескрипторах и позволяет эффективно реагировать на изменения их состояния [3].

Epoll реализует модель асинхронного ввода-вывода с использованием многозадачности на основе событий. Эта модель позволяет серверу эффективно отслеживать состояния множества файловых дескрипторов без блокировки потока выполнения.

Модель асинхронного ввода-вывода представлена на рисунке 1.1.

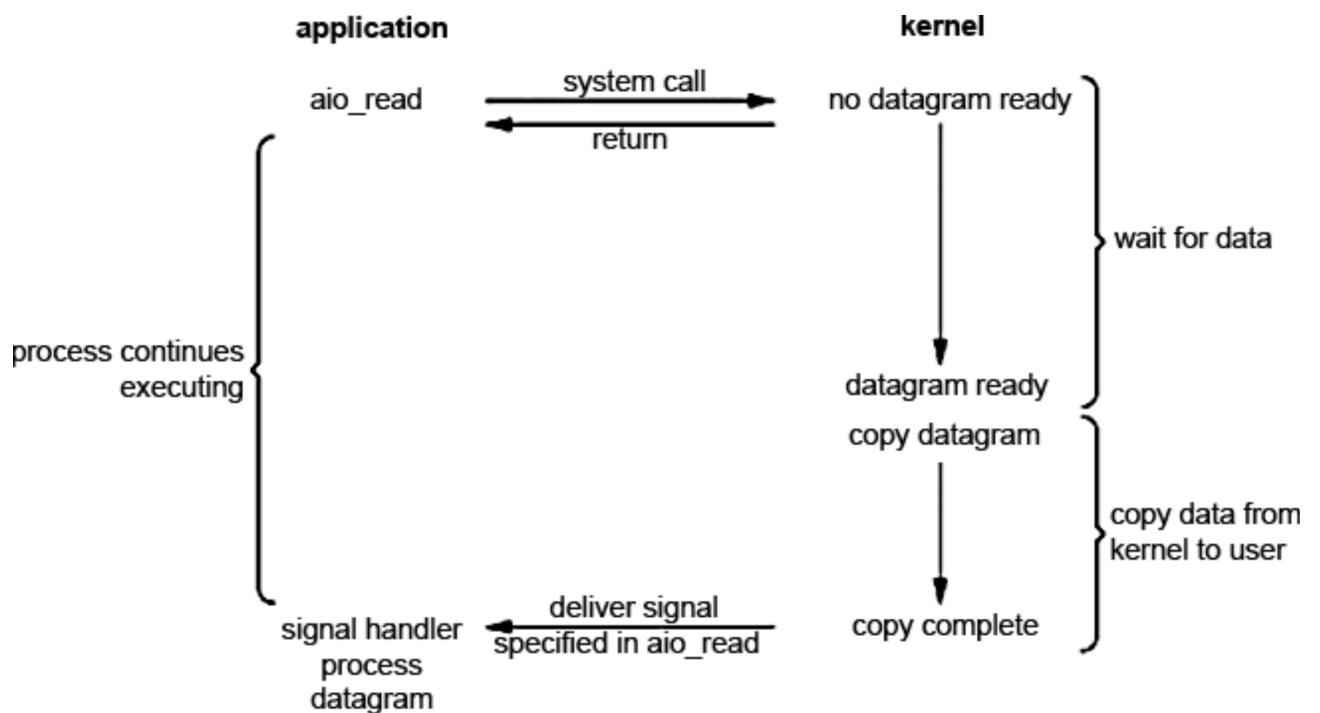


Рисунок 1.1 – Модель асинхронного ввода-вывода

2 Конструкторский раздел

2.1 Схема алгоритма работы сервера

На рисунке 2.1 представлена схема алгоритма работы основного процесса сервера.

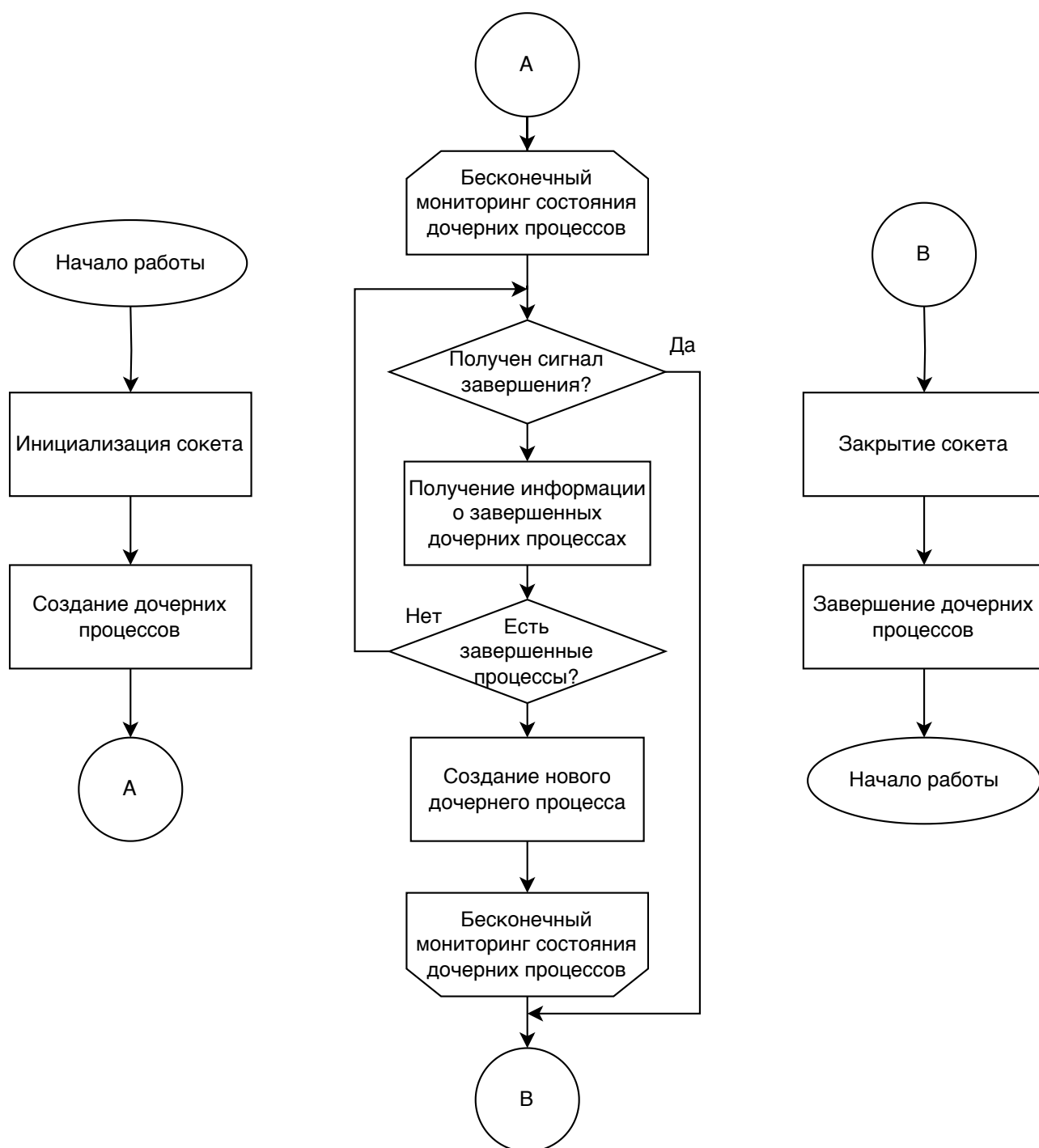


Рисунок 2.1 – Схема алгоритма работы основного процесса сервера

На рисунке 2.2 представлена схема алгоритма работы дочерних процессов сервера.

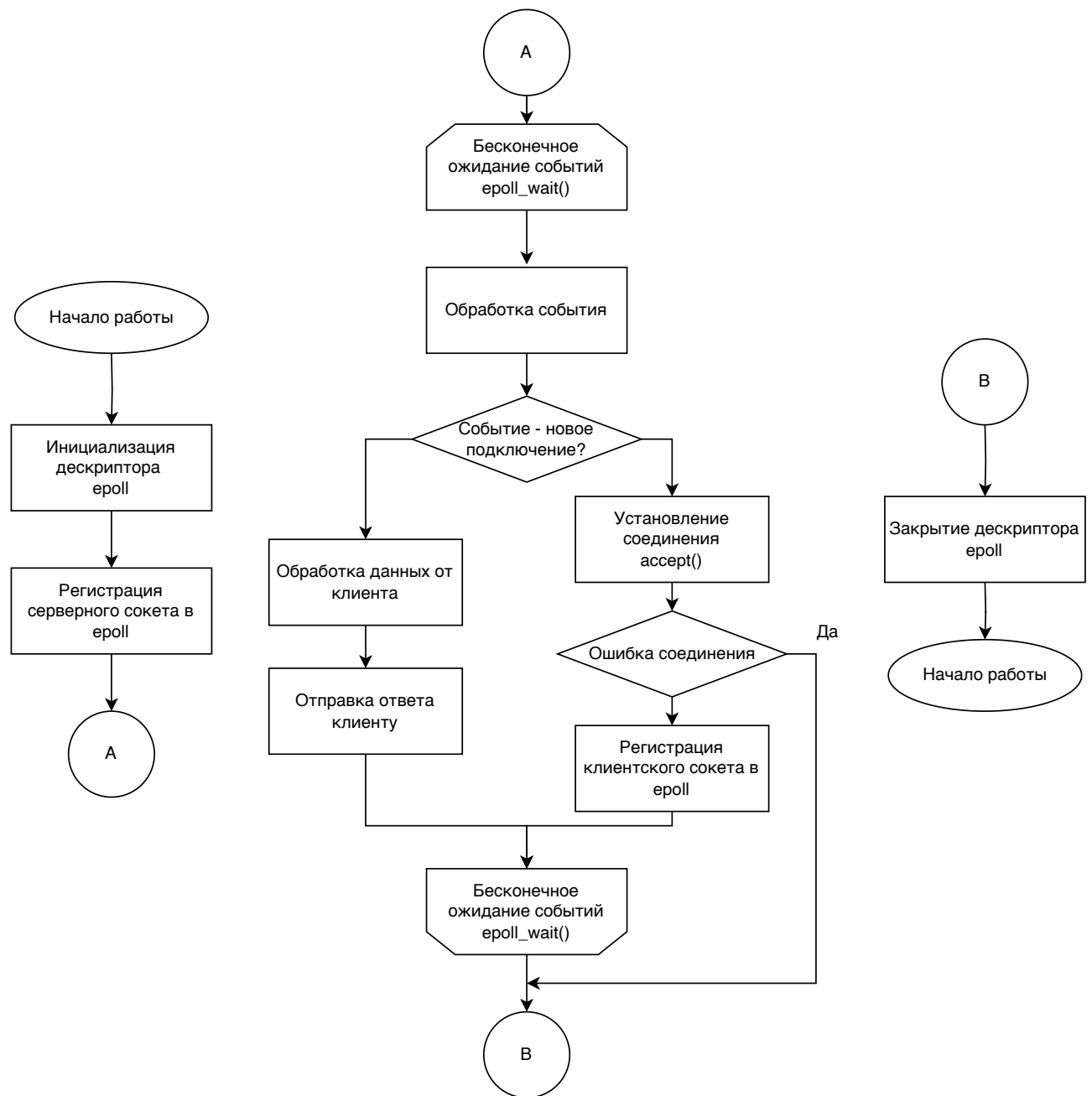


Рисунок 2.2 – Схема алгоритма работы дочерних процессов сервера

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 [4].

3.2 Реализация веб-сервера

Реализация функции инициализации серверного сокета представлена в листинге 3.1.

Листинг 3.1 – Реализация функции инициализации серверного сокета

```
int start_server(server_t *server) {
    server->server_fd = socket(AF_INET, SOCK_STREAM, 0);
    if (server->server_fd == -1) {
        perror("Socket creation failed");
        log_msg(ERROR, "Socket creation failed");
        return -1;
    }
    int opt = 1;
    setsockopt(server->server_fd, SOL_SOCKET, SO_REUSEADDR,
        &opt, sizeof(opt));
    struct sockaddr_in server_addr = {0};
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(PORT);
    server_addr.sin_addr.s_addr = INADDR_ANY;
    if (bind(server->server_fd, (struct sockaddr *)&server_addr,
        sizeof(server_addr)) == -1) {
        perror("Bind failed");
        log_msg(ERROR, "Bind failed");
        return -1;
    }
    if (listen(server->server_fd, SOMAXCONN) == -1) {
        perror("Listen failed");
        log_msg(ERROR, "Listen failed");
        return -1;
    }
    if (set_nonblocking(server->server_fd) == -1) {
        perror("Failed to set server socket to non-blocking");
        log_msg(ERROR, "Failed to set server socket to
            non-blocking");
        return -1;
    }
    log_msg(INFO, "Server started");
    return 0;
}
```

Реализация функции логирования представлена в листинге 3.2.

Листинг 3.2 – Реализация функции логирования

```
static void logger(const char *level, const char *message) {
    if (!LOGGER_ON) {
        return;
    }
    const time_t now = time(NULL);
    FILE *log_file = fopen(LOGGER_FILE, "a");
    if (log_file != NULL) {
        fprintf(log_file, "[%s] [%s] %s\n", strtok(ctime(&now),
            "\n"), level, message);
        fclose(log_file);
    } else {
        perror("Failed to open logger file");
    }
}
```

Реализация создания дочерних процессов представлена в листинге 3.3.

Листинг 3.3 – Реализация создания дочерних процессов

```
int create_prefork(server_t *server) {
    for (int i = 0; i < MAX_PROCESSES; ++i) {
        pid_t pid = fork();
        if (pid == 0) {
            prefork_process(server);
            exit(0);
        } else if (pid > 0) {
            server->child_pids[i] = pid;
        } else {
            perror("Fork failed");
            log_msg(ERROR, "Fork failed");
            return -1;
        }
    }
    return 0;
}
```

Реализация функции мониторинга дочерних процессов представлена в листинге 3.4.

Листинг 3.4 – Реализация функции мониторинга дочерних процессов

```
int monitor_prefork(server_t *server) {
    signal(SIGINT, handle_signal);
    while (!stop_requested) {
        int status;
        pid_t pid = waitpid(-1, &status, WNOHANG);
        if (pid == 0) {
            continue;
        }
        if (pid == -1 && errno == EINTR) {
            continue;
        }
        if (pid == -1) {
            perror("waitpid");
            return -1;
        }
        for (int i = 0; i < MAX_PROCESSES; ++i) {
            if (server->child_pids[i] == pid) {
                log_msg_int(INFO, "Child process pid: %d exited,
                    restarting...", pid);
                pid_t new_pid = fork();
                if (new_pid == 0) {
                    prefork_process(server);
                    exit(0);
                } else if (new_pid > 0) {
                    server->child_pids[i] = new_pid;
                    log_msg_int(INFO, "Child process restarted
                        with pid: %d", new_pid);
                } else {
                    log_msg(ERROR, "Failed to restart child
                        process");
                }
            }
            break;
        }
    }
    log_msg(INFO, "Monitoring process terminated.");
    return 0;
}
```

Реализация цикла обработки клиентов представлена в листинге 3.5.

Листинг 3.5 – Реализация цикла обработки клиентов

```
while (1) {
    int event_count = epoll_wait(epoll_fd, events, MAX_EVENTS,
        -1);
    for (int i = 0; i < event_count; ++i) {
        if (events[i].data.fd == server->server_fd) {
            struct sockaddr_in client_addr;
            socklen_t client_len = sizeof(client_addr);
            int client_fd = accept(server->server_fd, (struct
                sockaddr *)&client_addr, &client_len);
            if (client_fd == -1 && errno == EAGAIN) {
                continue;
            }
            if (client_fd == -1) {
                perror("Failed accept");
                log_msg(ERROR, "Failed accept");
                close(epoll_fd);
                exit(1);
            }
            set_nonblocking(client_fd);
            log_msg(INFO, "New client connection accepted");
            struct epoll_event client_event = {0};
            client_event.events = EPOLLIN | EPOLLET;
            client_event.data.fd = client_fd;
            if (epoll_ctl(epoll_fd, EPOLL_CTL_ADD, client_fd,
                &client_event) == -1) {
                perror("Failed to add client_fd to epoll");
                log_msg(ERROR, "Failed to add client_fd to
                    epoll");
                close(client_fd);
            }
        } else {
            int client_fd = events[i].data.fd;
            server->handler(client_fd);
        }
    }
}
```

3.3 Тестирование программного обеспечения

В листинге 3.6 представлен ответ на GET-запрос.

Листинг 3.6 – Ответ на GET-запрос

```
curl http://localhost:80/test.txt
hello world
```

В листинге 3.7 представлен ответ на HEAD-запрос для HTML файла.

Листинг 3.7 – Ответ на HEAD-запрос

```
curl -I http://localhost:80/index.html
HTTP/1.1 200 OK
Content-Length: 1415
Content-Type: text/html
Connection: close
```

В листинге 3.8 представлен ответ на запрос несуществующего файла.

Листинг 3.8 – Ответ на GET-запрос несуществующего файла

```
curl http://localhost:80/index.html1
<h1>404 Not Found</h1>
```

В листинге 3.9 представлен ответ на неразрешенный POST-запрос.

Листинг 3.9 – Ответ на неразрешенный POST-запрос

```
curl -X POST http://localhost:80/index.html
<h1>405 Method Not Allowed</h1>
```

В листинге 3.10 представлен ответ на GET-запрос без прав на доступ.

Листинг 3.10 – Ответ на GET-запрос без прав на доступ

```
curl http://localhost:80/forbidden.txt
<h1>403 Forbidden</h1>
```

В листинге 3.11 представлено логирование событий в системе.

Листинг 3.11 – Логирование событий в системе

```
[Sun Dec 22 21:56:21 2024] [INFO] Server started
[Sun Dec 22 21:56:27 2024] [INFO] New client connection accepted
[Sun Dec 22 21:56:27 2024] [INFO] Handling request for file
[Sun Dec 22 21:56:27 2024] [INFO] Sent response ok
[Sun Dec 22 21:56:56 2024] [INFO] Sent response: 404 Not Found
[Sun Dec 22 21:57:18 2024] [INFO] New client connection accepted
[Sun Dec 22 21:57:18 2024] [INFO] Sent response: 405 Method Not
    Allowed
[Sun Dec 22 22:00:33 2024] [INFO] Monitoring process terminated.
[Sun Dec 22 22:00:33 2024] [INFO] Server socket closed
[Sun Dec 22 22:00:33 2024] [INFO] Terminating child process with
    PID: 98814
[Sun Dec 22 22:00:33 2024] [INFO] Terminating child process with
    PID: 98815
[Sun Dec 22 22:00:33 2024] [INFO] Terminating child process with
    PID: 98816
[Sun Dec 22 22:00:33 2024] [INFO] Terminating child process with
    PID: 98817
[Sun Dec 22 22:00:33 2024] [INFO] child process with PID: 98814
    terminated
[Sun Dec 22 22:00:33 2024] [INFO] child process with PID: 98815
    terminated
[Sun Dec 22 22:00:33 2024] [INFO] child process with PID: 98816
    terminated
[Sun Dec 22 22:00:33 2024] [INFO] child process with PID: 98817
    terminated
[Sun Dec 22 22:00:33 2024] [INFO] Server shutting down...
```

4 Исследовательский раздел

4.1 Технические характеристики и описание исследования

Технические характеристики устройства, на котором выполнялось тестирование:

- 1) операционная система — Linux ubuntuserver 5.15.0-107-generic [5];
- 2) объем оперативной памяти — 8 Гбайт;
- 3) процессор — Intel Core I5, 4 ядра [6].

Во время проведения экспериментов ноутбук находился в режиме работы от сети и был задействован исключительно встроенными приложениями и программой тестирования.

Для сравнения производительности был выбран веб-сервер `nginx` [7], так как он считается одним из наиболее популярных и широко используемых решений. В качестве инструмента для создания нагрузки и измерения времени отклика применялся инструмент `ab` (Apache Benchmark) [8]. Основная задача исследования заключалась в сравнении производительности разработанного веб-сервера и известных аналогов на примере обработки запроса для получения файла размером 12.7 КБ при различном количестве запросов.

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
50	0.967	1.213
100	1.254	1.678
1000	1.345	2.845
5000	1.743	4.315
10000	2.077	5.421
50000	2.854	7.933
100000	3.014	9.102

Вывод

В исследовательском разделе было проведено сравнение реализованного статического веб-сервера и веб-сервера **nginx** по времени получения файла размером 12.7 КБ. Из таблицы 4.1 видно, что с увеличением количества запросов среднее время отклика для обоих серверов увеличивается, однако для разработанного веб-сервера это увеличение менее выражено, чем для **nginx**. При сравнении двух серверов, можно заметить, что разработанный веб-сервер стабильно показывает меньшее время обработки, с разницей в среднем около 0.4 миллисекунд на меньших объемах нагрузки (до 1000 запросов), и эта разница увеличивается до 2–2.5 миллисекунд для большего количества запросов (более 1000). В целом, разработанный веб-сервер демонстрирует более высокую производительность по сравнению с **nginx**, особенно при высокой нагрузке, где разница в времени отклика может достигать 3 раз.

ЗАКЛЮЧЕНИЕ

В ходе выполнения работы поставленная цель была достигнута: разработан сервер для отдачи статического содержимого с диска. Архитектура сервера основана на предварительном создании процессов (prefork) совместно с epoll.

В ходе выполнения работы были решены все 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. Apache HTTP Server. — [Электронный ресурс]. — Режим доступа: <https://httpd.apache.org/> (дата обращения: 2024-12-23).
2. Apache MPM prefork. — [Электронный ресурс]. — Режим доступа: <https://httpd.apache.org/docs/current/mod/prefork.html> (дата обращения: 10.12.24).
3. MAN epoll. — [Электронный ресурс]. — Режим доступа: <https://www.opennet.ru/man.shtml?topic=epoll> (дата обращения: 10.12.24).
4. *Ritchie D. M.* The C Programming Language. — 2nd. — Prentice Hall, 1978.
5. Ubuntu Linux. — [Электронный ресурс]. — Режим доступа: <https://ubuntu.com/> (дата обращения: 2024-12-23).
6. Intel® Processor Identification Utility. — [Электронный ресурс]. — Режим доступа: <https://www.intel.com/content/www/us/en/support/articles/000005657/processors.html> (дата обращения: 2024-12-23).
7. Nginx. — [Электронный ресурс]. — Режим доступа: <https://nginx.org/> (дата обращения: 2024-12-23).
8. Apache Benchmark (ab). — [Электронный ресурс]. — Режим доступа: <https://httpd.apache.org/docs/2.4/programs/ab.html> (дата обращения: 2024-12-23).

ПРИЛОЖЕНИЕ А

Презентация к курсовой работе

Презентация состоит из 9 слайдов.