



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 ИУ7 «Программное обеспечение ЭВМ и информационные технологии»

## КУРСОВАЯ РАБОТА

*НА ТЕМУ:*

*«Разработка статического веб-сервера»*

Студент      ИУ7-72Б

\_\_\_\_\_ Е. Д. Пермякова

(Подпись, дата)

(И.О.Фамилия)

Руководитель

\_\_\_\_\_ М. Н. Клочков

(Подпись, дата)

(И.О.Фамилия)

Рекомендованная руководителем оценка: \_\_\_\_\_

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

(индекс)

И. В. Рудаков

(И.О. Фамилия)

(подпись)

(дата)

## З А Д А Н И Е на выполнение курсового проекта

по дисциплине Компьютерные сети

Студент группы ИУ7-72Б Пермякова Екатерина Дмитриевна

(Фамилия, имя, отчество)

Тема курсового проекта Разработка статического веб-сервера

Направленность КП (учебная, исследовательская, практическая, производственная, др.)  
учебная

Источник тематики (кафедра, предприятие, НИР) кафедра

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 (лог);  
Учесть минимальные требования к безопасности серверов статического содержимого. Провести  
нагрузочное тестирование разработанного сервера: Максимальное количество обслуживаемых  
сетевых соединений; Скорость отдачи данных по каждому сетевому соединению и совокупная.  
Вариант архитектуры разрабатываемого сервера: prefork + poll().

Оформление курсового проекта:

Расчетно-пояснительная записка на 12-32 листах формата А4.

Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)

Презентация к курсовой работе на 8-16 слайдах

Дата выдачи задания « \_\_\_\_ » \_\_\_\_\_ 20 \_\_\_\_ г.

Руководитель курсового проекта

Клочков М. Н.

(подпись, дата)

(И.О. Фамилия)

Студент

Пермякова Е. Д.

(подпись, дата)

(И.О. Фамилия)

Примечание: Задание оформляется в двух экземплярах: один выдается студенту, второй хранится  
на кафедре.

## **РЕФЕРАТ**

Расчетно-пояснительная записка 20 с., 5 рис., 1 таблица, 8 источников, 1 приложение.

Цель работы — разработка HTTP-сервера для отдачи статического содержимого на основе архитектуры prefork с использованием системного вызова poll().

В рамках курсовой работы разработан HTTP-сервер для отдачи статического содержимого на основе архитектуры prefork с использованием системного вызова poll() и проведено нагружочное тестирование для определения максимального количества обслуживаемых сетевых соединений и скорости отдачи данных по каждому сетевому соединению и совокупной.

# СОДЕРЖАНИЕ

<b>РЕФЕРАТ</b>	3
<b>ВВЕДЕНИЕ</b>	5
<b>1 Аналитическая часть</b>	6
1.1 Анализ протокола HTTP для статического контента	6
1.2 Анализ архитектуры prefork + poll()	7
<b>2 Конструкторская часть</b>	10
2.1 Описание алгоритма запуска сервера	10
2.2 Описание обработки запроса пользователя	11
<b>3 Технологическая часть</b>	13
3.1 Средства реализации	13
3.2 Тестирование	13
<b>4 Исследовательская часть</b>	15
4.1 Технические характеристики	15
4.2 Цель исследования	15
4.3 Описание исследования	15
4.4 Результат исследования	16
4.5 Анализ результатов	17
4.6 Вывод из исследовательской части	17
<b>ЗАКЛЮЧЕНИЕ</b>	18
<b>СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ</b>	19
<b>Приложение А</b>	20

## **ВВЕДЕНИЕ**

Целью данной работы является разработка HTTP-сервера для отдачи статического содержимого на основе архитектуры prefork с использованием системного вызова poll().

Для достижения поставленной цели требуется решить следующие задачи:

- 1) Проанализировать протокол HTTP/1.1 и методы GET, HEAD, а также коды состояний 200, 403, 404, 405;
- 2) Спроектировать архитектуру сервера, соответствующую варианту: prefork с использованием poll() для мультиплексирования входящих соединений в каждом процессе;
- 3) Разработать веб-сервер с архитектурой prefork + poll();
- 4) Провести нагрузочное тестирование сервера для оценки максимального количества обслуживаемых соединений, скорости отдачи данных на соединение и совокупной пропускной способности;

# **1 Аналитическая часть**

В данном разделе рассматривается протокол HTTP/1.1 и архитектура prefork + poll().

## **1.1 Анализ протокола HTTP для статического контента**

### **HTTP/1.1**

HTTP (HyperText Transfer Protocol) — протокол прикладного уровня модели OSI, предназначенный для передачи гипертекстовых документов. Версия HTTP/1.1 [8] поддерживает постоянные соединения (keep-alive), сокращая накладные расходы на установку TCP-соединения для каждого запроса.

#### **Структура HTTP-сообщений**

Основной единицей взаимодействия в HTTP является сообщение. Запрос клиента состоит из:

- Стартовой строки — содержит метод, URI и версию протокола.
- Заголовков (headers) — параметры запроса в формате «ключ: значение».
- Тела запроса (body) — опциональные данные (отсутствует для GET и HEAD).

Ответ сервера состоит из:

- Строки статуса — включает версию протокола, код и описание состояния.
- Заголовков ответа — метаинформация о содержимом и сервере.
- Тела ответа — передаваемые данные (только для метода GET).

#### **MIME-типы контента**

Для корректной интерпретации данных клиентом используется система MIME-типов (Multipurpose Internet Mail Extensions), определяющая формат содержимого. Тип указывается в заголовке Content-Type ответа и формируется на основе расширения файла, например text/html, text/css, image/jpeg и т. д.

## **Поддерживаемые методы запросов**

Разрабатываемый сервер статического содержимого обрабатывает запросы:

- GET — возвращает содержимое ресурса с телом ответа.
- HEAD — возвращает только заголовки ответа без тела.
- Прочие методы (POST, PUT, DELETE) отклоняются с кодом состояния 405 Method Not Allowed.

## **Коды состояния HTTP**

Код состояния является трехзначным числом, информирующим клиента о результате обработки его запроса. Для корректной работы статического сервера необходимо реализовать обработку следующих статусов:

- 200 OK — запрос успешно выполнен; целевой ресурс передан в теле ответа.
- 403 Forbidden — сервер отказывается выполнить запрос из-за отсутствия прав доступа к целевому ресурсу;
- 404 Not Found — целевой ресурс не существует по указанному URI;
- 405 Method Not Allowed — метод запроса не поддерживается для целевого ресурса.

## **1.2 Анализ архитектуры prefork + poll()**

### **Модель prefork**

Prefork (предварительное порождение процессов) — это архитектурный подход к созданию параллельных серверов, при котором основной (родительский, master) процесс заранее создает фиксированный пул дочерних процессов (worker), ожидающих входящие соединения. Родительский процесс обычно отвечает за мониторинг состояния воркеров и их перезапуск в случае аварийного завершения.

Принцип работы модели prefork:

1. Сервер запускается и создает слушающий сокет.
2. Master-процесс порождает заданное количество идентичных worker-процессов с помощью системного вызова `fork()` [4].
3. Все worker-процессы наследуют слушающий сокет и одновременно вызывают `accept()` [1] для ожидания нового соединения. Благодаря механизму ядра, только один процесс получит входящее соединение.
4. Получивший соединение worker обрабатывает его независимо от других.

## **Модель мультиплексирования с использованием `poll()`**

Мультиплексирование ввода-вывода — это техника, позволяющая одному процессу (или потоку) отслеживать состояние нескольких файловых дескрипторов (например, сетевых сокетов) одновременно, не блокируя выполнение на ожидании данных от каждого из них по отдельности.

Системный вызов `poll()` [7] предоставляет механизм для синхронного ожидания событий на множестве дескрипторов.

Принцип работы системного вызова `poll()`:

1. Программа создает массив структур типа `struct pollfd`. Для каждого отслеживаемого файлового дескриптора в структуре указывается: сам дескриптор (`fd`), интересующие события (`events`) и произошедшие события (`revents`).
2. Типичные отслеживаемые события:
  - `POLLIN` — данные доступны для чтения (поступление нового HTTP-запроса).
  - `POLLOUT` — дескриптор готов для записи (можно отправлять HTTP-ответ).
  - `POLLERR, POLLHUP` — произошла ошибка или соединение закрыто.
3. Программа передает массив структур и таймаут в системный вызов `poll()`. Вызов блокирует выполнение процесса до тех пор, пока

не произойдет хотя бы одно из запрошенных событий на одном из дескрипторов, либо не истечет таймаут.

4. Когда `poll()` возвращает управление, программа сканирует массив, проверяя поле `revents` в каждой структуре, чтобы определить, на каких дескрипторах произошли события.
5. Для каждого «готового» дескриптора выполняется соответствующая операция (чтение запроса, отправка ответа, закрытие соединения).

## **prefork + poll()**

Архитектура `prefork + poll()` представляет собой гибридный подход, сочетающий преимущества многопроцессной модели и механизма мультиплексирования ввода-вывода.

При данной архитектуре каждый `worker`-процесс способен обслуживать несколько одновременных соединений в рамках одного потока выполнения, используя системный вызов `poll()` для отслеживания состояния множества файловых дескрипторов.

## **Вывод**

В данном разделе были представлено описание протокола HTTP/1.1 и архитектуры `prefork + poll()`.

## 2 Конструкторская часть

В данном разделе представлено описание работы сервера.

### 2.1 Описание алгоритма запуска сервера

На рисунке представлено описание алгоритма запуска сервера 2.1

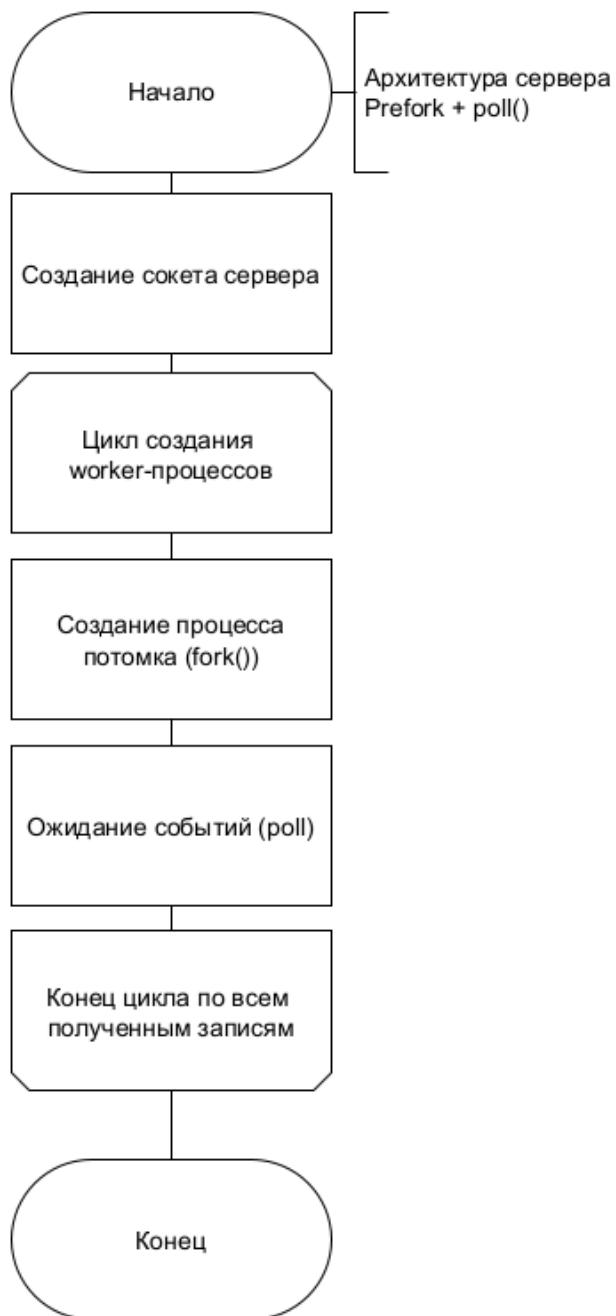


Рисунок 2.1 – Описание алгоритма запуска сервера

## 2.2 Описание обработки запроса пользователя

На рисунках 2.2-2.3 представлено описание алгоритма обработки запроса пользователя.

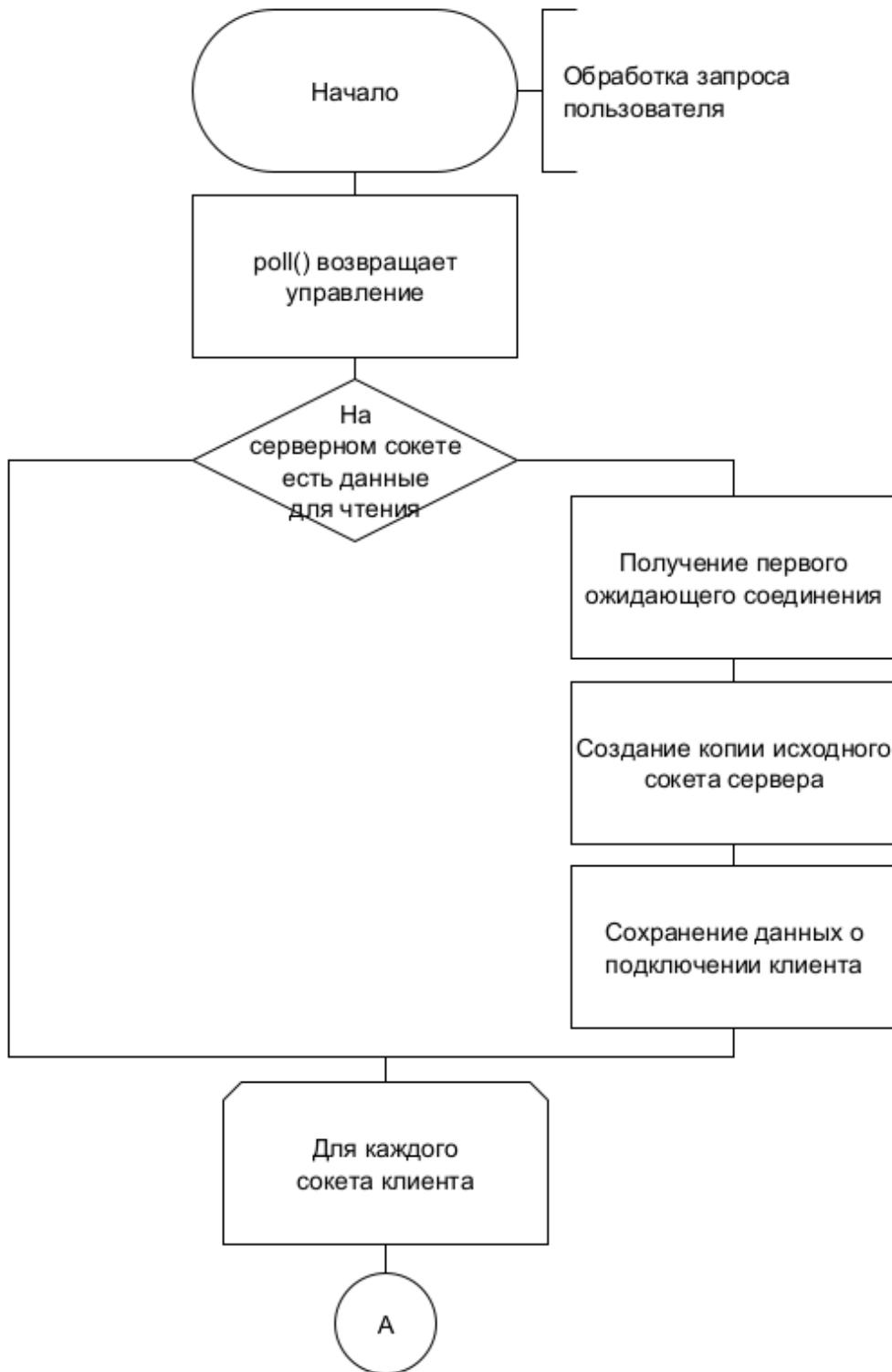


Рисунок 2.2 – Описание обработки запроса пользователя (часть 1)

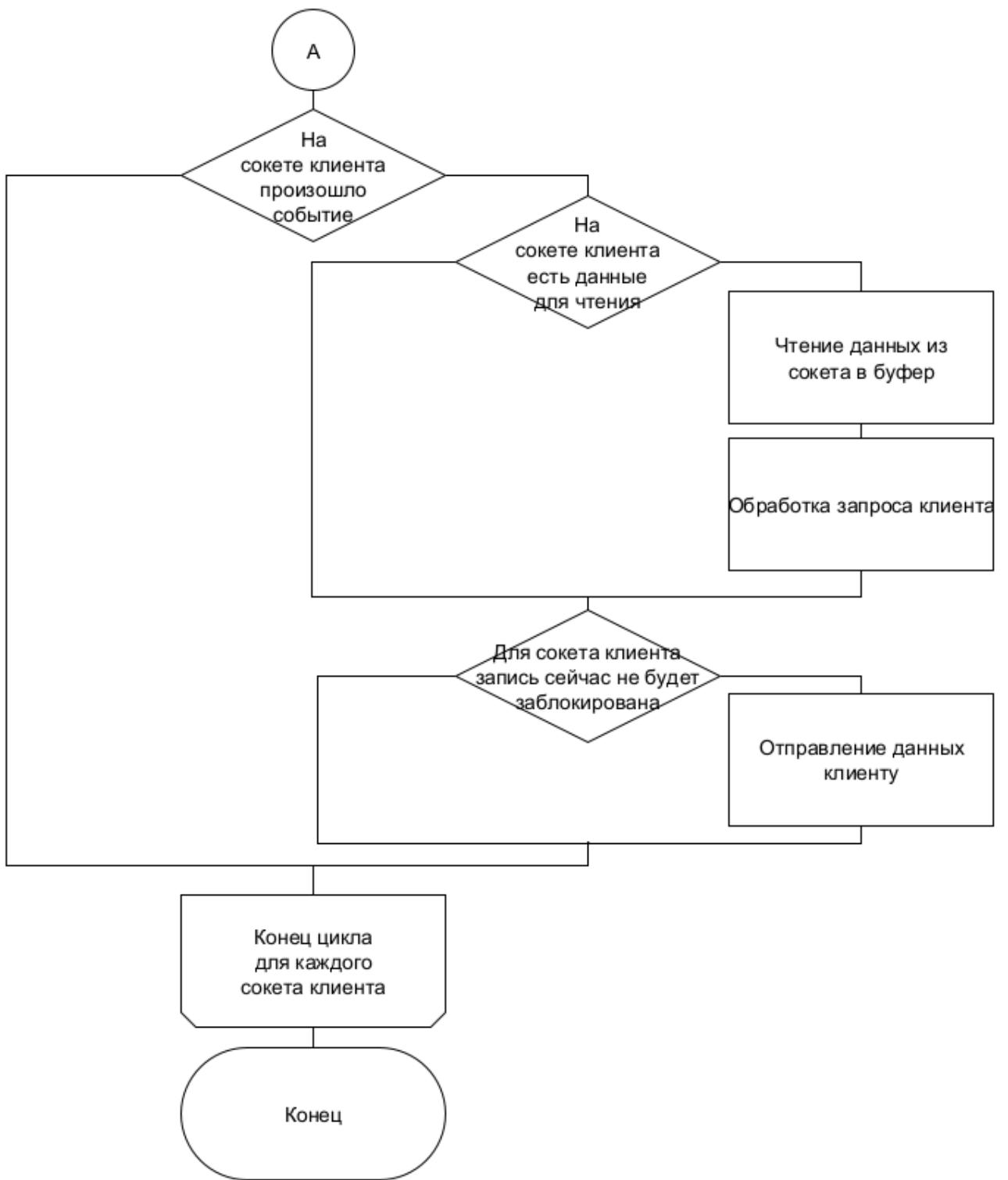


Рисунок 2.3 – Описание обработки запроса пользователя (часть 2)

## Вывод

В данном разделе были описаны начало работы сервера и алгоритм обработки запроса клиента.

### **3 Технологическая часть**

В данном разделе представлены средства реализации и описано проведенное тестирование.

#### **3.1 Средства реализации**

Для написания веб-сервера использовались средства реализации:

- Язык программирования: C [2];
- Компилятор: gcc [5];
- Система сборки: Make [6];

Тестирование проводилось с использованием утилиты curl [3].

#### **3.2 Тестирование**

Для функционального тестирования работы веб-сервера был написан скрипт, который отправляет запросы на сервера с помощью утилиты curl [3]. Были рассмотрены классы эквивалентности, описанные в таблице 3.1:

Таблица 3.1 – Функциональные тесты

Описание теста	Ожидаемый код состояния HTTP
HEAD запрос существующего файла размером до 128 Мбайт	200
GET запрос существующего файла размером до 128 Мбайт	200
Запрос несуществующего файла	404
Обращение к родительской директории (содержание двоеточия в пути)	403
Обращение к файлу доступ к которому запрещен	403
GET запрос существующего файла размером более 128 Мбайт	403
POST запрос	405
PUT запрос	405

## Продолжение таблицы 3.1

Описание теста	Ожидаемый код состояния HTTP
DELETE запрос	405

Все тесты пройдены успешно.

## **Вывод**

Была представлены средства реализации и описано проведенное тестирование.

## **4 Исследовательская часть**

В данном разделе описано нагружочное тестирование разработанного веб-сервера.

### **4.1 Технические характеристики**

- Гостевая операционная система — Ubuntu 22.04.5 LTS (WSL 2)
- Оперативная память (RAM) – 8,0 ГБ;
- Процессор – AMD Ryzen 7 5800H with Radeon Graphics, 3201 МГц, ядер: 8, логических процессоров: 16;

При проведении замеров времени ноутбук был включен в сеть электропитания, и были запущены только встроенное приложение окружения и система замеров времени.

### **4.2 Цель исследования**

Цель исследования – определение максимального количества обслуживаемых сетевых соединений и скорости отдачи данных по каждому сетевому соединению и совокупная для разработанного веб-сервера.

### **4.3 Описание исследования**

В ходе исследования производились следующие действия:

- 1) Поднимался разработанный веб-сервер на локальном хосте;
- 2) Проводилось нагружочное тестирование с помощью инструмента wrk с заданными параметрами: количество параллельных соединений, время работы;
- 3) По завершении тестирования веб-сервер останавливался;
- 4) Результаты тестирования сохранялись;
- 5) Шаги 1–4 повторялись 10 раз для компенсации случайных отклонений;
- 6) Для каждого набора параметров вычислялось среднее арифметическое всех метрик, собранных на шаге 5;

- 7) Увеличивалось количество рассматриваемых параллельных соединений на 50;
- 8) Шаги 1–6 повторялись до достижения 1500 параллельных соединений.

#### 4.4 Результат исследования

По результатам тестирования были построены графики: зависимость количества запросов в секунду от числа одновременных запросов к серверу(рисунок 4.1) и зависимость скорости отдачи данных по каждому сетевому соединению и в совокупности(рисунок 4.2).

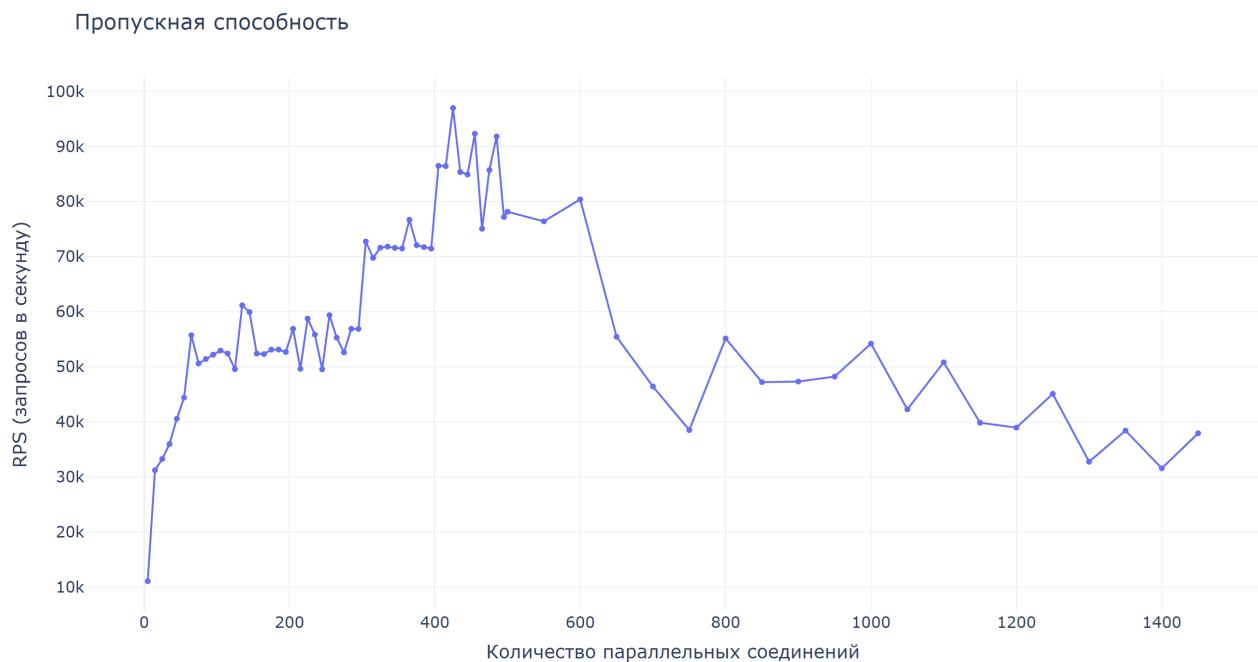


Рисунок 4.1 – Зависимость количества запросов в секунду от числа одновременных запросов к серверу

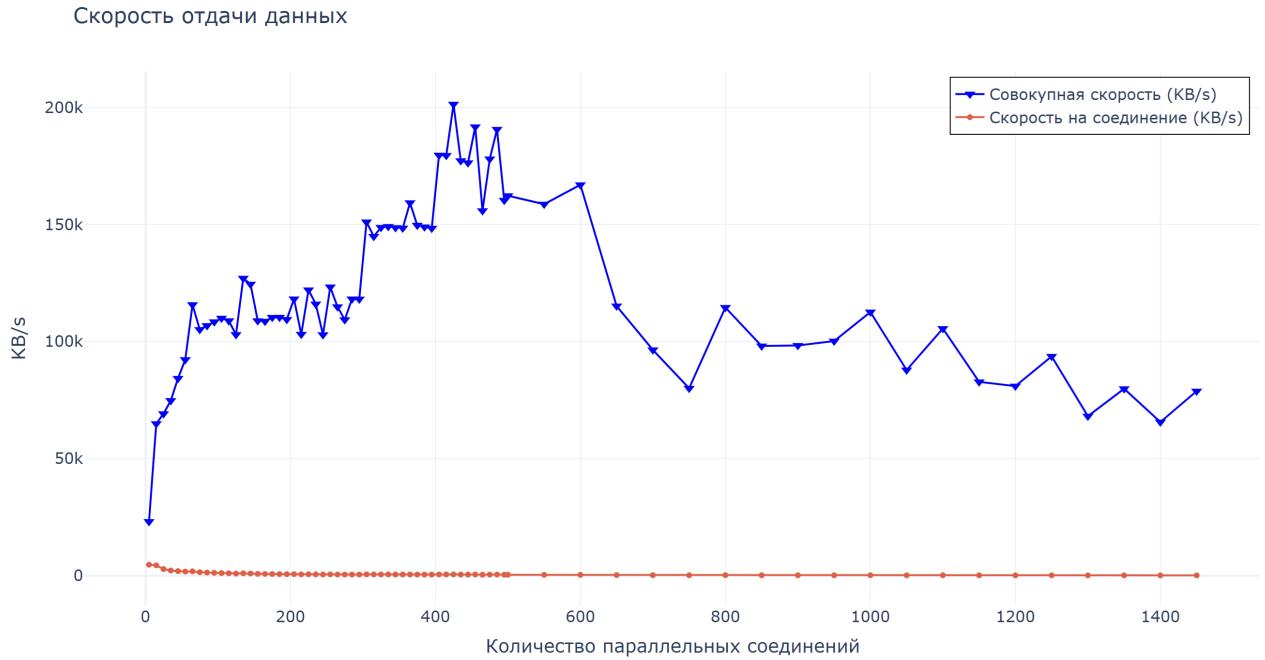


Рисунок 4.2 – Зависимость скорости отдачи данных по каждому сетевому соединению и в совокупности

## 4.5 Анализ результатов

В ходе тестирования установлено, что критический порог производительности составляет 500 одновременных соединений, после которого наблюдается снижение эффективности работы сервера. При данной нагрузке максимальная скорость обработки данных достигает 170 МБайт/секунду.

## 4.6 Вывод из исследовательской части

В данном разделе было описано нагружочное тестирование разработанного веб-сервера и определены максимальное количество обслуживаемых сетевых соединений и скорость отдачи данных по каждому сетевому соединению и совокупная.

## **ЗАКЛЮЧЕНИЕ**

В ходе курсовой работы был разработан веб-сервер с архитектурой prefork + poll().

В процессе выполнения данной работы были выполнены все поставленные задачи:

1. Проанализировать протокол HTTP/1.1 и методы GET, HEAD, а также коды состояний 200, 403, 404, 405;
2. Спроектировать архитектуру сервера, соответствующую варианту: prefork с использованием poll() для мультиплексирования входящих соединений в каждом процессе;
3. Разработать веб-сервер с архитектурой prefork + poll();
4. Провести нагрузочное тестирование сервера для оценки максимального количества обслуживаемых соединений, скорости отдачи данных на соединение и совокупной пропускной способности;

## **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. accept(2) - Linux manual page [Электронный ресурс]. — Режим доступа: <https://man7.org/linux/man-pages/man2/accept.2.html> (дата обращения: 13.12.2025).
2. C language [Электронный ресурс]. — Режим доступа: <https://en.cppreference.com/w/c/language.html> (дата обращения: 13.12.2025).
3. curl(1) - Linux manual page [Электронный ресурс]. — Режим доступа: <https://man7.org/linux/man-pages/man1/curl.1.html> (дата обращения: 13.12.2025).
4. fork(2) - Linux manual page [Электронный ресурс]. — Режим доступа: <https://man7.org/linux/man-pages/man2/fork.2.html> (дата обращения: 13.12.2025).
5. gcc(1) - Linux manual page [Электронный ресурс]. — Режим доступа: <https://man7.org/linux/man-pages/man1/gcc.1.html> (дата обращения: 13.12.2025).
6. make(1) - Linux manual page [Электронный ресурс]. — Режим доступа: <https://man7.org/linux/man-pages/man1/make.1.html> (дата обращения: 13.12.2025).
7. poll(2) - Linux manual page [Электронный ресурс]. — Режим доступа: <https://man7.org/linux/man-pages/man2/poll.2.html> (дата обращения: 13.12.2025).
8. RFC 2616 [Электронный ресурс]. — Режим доступа: <https://2rfc.net/2616> (дата обращения: 13.12.2025).

## **Приложение А**

Презентация к курсовой работе 7 с.