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»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

«Разработка сервера для отдачи статического содержимого с диска по протоколу HTTP»

Студент **ИУ7-75Б** _____ **Ф. Х. Равашдех**

(Подпись, дата) (И.О.Фамилия)

Руководитель _____ **М. Д. Мицевич**

(Подпись, дата) (И.О.Фамилия)

2025 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ **5**

1 Аналитический раздел **6**

1.1 Протокол передачи гипертекста 6

1.2 Веб-сервер 6

1.3 Существующие решения 7

1.4 Сокеты 7

1.5 Мультиплексирование 8

1.6 Параллельная обработка запросов 9

2 Конструкторский раздел **11**

2.1 Алгоритм работы HTTP-сервера 11

2.2 Алгоритм работы модели prefork 12

2.3 Архитектура сервера 12

3 Технологический раздел **14**

3.1 Выбор инструментов 14

3.2 Структура проекта 14

3.3 Тестирование 14

3.3.1 Проверка обработки HTTP-запросов 14

3.3.2 Проверка корректности передачи данных 15

4 Исследовательский раздел **16**

4.1 Технические характеристики 16

4.2 Цель исследования 16

4.3 Результаты исследования 17

4.3.1 Максимальное количество обслуживаемых сетевых соединений 17

4.3.2 Скорость отдачи данных по каждому сетевому соединению и совокупная 18

ЗАКЛЮЧЕНИЕ 22

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 23

ВВЕДЕНИЕ

В настоящее время протокол HTTP является одним из основных протоколов прикладного уровня и широко используется для передачи данных в сети Интернет. Значительная часть веб-ресурсов представляет собой статическое содержимое в виде HTML-страницы, таблицы стилей, изображения и другие файлы, не требующие динамической генерации на стороне сервера. Эффективная передача такого содержимого является важной задачей при разработке сетевых приложений.

Целью данной курсовой работы является разработка сервера для отдачи статического содержимого с диска по протоколу HTTP.

Для достижения поставленной цели необходимо выполнить следующие задачи:

1) провести анализ предметной области;
2) спроектировать программное обеспечение;
3) реализовать спроектированное программное обеспечение;
4) провести исследование разработанного сервера на основе нагрузочного тестирования.

1 Аналитический раздел

В этом разделе будет перечислены теоретические сведения, необходимые для решения поставленной задачи.

1.1 Протокол передачи гипертекста

HTTP (HyperText Transfer Protocol, протокол передачи гипертекста) — протокол прикладного уровня, предназначенный для передачи данных между клиентом и сервером по модели «запрос–ответ». Протокол является текстовым и не хранит состояние (stateless). HTTP-сообщение состоит из стартовой строки, набора заголовков и, при необходимости, тела сообщения.

Методы, необходимые для отдачи статического содержимого:

1) **GET** используется для запроса ресурса, расположенного на сервере, при успешной обработке запроса сервер возвращает заголовки и тело ответа, содержащее запрошенный файл;
2) **HEAD** аналогичен методу `GET`, однако сервер возвращает только заголовки ответа без передачи тела сообщения, данный метод используется для получения информации о ресурсе без его загрузки.

В результате обработки запросов сервер возвращает следующие коды состояния HTTP:

1) **200 OK** — запрос выполнен успешно;
2) **403 Forbidden** — доступ к ресурсу запрещён;
3) **404 Not Found** — запрашиваемый ресурс не найден;
4) **405 Method Not Allowed** — метод запроса не поддерживается сервером.

1.2 Веб-сервер

Веб-сервер — это программное обеспечение, принимающий HTTP-запросы от клиентов и выдающий им HTTP-ответы. Веб-серверы обладают возможностью предоставлять клиентам доступ к статическому содержимому из хранилища сервера, для этого клиенту требуется отправлить запрос к веб-серверу, который находит необходимый файл, считывает его и отсылает клиенту. Веб-серверы также могут являтся связующим звеном между клиентами, например браузерами, и серверами-приложенями, проксируя трафик.

1.3 Существующие решения

Для отдачи статического содержимого в основном используются уже существующие решения.

Наиболее распространёнными веб-серверами являются:

1) Apache HTTP Server;
2) Nginx;
3) Lighttpd.

Apache HTTP Server — свободный веб-сервер, наиболее часто используемый в UNIX-подобных операционных системах. Основными достоинствами Apache считаются надёжность и гибкость конфигурации. Он позволяет подключать внешние модули для предоставления данных, использовать СУБД для аутентификации пользователей, модифицировать сообщения об ошибках и т. д.

Nginx — позиционируется производителем как простой, быстрый и надёжный сервер, не перегруженный функциями. Применение nginx целесообразно прежде всего для статических веб-сайтов и как обратного прокси-сервера перед динамическими сайтами.

Lighttpd — свободный веб-сервер, разрабатываемый с расчётом на скорость и защищённость, соответствие стандартам и с небольшим размером. Сложнее в конфигурации чем другие. Часто используется во встроенных системах, например в маршрутизаторах.

В рамках курсовой работы разрабатывается упрощённый HTTP-сервер, реализующий только необходимую функциональность, что позволит на практике изучить принципы работы веб-серверов.

1.4 Сокеты

Сокет — это абстракция конечной точки соединения, которая используется для обеспечения обмена данными между устройствами сети. Сокеты являются ключевым компонентом для установки и управления сетевыми соединениями. Сокеты предоставляют интерфейс для создания конечных точек соединения в сети с использованием протоколов передачи данных. При разработке веб-серверов сокеты используются для «прослушивания» входящих соединений от клиентов и передачи данных между сервером и клиентами.

Процесс создания сервера с использованием сокетов включает в себя следующие шаги.

1) создание сокета, который будет слушать входящие соединения;
2) привязка сокета к адресу и порту, на котором будет прослушиваться входящий трафик;
3) установка сервера в состояние прослушивания входящих соединений;
4) принятие входящего соединения;
5) обработка запросов и передача данных.

Использование сокетов в разработке серверов позволяет эффективно управлять сетевыми соединениями и обеспечивать передачу статического контента клиентам по сети.

1.5 Мультиплексирование

Мультиплексирование ввода-вывода является методом обработки нескольких операций ввода-вывода в одном потоке выполнения программы, что позволяет повысить эффективность управления множеством соединений в сетевых приложениях или приложениях с асинхронным вводом-выводом. Это позволяет уменьшить задержки и повысить производительность.

Мультиплексоры позволяют приложению ожидать ввода или вывода данных из нескольких источников, таких как сокеты, файлы или сетевые устройства, используя один системный вызов вместо создания или управления каждым потоком или процессом отдельно.

Использование мультиплексирования ввода-вывода требует более сложной логики обработки событий, чем простое параллельное программирование, но оно может обеспечить более эффективное использование системных ресурсов и более высокую производительность.

Существуют следующие механизмы мультиплексирования сетевых соединений:

1) select;
2) pselect;
3) poll;
4) epoll (kqueue).

Вызов **select** — это системный вызов, используемый в различных операционных системах для ожидания событий на нескольких файловых дескрипторах, таких как сокеты [4]. Он позволяет одному потоку контролировать несколько соединений одновременно. Данный мультиплексор имеет ограничение на максимальное количество отслеживаемых файловых дескрипторов (1024).

Вызов **pselect** схож по принципу работы с select и имеет те же ограничения, од- нако реализует более продвинутую обработку сигналов. В отличие от select, pselect позволяет процессу заблокироваться на ожидании событий ввода-вывода и одновременно игнорировать определенные сигналы или обрабатывать их в специфическим способом.

Вызов **poll** — это более новый метод опроса сокетов, созданный после того, как люди начали пытаться писать большие и высоконагруженные сетевые сервисы. Он спроектирован намного лучше и не страдает от большинства недостатков метода select, например от ограничения количества наблюдаемых дескрипторов.

Вызов **epoll (kqueue)** — истемный вызов, предоставляемый в ядре Linux, обеспечивает более гибкие варианты работы с событиями ввода-вывода за счёт реализации модели уведомлений. В отличие от select и pselect, epoll предоставляет более эффективный способ мониторинга множества файловых дескрипторов на предмет готовности к вводу или выводу и не имеет ограничений на обработку файловых дескрипторов. Является наиболее современным и эффективным механизмом в сравнении с select и pselect и часто используется в высоконагруженных сетевых приложениях на ОС Linux [7].

1.6 Параллельная обработка запросов

Для ускорения обработки запросов веб-серверы реализуют их параллельную обработку в многопроцессорной среде.

В данной работе будут несколько основных существующих подходов к параллелизации работы веб-серверов:

1) пул потоков (thread pool);
2) разветвление (prefork)

Thread Pool представляет собой набор потоков, готовых к выполнению задач. Когда в систему поступает новая задача, она помещается в очередь, и

один из доступных потоков в пуле забирает эту задачу на выполнение. После завершения задачи поток возвращается обратно в пул и становится доступным для выполнения новых задач. Это способствует уменьшению накладных расходов на создание и уничтожение потоков, а также позволяет управлять ресурсами более эффективно и обеспечивает параллельное выполнение задач.

Prefork — это модель параллелизации, при которой веб-сервер создаёт отдельные процессы для обработки запросов. Такой подход позволяет изолировать обработку каждого запроса и повышает надежность веб-сервера, так как сбои в одном процессе не влияют на остальные. Однако создание процессов требует больше дополнительных ресурсов, чем использование пула потоков.

Выбор между подходами зависит от конкретных требований веб-сервера, предполагаемой нагрузки и характера обрабатываемых запросов.

Вывод

В этом разделе были перечислены теоретические сведения, необходимые для решения поставленной задачи, такие как протокол передачи гипертекста, веб-сервер и его существующие решения, сокеты и механизмы мультиплексирования.

2 Конструкторский раздел

В данном разделе будут рассмотрены алгоритмы работы сервера и работы prefork.

2.1 Алгоритм работы HTTP-сервера

Алгоритм работы сервера для отдачи статического содержимого по протоколу HTTP включает следующие основные этапы:

1) Инициализация серверного сокета: создание сокета, установка параметров, привязка к заданному порту и перевод сокета в режим прослушивания.
2) Инициализация структуры для хранения активных клиентских соединений.
3) Ожидание сетевых событий с использованием механизма мультиплексирования ввода-вывода.
4) При появлении события на слушающем сокете сервер принимает новое клиентское соединение и переводит соответствующий сокет в неблокирующий режим.
5) При готовности клиентского сокета к чтению сервер считывает входные данные и анализирует метод и путь к запрашиваемому ресурсу.
6) В случае неподдерживаемого метода формируется ответ с кодом состояния 405.
7) При запросе поддерживаемого метода определяется путь к файлу в каталоге статического содержимого и выполняется проверка существования и прав доступа к ресурсу.
8) Если файл не найден, формируется ответ с кодом состояния 404; при отсутствии прав доступа — ответ с кодом состояния 403.
9) При успешной проверке формируется HTTP-ответ с кодом состояния 200 и заголовками ответа.
10) При использовании метода GET сервер передаёт содержимое файла клиенту частями, учитывая ограничения размера буфера.
11) При использовании метода HEAD сервер передаёт только заголовки ответа без тела сообщения.

12) После завершения передачи данных сервер закрывает клиентское соединение и освобождает ресурсы.

13) Сервер возвращается к ожиданию новых сетевых событий.

2.2 Алгоритм работы модели prefork

Алгоритм работы сервера с использованием модели prefork состоит из следующих этапов:

1) Основной процесс сервера выполняет инициализацию сетевого окружения и создаёт слушающий сокет.

2) После успешной инициализации основной процесс создаёт фиксированное количество дочерних процессов с помощью системного вызова `fork`.

3) Каждый дочерний процесс наследует слушающий сокет и переходит в основной цикл обработки клиентских соединений.

4) Дочерние процессы независимо друг от друга ожидают сетевые события и принимают входящие соединения.

5) После принятия соединения дочерний процесс обрабатывает запрос клиента в соответствии с алгоритмом работы HTTP-сервера.

6) Каждый дочерний процесс может одновременно обслуживать несколько клиентских соединений с использованием механизма мультиплексирования ввода-вывода.

7) Основной процесс после создания дочерних процессов переходит в режим ожидания и не участвует в обработке клиентских запросов.

8) В случае завершения работы одного из дочерних процессов основной процесс может создать новый процесс для поддержания заданного количества рабочих процессов.

2.3 Архитектура сервера

Разрабатываемый HTTP-сервер построен на основе архитектуры prefork, при которой главный процесс сервера отвечает за инициализацию и создание набора дочерних процессов, предназначенных для обработки клиентских соединений. Каждый дочерний процесс функционирует независимо и готов принимать входящие соединения от клиентов.

Для обслуживания нескольких сетевых соединений в рамках одного процесса используется механизм мультиплексирования ввода-вывода select(), позволяющий отслеживать состояние множества файловых дескрипторов и реагировать на события ввода без блокировки выполнения процесса. Такой подход позволяет одному процессу эффективно обрабатывать несколько клиентских запросов одновременно.

Выбранная архитектура обеспечивает изоляцию процессов, повышает надёжность сервера и упрощает обработку ошибок, так как аварийное завершение одного дочернего процесса не приводит к остановке всего приложения. Кроме того, использование стандартных POSIX-механизмов обеспечивает переносимость решения между UNIX-подобными операционными системами.

Вывод

В данном разделе были рассмотрены алгоритмы рассмотрены алгоритмы работы сервера и работы prefork, а также архитектура сервера.

3 Технологический раздел

3.1 Выбор инструментов

Разработанный HTTP-сервер реализован на языке C++ с использованием стандартных POSIX-механизмов для сетевого взаимодействия и мультиплексорной обработки соединений. В качестве компилятора использовался clang 15.0.0 с поддержкой стандартов языка C++17 [6]. Разработка велась в редакторе Visual Studio Code с настройками для C/C++.

3.2 Структура проекта

Проект разделён на модули.

1) main.cpp — очка входа, создание слушающего сокета и запуск процессов;
2) http_server.cpp/h — обработка соединений, мультиплексирование через select() [1], разбор HTTP-запросов, формирование ответов, поддержка GET/HEAD и кодов 200, 403, 404, 405 [2];
3) log.cpp/h — логирование запросов с указанием метода, пути и кода ответа [3].

3.3 Тестирование

В рамках проверки корректности работы разработанного HTTP-сервера было проведено функциональное тестирование, включающее проверку обработки HTTP-методов, кодов ответа сервера, а также корректности передачи данных.

3.3.1 Проверка обработки HTTP-запросов

Для проверки базовой функциональности сервера использовалась утилита `curl`. При выполнении HTTP GET-запроса к корневому ресурсу сервера (`/`) сервер корректно вернул код ответа `200 OK` и передал содержимое главной HTML-страницы. Полученный результат подтверждает корректную обработку запросов на получение существующего ресурса.

Для проверки поддержки метода HEAD был выполнен запрос заголовков

без получения тела ответа. Сервер вернул код `200 OK` и корректный заголовок `Content-Length`, при этом тело ответа отсутствовало, что соответствует требованиям протокола HTTP.

При обращении к файлу, для которого отсутствуют права на чтение, сервер корректно определил невозможность доступа к ресурсу и вернул код ответа `403 Forbidden`. Это подтверждает корректную проверку прав доступа к файлам на стороне сервера.

Запрос несуществующего ресурса привёл к возврату кода ответа `404 Not Found`, что свидетельствует о корректной обработке ситуации отсутствия запрашиваемого файла в корневом каталоге сервера.

При попытке выполнения запроса с использованием неподдерживаемого HTTP-метода (POST) сервер вернул код ответа `405 Method Not Allowed`. Таким образом, сервер корректно ограничивает набор поддерживаемых методов и соответствует требованиям спецификации HTTP.

3.3.2 Проверка корректности передачи данных

Для проверки целостности передаваемых данных был выполнен тест загрузки файла размером 128 Мбайт. После загрузки файла контрольная сумма SHA-256 загруженного файла была сравнена с контрольной суммой исходного файла, расположенного на сервере. Полученные контрольные суммы полностью совпали, что подтверждает корректную передачу данных без потерь и искажений.

Вывод

В технологическом разделе описаны используемые инструменты и среда разработки, структура проекта, тестирование, что обеспечивает систематичное и эффективное покрытие всех сценариев работы сервера.

4 Исследовательский раздел

В данном разделе будет приведены результаты проведенного нагрузочного тестирования разработанного сервера по максимальному количеству обслуживаемых сетевых соединений и скорости отдачи данных по каждому сетевому соединению и совокупной.

4.1 Технические характеристики

Характеристики устройства, на котором выполнялись замеры [5]:

1) операционная система — macOS Sonoma 14.1 (23B2073);
2) процессор — Apple M3;
3) оперативная память — 16 Гб.

4.2 Цель исследования

Целью исследования является провести нагрузочное тестирование разработанного сервера:

1) Максимальное количество обслуживаемых сетевых соединений
2) Скорость отдачи данных по каждому сетевому соединению и совокупная

4.3 Результаты исследования

4.3.1 Максимальное количество обслуживаемых сетевых соединений

В таблице 4.1 представлены результаты нагрузочного тестирования максимальное количество обслуживаемых сетевых соединений для разного количества процессов prefork.

Таблица 4.1 — Нагрузочное тестирование сервера при различном числе процессов

Процессов	**Соединений**	**Успех**	**Неуспех**	**%**
2	16	16	0	100
2	32	32	0	100
2	64	64	0	100
2	128	128	0	100
2	256	128	128	50
2	512	130	382	25.3906
2	1024	245	779	23.9258
4	16	16	0	100
4	32	32	0	100
4	64	64	0	100
4	128	128	0	100
4	256	129	127	50.3906
4	512	128	384	25
4	1024	264	760	25.7812
8	16	16	0	100
8	32	32	0	100
8	64	64	0	100
8	128	128	0	100
8	256	128	128	50
8	512	166	346	32.4219
8	1024	383	641	37.4023

На рисунке 4.1 представлены графики зависимости процента успешных

соединений от количества соединений для различного числа процессов.

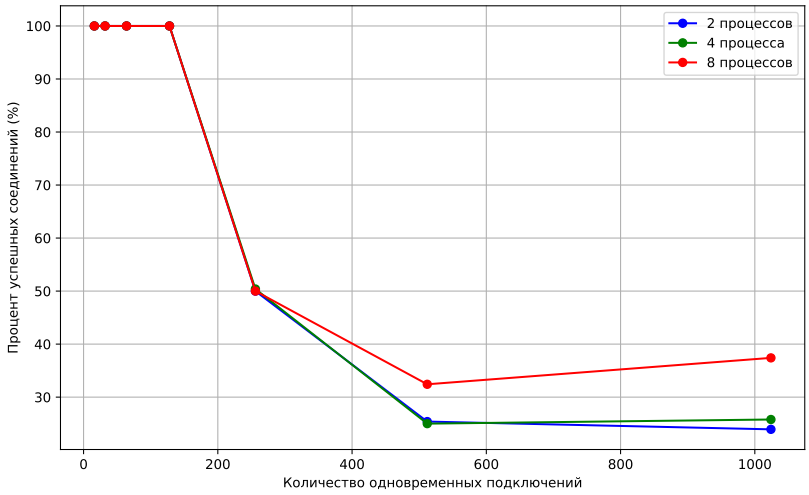

Рисунок 4.1 — Графики зависимости процента успешных соединений от количества соединений для различного числа процессов

4.3.2 Скорость отдачи данных по каждому сетевому соединению и совокупная

В таблице 4.2 представлены результаты нагрузочного тестирования скорости отдачи данных для различного числа процессов prefork.

На рисунке 4.2 представлены графики зависимости скорости отдачи данных от количества соединений для различного числа процессов на соединение.

На рисунке 4.3 представлены графики зависимости совокупной скорости отдачи данных от количества соединений для различного числа процессов.

Таблица 4.2 — Совокупная скорость отдачи данных сервера при различном числе процессов

Процессов	**Соединений**	**МБ**	**Время (с))**	**Скорость (МБ/с)**
2	4	4.00024	0.00650454	614.992
2	8	8.00048	0.00949479	842.618
2	16	16.001	0.015737	1016.77
2	32	32.0019	0.0232845	1374.39
2	64	64.0038	0.0367464	1741.77
2	100	100.006	0.0591705	1690.13
4	4	3.00018	0.00339233	884.4
4	8	8.00048	0.00708058	1129.92
4	16	16.001	0.0116171	1377.36
4	32	32.0019	0.0244327	1309.8
4	64	64.0038	0.0232089	2757.73
4	100	100.006	0.0418988	2386.85
8	4	4.00024	0.00475004	842.149
8	8	8.00048	0.006594	1213.3
8	16	16.001	0.0123947	1290.95
8	32	32.0019	0.0240008	1333.37
8	64	64.0038	0.0224389	2852.36
8	100	100.006	0.0381876	2618.81

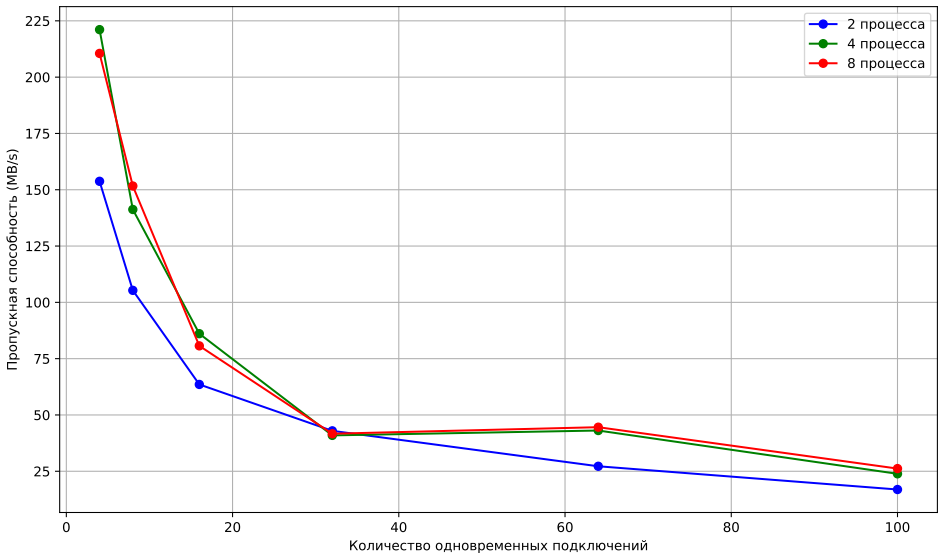

Рисунок 4.2 — Графики скорости отдачи данных сервера от количества соединений для различного числа процессов на соединение

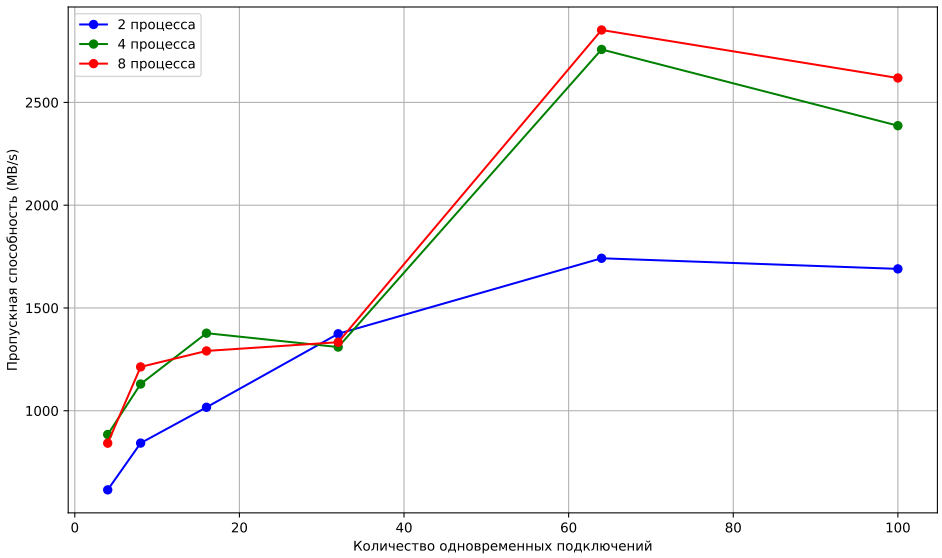

Рисунок 4.3 — Графики совокупной скорости отдачи данных сервера от количества соединений для различного числа процессов

Вывод

В результате проведённого нагрузочного тестирования была исследована совокупная скорость отдачи данных сервером при различном числе процессов prefork и различном количестве одновременных сетевых соединений.

Установлено, что при малом числе соединений увеличение количества процессов приводит к росту совокупной пропускной способности сервера. Так, при 4-16 соединениях сервер с 4 и 8 процессами демонстрирует более высокую скорость передачи данных по сравнению с конфигурацией из 2 процессов. Это связано с уменьшением времени ожидания обработки запросов и более эффективным использованием вычислительных ресурсов.

При увеличении числа соединений до 64 наблюдается максимальная совокупная скорость отдачи данных. Наиболее высокие значения достигаются при использовании 8 процессов prefork, где скорость передачи превышает 2800 МБ/s. Это указывает на то, что при достаточном уровне параллелизма сервер способен эффективно масштабироваться по числу обслуживаемых соединений.

При дальнейшем увеличении числа соединений до 100 рост совокупной скорости замедляется. Данный эффект объясняется ограничениями системных ресурсов, таких как пропускная способность сети, контекстные переключения между процессами и накладные расходы на обслуживание большого количества одновременных соединений.

Таким образом, увеличение числа процессов prefork положительно влияет на совокупную скорость отдачи данных до определённого предела.

ЗАКЛЮЧЕНИЕ

Цель работы достигнута: разработан сервер для отдачи статического содержимого с диска по протоколу HTTP. Можно сделать вывод, что увеличение числа процессов prefork положительно влияет на совокупную скорость отдачи данных до определённого предела.

Для достижения поставленной цели необходимо выполнить следующие задачи:

1) проведен анализ предметной области;
2) спроектировано программное обеспечение;
3) реализовано спроектированное программное обеспечение;
4) проведено исследование разработанного сервера на основе нагрузочного тестирования.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. The Open Group. select — synchronous I/O multiplexing. — 2018. — URL: https://pubs.opengroup.org/onlinepubs/9699919799/functions/select.html.

2. Fielding R., Nottingham M., Reschke J. HTTP Semantics. — 2022. — URL: https://www.rfc-editor.org/rfc/rfc9110.

3. Stevens W. R., Fenner B., Rudoff A. UNIX Network Programming, Volume 1: The Sockets Networking API. — Addison-Wesley, 2004.

4. Столяров А. В. Программирование: введение в профессию

5. Технические характеристики MacBook Pro [Электронный ресурс]. Режим доступа: https://support.apple.com/en-by/117736 (дата обращения 11.12.24)

6. Документация к языку C++ https://en.cppreference.com/w/

7. select / poll / epoll: практическая разница — URL: https://habr.com/ru/companies/infopulse/articles/415259/