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

ФАКУЛЬТЕТ ИУ «Информатика и системы управления»

КАФЕДРА ИУ-7 «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

«Разработка статического HTTP сервера»

Студент группы ИУ7-73Б _____ Паламарчук А.Н.
(Подпись, дата) (Фамилия И.О.)

Руководитель _____ Клочков М.Н.
(Подпись, дата) (Фамилия И.О.)

2025 г.

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

УТВЕРЖДАЮ
Заведующий кафедрой ИУ7
_____ И. В. Рудаков
«__» сентября 2025 г.

З А Д А Н И Е
на выполнение курсовой работы

по дисциплине **«Компьютерные сети»**

Студент группы **ИУ7-73Б Паламарчук Андрей Николаевич**

Тема курсовой работы **Разработка статического HTTP сервера**

Направленность КР (учебная, исследовательская, практическая, др.) **учебная**

Источник тематики (кафедра, предприятие, НИР) **кафедра**

График выполнения КР: 25% к 5 нед., 50% к 8 нед., 75% к 11 нед., 100% к 15 нед.

Задание:

Разработать сервер для отдачи статического содержимого с диска по протоколу HTTP. Предусмотреть поддержку запросов GET и HEAD, поддержку статусов 200, 403, 404. Предусмотреть возможность ответа сервера на неподдерживаемые запросы статусом 405. Обеспечить корректную передачу файлов размером до 128 Мбайт. Реализовать мультиплексирование - каждый процесс или поток должен отдавать данные по нескольким сетевым соединениям. Сервер по умолчанию должен возвращать HTML-страницу на выбранную тему с CSS-стилем. Реализовать запись информации о событиях в журнал. Учесть минимальные требования к безопасности серверов статического содержимого. Вариант архитектуры разрабатываемого сервера: prefork + pselect().

Оформление курсовой работы:

Расчетно-пояснительная записка объемом 12-32 страницы формата А4. Презентация к курсовой работе объемом 8-16 слайдов.

Дата выдачи задания «22» октября 2025 г.

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

Студент _____ Паламарчук А.Н.
(Подпись, дата) (Фамилия И.О.)

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В соответствии с заданием на КР необходимо разработать сервер для отдачи статического содержимого с диска по протоколу HTTP.

Для решения поставленной задачи необходимо:

—проанализировать протокол HTTP, и мультиплексирование pselect;

—реализовать поддержку HTTP-запросов методов GET и HEAD, а также обработку ошибок;

—реализовать мультиплексирование;

—реализовать журналирование и обеспечить корректную передачу файлов заданного размера;

—провести нагрузочное тестирование разработанного сервера.

1. Аналитическая часть

1.1. Протокол HTTP

Протокол HTTP (HyperText Transfer Protocol) является протоколом прикладного уровня, используемым для передачи гипертекстовых документов и произвольных данных в распределенных системах [1]. В основе работы протокола лежит модель «клиент-сервер», где клиент инициирует соединение и отправляет запрос, а сервер обрабатывает его и возвращает ответ.

Для реализации статического сервера необходимо обеспечить поддержку базовых механизмов протокола HTTP:

—метод GET — основной метод для запроса содержимого указанного ресурса;

—метод HEAD — аналогичен GET, но сервер возвращает только заголовки ответа без тела сообщения (используется для получения метаданных);

—обработку кодов состояния — для информирования клиента о результатах.

Для информирования клиента о результате обработки запроса сервер формирует ответ с соответствующим кодом состояния:

—200 OK — успешная обработка запроса и передача запрашиваемого ресурса;

—403 Forbidden — отказ в доступе к ресурсу по причинам безопасности;

—404 Not Found — запрашиваемый ресурс не найден на сервере;

—405 Method Not Allowed — запрошенный метод не поддерживается сервером.

1.2. Архитектуры серверов

Для обеспечения одновременной обработки множества клиентских запросов существуют различные архитектурные паттерны. Выбор архитектуры напрямую влияет на производительность, масштабируемость и надежность серверного программного обеспечения.

Наиболее распространенными подходами являются:

—последовательная обработка — один процесс обрабатывает один запрос за раз;

—многопоточная/многопроцессная (Thread/Process per client) — на каждого клиента создается отдельный поток или процесс (высокие накладные расходы на переключение контекста);

—пул потоков/процессов (Prefork) — предварительное создание фиксированного набора обработчиков.

В курсовой работе выбрана архитектура prefork, которая подразумевает, что родительский процесс при запуске создает фиксированный пул дочерних процессов, используя системный вызов fork(). Все дочерние процессы наследуют слушающий сокет и конкурируют за принятие входящих соединений [5]. Сбой одного дочернего процесса не обрушивает весь сервер и позволяет утилизировать ресурсы многоядерных процессоров без накладных расходов на создание процессов «на лету».

1.3. Мультиплексирование ввода-вывода

Механизмы мультиплексирования ввода-вывода позволяют одному процессу отслеживать состояние множества файловых дескрипторов (сокетов) одновременно, не блокируясь на операциях чтения или записи. В операционных системах семейства UNIX/Linux существуют системные вызовы select, poll, epoll и pselect. В курсовой работе используется системный вызов pselect.

Функция pselect является расширенной версией классического select. Особенностью является возможность атомарного управления маской сигналов во время ожидания событий на дескрипторах [3]. Это устраняет состояние гонки, которое может возникнуть, если сигнал придет между проверкой флагов и вызовом функции ожидания.

2. Конструкторская часть

2.1. Проектирование архитектуры сервера

HTTP-сервер для отдачи статического контента реализует prefork для использования многоядерности и мультиплексирование ввода-вывода для конкурентной обработки соединений внутри одного процесса.

Главный процесс отвечает за инициализацию слушающего сокета, настройку окружения, создание пула рабочих процессов и обработку сигналов завершения.

Дочерние процессы (Workers): выполняют непосредственное взаимодействие с клиентами. Каждый процесс работает в собственном адресном пространстве и обслуживает множество соединений.

Сетевое взаимодействие: реализует неблокирующий ввод-вывод и мультиплексирование через pselect().

2.2. Взаимодействие процессов

При запуске сервер создает слушающий сокет и переводит его в неблокирующий режим. Далее, используя системный вызов fork(), создается заданное количество дочерних процессов.

Каждый рабочий процесс наследует файловый дескриптор слушающего сокета. Дочерние процессы входят в бесконечный цикл обработки событий, где конкурируют за новые соединения. Балансировка нагрузки между процессами осуществляется средствами операционной системы. Падение одного дочернего процесса не приводит к остановке всего сервера.

2.3. Обработка сетевых соединений

Внутри каждого рабочего процесса реализован событийный цикл на основе системного вызова pselect(). Сервер поддерживает таблицу активных соединений, где для каждого клиента хранится его текущее состояние:

—READING_REQUEST — накопление данных из сокета до получения полных заголовков;

—SENDING_HEADERS — формирование и отправка HTTP-заголовков ответа;

—SENDING_BODY — потоковая отправка содержимого файла;

—CLOSING — завершение соединения.

2.4. Алгоритм передачи статического содержимого

Для работы с файлами большого объема (до 128 Мбайт и выше) используется механизм потоковой передачи. Файл не загружается в оперативную память целиком:

—файл открывается только на чтение (O_RDONLY);

—данные считываются в буфер меньшего размера;

—буфер отправляется в сокет клиента;

—если сокет не готов принять данные (возвращена ошибка EAGAIN или EWOULDBLOCK), передача приостанавливается, текущее смещение в файле сохраняется, а дескриптор сокета добавляется в набор отслеживания pselect на запись.

3. Технологический раздел

3.1. Средства реализации

В качестве языка программирования был выбран язык C++ (стандарт C++17). Выбор обусловлен необходимостью прямого доступа к системным вызовам POSIX. В работе использованы следующие технологии и библиотеки:

—BSD Sockets — для организации сетевого взаимодействия;

—системные вызовы — fork, pselect, accept, recv, send, sigaction [4].

В качестве компилятора был выбран g++, в качестве среды разработки был выбран редактор VS Code.

3.2. Реализация

3.2.1. Prefork

При запуске сервер создает пул рабочих процессов. Это выполняется в функции run_server с использованием системного вызова fork().

Листинг 3.1 — Создание пула процессов

```
for (int i = 0; i < cfg.workers; ++i) {

    pid_t pid = fork();

    if (pid < 0) {

        perror("fork");

        return 1;

    } else if (pid == 0) {

        run_worker(listen_fd, cfg);

        _exit(0);

    }

}
```

3.2.2. Мультиплексирование

Каждый рабочий процесс использует pselect для ожидания событий на множестве дескрипторов, что позволяет обрабатывать новые подключения и данные от уже подключенных клиентов в одном потоке.

Листинг 3.2 — pselect

```
sigset_t empty_mask;
sigemptyset(&empty_mask);


int ready = pselect(maxfd + 1,
                    &readfds, &writefds, nullptr,
                    &timeout, &empty_mask);
```

3.2.3. Обработка сигналов

Для корректного завершения работы используется обработка сигналов SIGINT и SIGTERM. Изменение флага позволяет процессам корректно закрыть сокеты и завершиться.

Листинг 3.3 — Обработчик сигналов

```
static volatile sig_atomic_t server_running = 1;
static void handle_signal(int sig) {
    server_running = 0;
}
// Установка обработчика в run_server
struct sigaction sa{};
sa.sa_handler = handle_signal;
sigemptyset(&sa.sa_mask);
sigaction(SIGINT, &sa, nullptr);
sigaction(SIGTERM, &sa, nullptr);
```

3.2.4. Передача данных

Отправка статического контента реализована в функции handle_write. Файл читается блоками и отправляется в сокет. Если сокет не готов к записи (возвращен EAGAIN), передача приостанавливается.

```
char file_buf[FILE_CHUNK];

// Чтение блока из файла

ssize_t r = ::read(conn.file_fd, file_buf, to_read);

if (r > 0) {

    conn.file_offset += r;

    ssize_t sent_total = 0;

    // Отправка блока клиенту

    while (sent_total < r) {

        ssize_t n = ::send(conn.fd, file_buf + sent_total, r - sent_total, 0);

        if (n > 0) {

            sent_total += n;

        } else if (n < 0 && (errno == EAGAIN || errno == EWOULDBLOCK)) {

            // Сокет переполнен, сохраняем остаток и ждем

            conn.out_buf.assign(file_buf + sent_total, r - sent_total);

            return;

        }

    }

}
```

4. Исследовательская часть

4.1. Технические характеристики

Технические характеристики устройства:

—операционная система Manjaro Linux x86_64;

—процессор Ryzen 5500U 6 ядер, тактовая частота 2.1 ГГц;

—оперативная память 16 Гбайт.

При тестировании ноутбук был включён в сеть электропитания. Во время тестирования ноутбук был нагружен только системными приложениями окружения, а также системой тестирования.

4.2. Нагрузочное тестирование

Целью нагрузочного тестирования является оценка производительности разработанного сервера при различной конкурентной нагрузке.

Тестирование проводилось при помощи ApacheBench [2]. Тестовым ресурсом является файл размером 10 МБ, заполненный случайными данными.

Варьируемые параметры:

—количество рабочих процессов: 1, 2, 4, 8;

—уровень конкурентности: от 1 до 1000.

4.3. Результаты тестирования

Результаты нагрузочного тестирования представлены на графиках ниже.

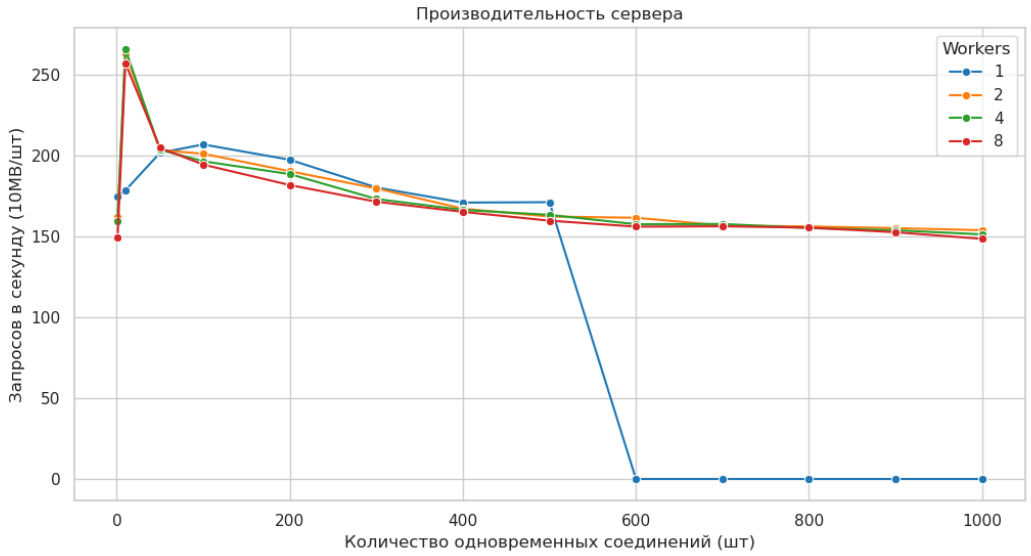

Рисунок 4.1 — Зависимость производительности от числа клиентов

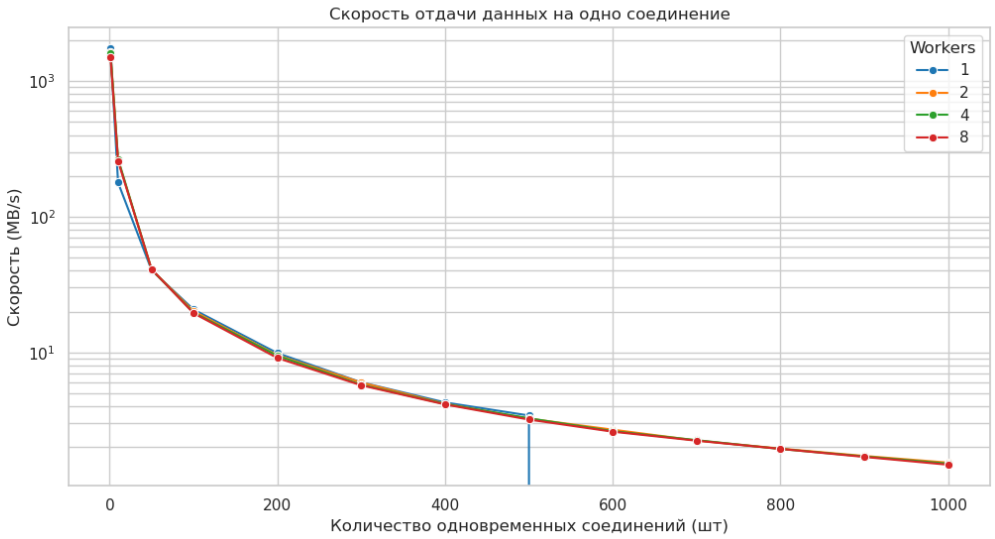

Рисунок 4.2 — Скорость отдачи данных на одно соединение

Сводные данные пиковой производительности приведены в таблице 4.1.

Таблица 4.1 — Пиковая производительность сервера

Кол-во обработчиков	Пиковый RPS	Макс кол-во стабильных соединений
1	206.97	500
2	263.58	>1000
4	265.92	>1000
8	257.26	>1000

4.4. Выводы по результатам тестирования

Анализ полученных данных позволяет сделать следующие выводы:

Масштабируемость — при увеличении количества рабочих процессов с 1 до 2 наблюдается прирост производительности (RPS увеличился с ~206 до ~263). Дальнейшее увеличение числа обработчиков (до 4 и 8) не дает линейного роста RPS в данном тесте, так как узким местом становится пропускная способность сетевого интерфейса или дисковой подсистемы при передаче крупных файлов (10 МБ).

Устойчивость под нагрузкой — конфигурации с 2, 4 и 8 обработчиками успешно справились с нагрузкой в 1000 одновременных соединений, сохраняя стабильную скорость обработки. Конфигурация с 1 процессом показала падение

производительности до нуля при нагрузке свыше 500 соединений. Наблюдаемое падение вызвано архитектурным ограничением системного вызова pselect. Размер битовой маски дескрипторов (FD_SETSIZE) в Linux по умолчанию составляет 1024.

Для обслуживания одного клиента сервер в активной фазе использует два файловых дескриптора: один для сетевого сокета, второй для чтения файла с диска. Следовательно, один процесс может обслуживать не более 1024/2≈512 клиентов. При нагрузке в 600 соединений лимит дескрипторов исчерпывается, сервер принудительно закрывает лишние соединения.

ЗАКЛЮЧЕНИЕ

В ходе курсовой работы была успешно достигнута поставленная цель — разработан и протестирован статический HTTP-сервер, использующий архитектуру prefork в сочетании с механизмом мультиплексирования pselect.

Были решены все поставленные задачи:

- разработан сервер, корректно обрабатывающий GET/HEAD запросы и основные коды состояния HTTP;
- реализована потоковая передача файлов большого объема и журналирование событий;
- проведено нагрузочное тестирование, которое показало, что архитектура prefork эффективно решает проблему масштабирования по количеству соединений, преодолевая ограничение системного вызова pselect по числу файловых дескрипторов на один процесс.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Fielding R., Gettys J., Mogul J. Hypertext Transfer Protocol — HTTP/1.1 [Электронный ресурс]. — Режим доступа: https://www.rfc-editor.org/rfc/rfc2616 (дата обращения: 01.12.2025).
2. The Apache Software Foundation. Apache HTTP Server Benchmarking Tool [Электронный ресурс]. — Режим доступа: https://httpd.apache.org/docs/2.4/programs/ab.html (дата обращения: 01.12.2025).
3. The Linux Foundation. pselect(2) — Linux manual page [Электронный ресурс]. — Режим доступа: https://man7.org/linux/man-pages/man2/pselect.2.html (дата обращения: 01.12.2025).
4. Kerrisk M. The Linux Programming Interface — San Francisco: No Starch Press, 2010. — 1552 p.
5. Stevens W. R., Rago S. A. Advanced Programming in the UNIX Environment. 3rd Edition. — Addison-Wesley Professional, 2013. — 1024 p.
6. Kurose J. F., Ross K. W. Computer Networking: A Top-Down Approach. 7th Edition. — Pearson, 2017. — 888 p.

ПРИЛОЖЕНИЕ А

Презентация к курсовой работе состоит из 8 слайдов.