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 «ИНФОРМАЦИОННОЕ ОБЕСПЕЧЕНИЕ ЭВМ И ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ»

КУРСОВАЯ РАБОТА

НА ТЕМУ:

Разработка статического веб-сервера

Студент **ИУ7-72Б** _____ _____
(группа) (подпись, дата) (И.О. Фамилия)

Руководитель курсового проекта _____ **Клочков М.Н.**
(подпись, дата) (И.О. Фамилия)

Консультант _____ _____
(подпись, дата) (И.О. Фамилия)

2025 г.

РЕФЕРАТ

Курсовая работа 22 с., 7 рис., 14 ист., 1 табл., 1 прил.

Ключевые слова: Веб-сервер, многопоточность, мультиплексирование, кеширование, асинхронность, C, UNIX,

Цель работы: разработка статического веб-сервера

Результаты работы:

— рассмотрены требуемые системные вызовы UNIX;

— спроектирована архитектура сервера и состояния для обработки входящих запросов;

— формализованы алгоритмы обработки обработки запроса;

— выбраны средства реализации статического веб-сервера;

— реализован статический веб-сервер;

— исследовано максимальное количество обслуживаемых соединений и скорость отдачи файлов.

Исследование показало, что при 8 рабочих потоках веб-сервер выдерживает одновременное количество соединений не более 800 без существенной потери производительности, а также в среднем имеет пропускную способность 50 МБайт/с при данном количестве соединений.

Содержание

ВВЕДЕНИЕ

Целью данной работы является разработка статического веб-сервера.

Для достижения поставленной цели необходимо решить следующие задачи:

1) рассмотреть требуемые системные вызовы UNIX;
2) спроектировать архитектуру сервера и состояния для обработки входящих запросов;
3) формализовать алгоритмы обработки обработки запроса;
4) выбрать средства реализации статического веб-сервера;
5) реализовать статический веб-сервер;
6) исследовать максимальное количество обслуживаемых соединений и скорость отдачи файлов.

1 Аналитическая часть

В аналитической части будет проведён анализ требований к разрабатываемому приложению, описаны системные вызовы, требуемые для реализации работы.

1.1 Требование к разрабатываемому приложению

Разрабатываемый статический веб-сервер предназначен для отдачи статического содержимого с диска, из заданной директории по протоколу HTTP. При этом должна быть реализована следующая функциональность:

— сервер должен поддерживать HTTP запросы GET и HEAD;

— путь в URL [1] – путь к файлу, относительно заданной директории на диске;

— в случае если путь пустой(/) сервер должен отдавать HTML-страницу с информацией о сервере с заданным CSS-стилем;

— в случае, если запрошенный файл не найден на диске, сервер должен отдавать ответ со статусом 404;

— в случае, если сервер не имеет доступ к запрошенному файлу, он должен отдавать статус 403;

— в случае запроса с неподдерживаемым методом, сервер должен отдавать статус 405.

При этом при реализации должны быть учтены следующие технические особенности:

— должна быть обеспечена корректная обработка файлов размером до 128 Мбайт;

— должны быть реализованы записи информации о событиях в журнал (лог);

— должно быть реализовано мультиплексирование с псевдопараллельной обработкой – каждый поток или процесс должны отдавать файлы по нескольким соединениям;

— для обеспечения псевдопараллельности нужно использовать пул потоков (thread pool);

— для обеспечения мультиплексирования нужно использовать системный вызов pselect.

1.2 Используемые системные вызовы

Для разработки статического веб-сервера потребуются системные вызовы [2], обеспечивающие работу с многопоточность, мультиплексированием файлов (сокетов, обычных файлов), работы с файлами на диске.

1.2.1 Многопоточность: pthreads

Библиотека POSIX Threads [3] предоставляет системные вызовы для работы с потоками операционной системы.

— **pthread_create** — создаёт новый поток выполнения внутри процесса. С помощью него организуется пул потоков, для организации псевдопараллельной обработки запросов;

— **pthread_join** — ожидание завершения потока.

— **pthread_mutex_init**, **_lock**, **_unlock** — создают и управляют взаимоисключениями (мьютексами), которые обеспечивают одновременный доступ только одного потока к общим ресурсам (участкам кода).

— **pthread_cond_init**, **_wait**, **_signal** — функциональность условных переменных (condition variables). Может быть использовано для блокировки потоков, ожидающих поступления новых задач, и для их пробуждения при добавлении задания в очередь, или как индикатор завершении работы.

— **pthread_rwlock_init**, **_rdlock**, **_wrlock**, **_unlock** — функции для работы с раздельными блокировками чтения и записи. Применяется для разрешения одновременного чтения, но запрета на одновременную запись и чтения или нескольких записей, например при доступе к кешированным файлам.

1.2.2 Мультиплексирование: pselect

pselect [4] — системный вызов, позволяющий мониторить множество файловых дескрипторов (сокеты, файлы, каналы) на готовность к операции чтения, записи или возникновение исключений.

Работа с pselect строится следующим по следующему циклу:

1) **Инициализация:** создаётся два множества, в которые добавляются все дескрипторы, который нужно прослушивать на возможность чтения и записи соответственно;

2) **Вызов pselect:** после чего поток блокируется до возможности чтения или записи на одном или нескольких дескрипторов;

3) **Обработка дескрипторов:** с помощью макроса FD_ISSET нужно проверить все дескрипторе в множествах на готовность. В случае положительно результата производится обработка дескриптора;

В разрабатываемом сервере pselect применяется для одновременной работы с несколькими соединениями в рамках одного потока.

1.2.3 Работа с файлами

Для работы со статическим содержимым используются стандартные POSIX-вызовы:

— **open** [5] — открывает файл по указанному пути и возвращает файловый дескриптор;

— **read** [6] — читает данные из файла в буфер. Применяется для чтения запроса из сокетов и чтения файлов с диска;

— **write** [7] — записывает данные в файловый дескриптор клиента. Используется для отправки HTTP-заголовков и тела ответа;

— **close** [5] — закрывает файловый дескриптор после завершения обработки запроса;

— **lstat** [8] — получает информацию о файле (тип, права доступа, размер). Используется для определения существования файла, возможности чтения и максимального размера.

Вывод

В результате аналитической части был проведён анализ требований для разрабатываемого ПО и рассмотрены системные вызовы, необходимые при реализации статического веб-сервера.

2 Конструкторская часть

В данной главе рассматривается архитектура разрабатываемого веб-сервера, его основные компоненты и их взаимодействие. Также описываются состояния, через которые проходит HTTP-запрос в процессе обработки, и алгоритмы обработки в каждом из состояний.

2.1 Архитектура веб-сервера

Для разграничения зон ответственности, веб-сервер был разбит на отдельные компоненты, выполняющие определённые функции и предоставляющие другим компонентам интерфейс взаимодействие. Так как основная функция статического веб-сервера – отдача файла, можно выделить основные задачи:

— принятие входящих запросов и распределение их по потокам;

— работа с HTTP;

— работа с файловой системой;

Исходя из этого было выделено 5 основных компонентов:

— **ядро сервера** — управляет жизненным циклом системы, создаёт пул рабочих потоков, менеджер кэша и пул читателей, а также принимает входящие соединения и распределяет их между рабочими потоками;

— **менеджер кэша** — хранит считанные файлы в памяти, при необходимости освобождает наименее используемые файлы;

— **пул читателей** — выполняет асинхронное чтение файлов с диска в кэш и уведомляет рабочие потоки о завершении операций.

— **пул рабочих потоков** — обрабатывает несколько HTTP-запросов, используя `pselect` для мультиплексирования;

— **подсистема обработки запросов** — управляет состояниями запросов, выполняет чтение данных, парсинг HTTP-заголовков и подготовку ответа.

На рисунке 2.1 представлена диаграмма компонентов веб-сервера.

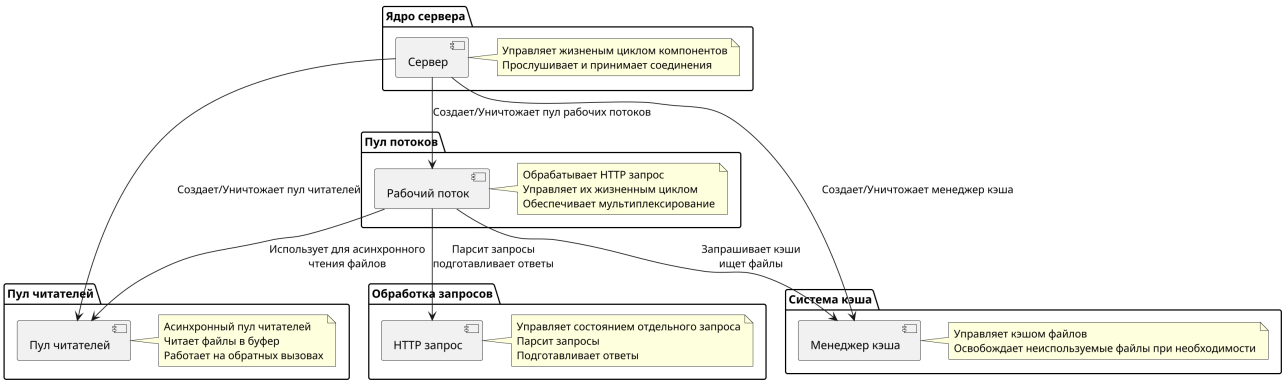

Рисунок 2.1 — Диаграмма компонентов веб-сервера

2.2 Состояния обрабатываемого запроса

В процессе обработки запроса сервером можно выделить несколько состояний, каждый из которых отвечает за решение определённых задач:

— **соединение** — запрос принят ядром сервером и назначен рабочему потоку;

— **чтение** — чтение запроса из сокета и его разбор на составляющие;

— **ожидание тела** — файл отсутствует в кэше, ожидание завершения его чтения;

— **запись** — отправка HTTP-ответа, с телом или без;

— **закончен** — ответ успешно отправлен, ресурсы освобождены;

— **ошибка** — возникла необрабатываемая ошибка – отправка ошибки и освобождение ресурсов;

На рисунке 2.2 изображена диаграмма состояний, описывающих переходы между ними.

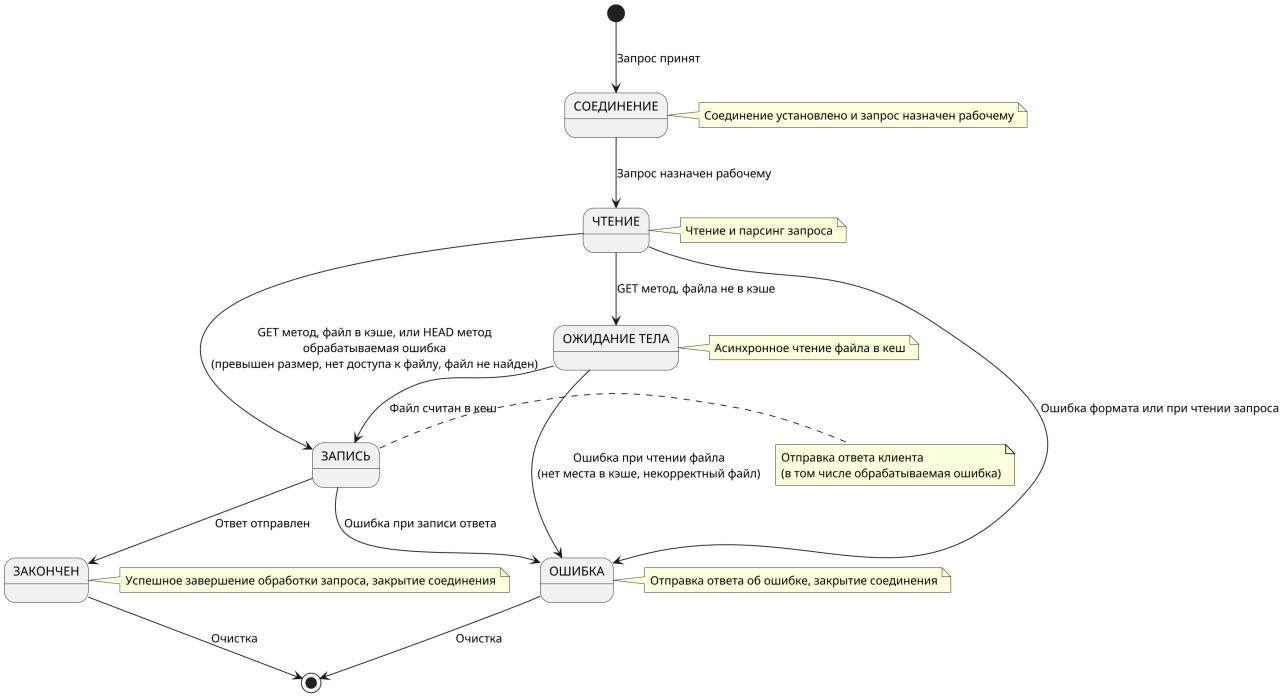

Рисунок 2.2 — Диаграмма состояний HTTP-запроса

2.3 Алгоритмы обработки запросов

В случае состояний «соединение», «ожидание тела», «закончен» и «ошибка» обработка тривиальна, и включает в себя простые шаги:

— **соединение** — при первой обработке, рабочий поток переводит в «чтение»;

— **ожидание тела** — при завершении считывания файла, перевести запрос в «запись»;

— **закончен** — освободить ресурсы, занимаемые запросом;

— **ошибка** — отправить ошибку и освободить ресурсы, занимаемые запросом;

В случае «чтение» и «запись», алгоритмы сложнее. Запросы в таких состояниях участвуют в цикле мультиплексирования и считывание или запись происходит постепенно, по готовно-

сти сокетов. Алгоритм чтения описан на рисунках 2.3- 2.4, а алгоритм записи – на рисунке 2.5.

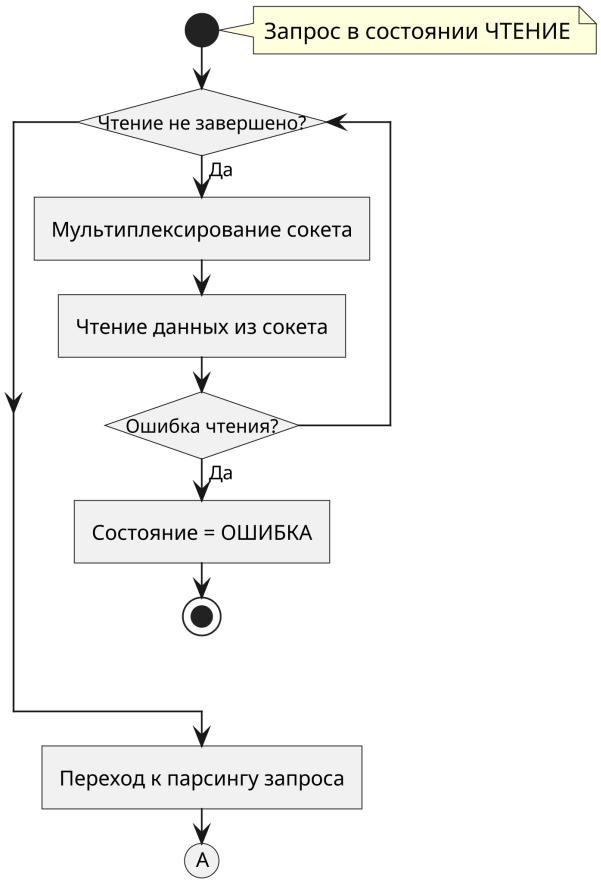

Рисунок 2.3 — Схема алгоритма обработки запроса в состоянии чтения

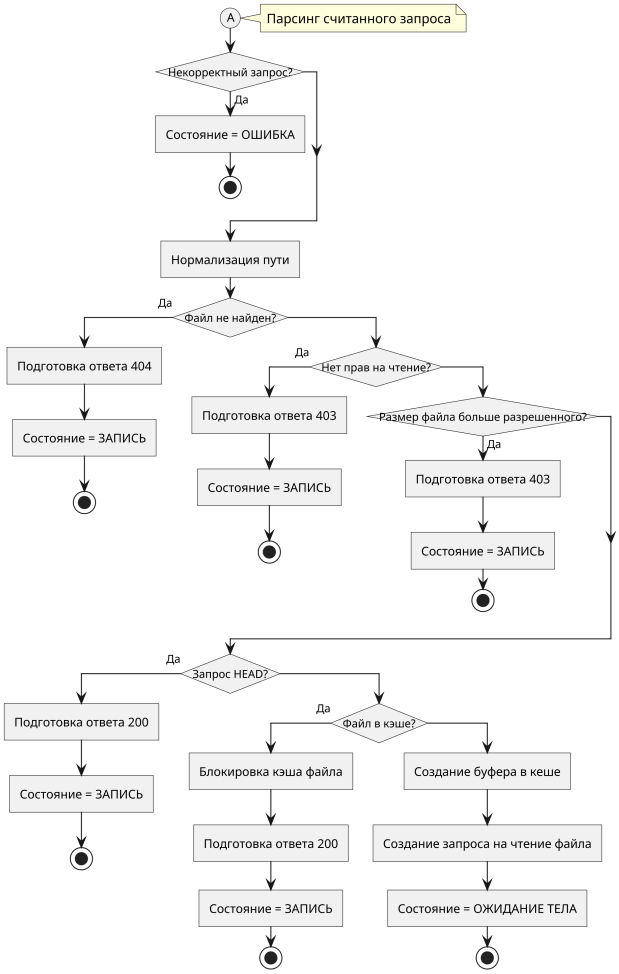

Рисунок 2.4 — Схема алгоритма парсинга запроса

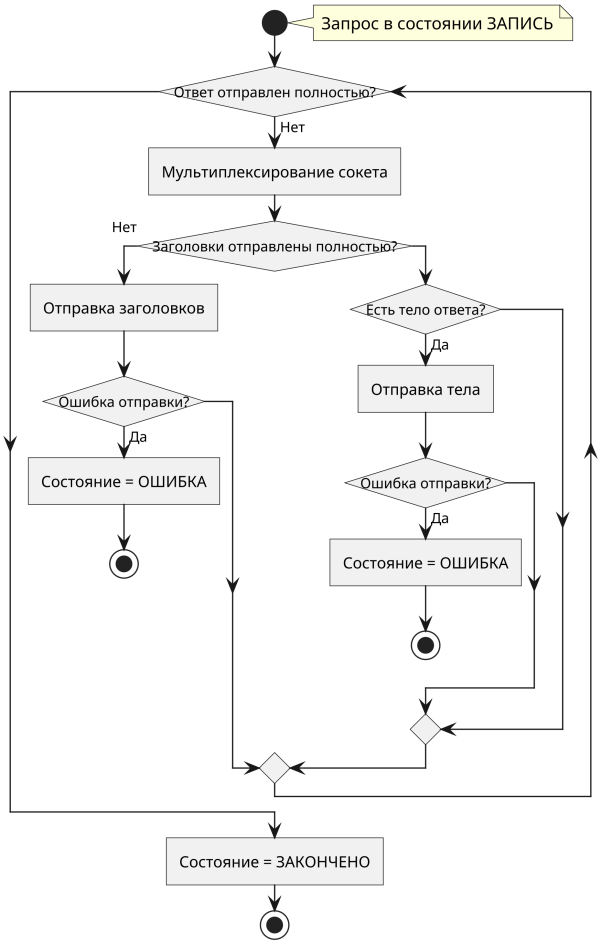

Рисунок 2.5 — Схема алгоритма обработки запроса в состоянии записи

Вывод

В данной части были разработаны архитектура веб-сервера, подробно описаны его основные компоненты и их взаимодействие. Рассмотрены состояния жизненного цикла HTTP-запроса, а также алгоритмы обработки запросов в каждом из состояний.

3 Технологическая часть

В рамках технологической части будут выбраны средства реализации работы, описан интерфейс разработанного приложения, а также описано проведённое тестирование.

3.1 Средства реализации

Для реализации многопоточного веб-сервера с кэшированием были выбраны следующие средства разработки:

— **Язык программирования:** C (стандарт C17) [10]. Выбор обусловлен возможностью использовать на нём системные вызовы, а также тем, что высокоуровневые языки запрещены в рамках данной работы.

— **Компилятор:** GCC [11]. Используется для компиляции исходного кода.

— **Система сборки:** Make [12]. Автоматизирует процесс компиляции и сборки проекта на основе Makefile.

— **Библиотеки:**

- libuuid — для генерации уникальных идентификаторов запросов.
- libcheck [13] — фреймворк для модульного тестирования.
- GCov — инструмент для анализа покрытия кода тестами.

3.2 Интерфейс веб-сервера

Веб-сервер поддерживает следующие аргументы командной строки для настройки его работы:

Опция	Назначение	Формат	По умолчанию	Ограничения
-r <root>	Каталог статических файлов	Строка	data	Существующий каталог
-p <port>	Номер порта для прослушивания	Целое число	8080	1–65535
-c <size>	Максимальный размер кэша	Число с суффиксом (k, m, g)	4g	>0
-e <num>	Максимальное количество записей в кэше	Целое число	1024	>0
-s <size>	Максимальный размер одной записи кэша	Число с суффиксом (k, m, g)	2g	>0
-a <num>	Количество асинхронных читателей	Целое число	4	>0
-m <num>	Максимальное количество запросов на воркер	Целое число	1024	>0
-w <num>	Количество рабочих потоков	Целое число	8	>0
-h	Показать справку	-	-	-

3.3 Тестирование

В рамках тестирования приложения был использован модульный подход с применением фреймворка Check для написания и выполнения unit-тестов.

3.3.1 Тестирование приложения

Для тестирования приложения использовались следующие инструменты:

— **Фреймворк тестирования:** Check — библиотека для модульного тестирования на языке C.

— **Анализ покрытия кода:** GCov — инструмент для измерения покрытия кода тестами.

Тестированию подверглись следующие компоненты приложения:

— Модуль кэширования;

— Модуль хэширования;

— Модуль чтения файлов;

— Модуль определения типа контента;

— Модуль работы с датами;

— Модуль логирования;
— Модуль работы со строками;
— Модуль утилит для строк;

Общий уровень покрытия кода тестами составил **36.85%**.

Вывод

В результате технологической части были выбраны средства реализации приложения, реализован и протестирован веб-сервер, а также описан интерфейс командной строки сервера.

4 Исследовательская часть

В данной части будет проведено нагрузочное тестирование разработанного веб-сервера.

4.1 Описание исследования

Целями данного исследования является определение:

— Максимального количества обслуживаемых сетевых соединений;

— Скорости отдачи данных по каждому сетевому соединению и совокупная.

Нагрузочное тестирование сервера проводится с помощью утилиты apache benchmark (ab) [14], которая позволяет отправлять http-запросы одновременно с нескольких соединений. Также ab позволяет получить следующую информацию при проведении тестирования:

— количество запросов в секунду (rps);

— среднее время ответа сервера (tpr);

— средняя пропускная способность;

Превышения максимального числа запросов характеризуется понижением пропускной способности и количества запросов в секунду, поэтому для его определения необходимо провести серию замеров, где постепенно будет увеличиваться количество одновременных соединений. Аналогичным образом проводится определение скорости отдачи данных, так как она зависит от количества соединений и в какой-то момент должна начать снижаться, при увеличении количества соединений.

Для замеров использовалась следующая конфигурация сервера и ab:

— **рабочие потоки:** 8;

— **размер запрашиваемого файла:** 1.2КБ;

— **количество запросов ab**: 2000;

Замеры проводились при отключённом логировании. Также для каждого количества одновременных соединений проводилось 20 замеров, а результаты затем усреднялись.

Замеры проводились на устройстве со следующими характеристиками:

— Процессор: AMD Ryzen 7 5800H (16 ядер, 4.46 ГГц);

— Оперативная память: 16 ГБ;

— Операционная система: CachyOS x86_64 (ядро linux 6.14.4-2);

4.2 Результаты исследования

На рисунке 4.1 изображён график, отображаемый количество запросов в секунду в зависимости от числа одновременных запросов к серверу. На рисунке 4.2 изображён график пропускной способности, также от числа одновременных запросов к серверу.

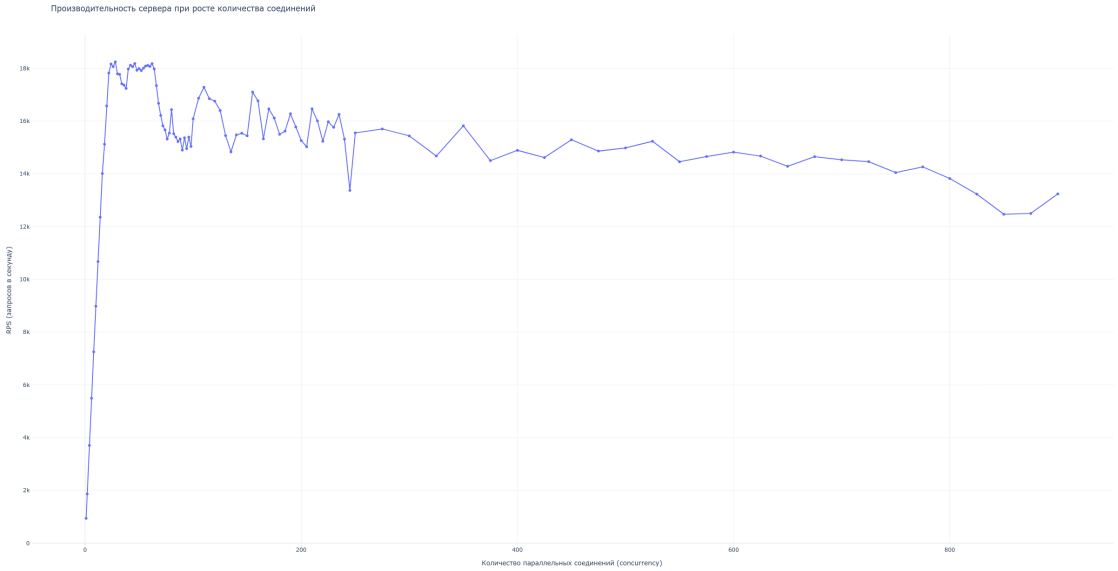

Рисунок 4.1 — График зависимости количества запросов в секунду в зависимости от числа одновременных запросов к серверу

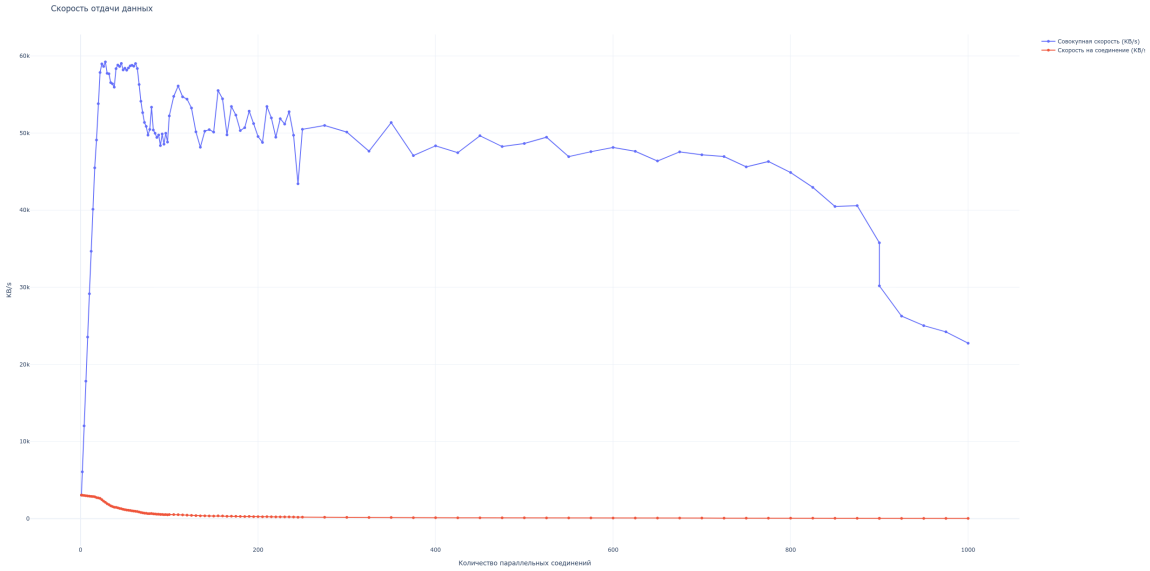

Рисунок 4.2 — Графики зависимости пропускной способности от числа одновременных запросов к серверу

Таким образом, падений производительности сервера в данной конфигурации началось при 800 запросах, а пропускная способность составила 50 МБайт/с при таких значениях.

ЗАКЛЮЧЕНИЕ

Цель данной работы: разработка статического веб-сервера – выполнена.

В результате работы были описаны требования к разрабатываемому приложению, системные вызовы UNIX, необходимые для разработки веб-сервера.

Была разработана архитектура многопоточного веб-сервера с мультиплексированием, lru-кешем файлов и асинхронным чтением. Рассмотрены состояния http-запроса, переходы между ними и алгоритмы их обработки. Был разработан сам веб-сервер на языке C. Проведено модульное тестирование разработанного веб-сервера с использованием фреймворка libcheck.

Было проведено нагрузочное тестирование, которое показало, что при 8 рабочих потоках веб-сервер выдерживает одновременное количество соединений не более 800 без существенной потери производительности, а также в среднем имеет пропускную способность 50 МБайт/с при данном количестве соединений

В ходе работы поставленные цель и задачи были выполнены.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Uniform Resource Locator (URL) // Yandex cloud URL: https://yandex.cloud/ru/docs/glossary/url (дата обращения: 04.12.2025).

2. Michael Kerrisk The linux programming interface. - San Francisco: No Starch Press, 2010

3. pthreads // Linux manual page URL: https://man7.org/linux/man-pages/man7/pthreads.7.html (дата обращения: 04.12.2025).

4. pselect // Linux manual page URL: https://man7.org/linux/man-pages/man3/pselect.3p.html (дата обращения: 04.12.2025).

5. fnctl // Linux manual page URL: https://man7.org/linux/man-pages/man2/fcntl.2.html (дата обращения: 04.12.2025).

6. read // Linux manual page URL: https://man7.org/linux/man-pages/man2/read.2.html (дата обращения: 04.12.2025).

7. write // Linux manual page URL: https://man7.org/linux/man-pages/man2/write.2.html (дата обращения: 04.12.2025).

8. stat // Linux manual page URL: https://man7.org/linux/man-pages/man2/stat.2.html (дата обращения: 04.12.2025).

9. PlantUML с первого взгляда // PlantUML URL: https://plantuml.com/ru (дата обращения: 04.12.2025).

10. Programming languages — C // INTERNATIONAL STANDARD ISO/IEC URL: https://www.open-std.org/jtc1/sc22/wg14/www/docs/n2310.pdf (дата обращения: 04.12.2025).

11. GCC, the GNU Compiler Collection // GCC URL: https://gcc.gnu.org/ (дата обращения: 04.12.2025).

12. GNU Make // GNU URL: https://www.gnu.org/software/make/ (дата обращения: 04.12.2025).

13. Unit Testing Framework for C // Check URL: https://libcheck.github.io/check/ (дата обращения: 04.12.2025).

14. ab - Apache HTTP server benchmarking tool // Apache URL: 04.12.2025 (дата обращения: https://httpd.apache.org/docs/current/programs/ab.html).

Приложение А