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 ИУ «Информатика и системы управления»

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

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

«Разработка статического сервера»

Студент ИУ7-76Б _____ Авсюнин А. А.
(Подпись, дата) (Фамилия И. О.)

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

2023 г.

РЕФЕРАТ

Научно-исследовательская работа 22 с., 6 рис., 7 ист.

СТАТИЧЕСКИЙ СЕРВЕР, ЯЗЫК СИ, THREADPOOL, NGINX, APACHE BENCHMARK.

Цель работы: разработка статического веб-сервера для раздачи информации с жёсткого диска.

В данной работе рассматриваются методы работы статический серверов, разрабатывается статический сервер, основанный на шаблоне управления потоками threadpool и системном вызове select. Проводится сравнение разработанного сервера с сервером nginx при помощи apache benchmark.

Результаты: Разработанный сервер уступает nginx на примерно на 20% при одновременном обращении до 500 клиентов. С увеличением количества одновременных обращений время отклика разработанного сервера увеличивается медленнее, чем у сервера nginx. Это уменьшает разницу во времени до 5%.

Содержание

Введение

Для разработки любого веб-приложения необходим сервер, который отвечает за обработку и доставку файлов пользователю. Статический сервер предназначен для раздачи файлов различных форматов через протокол HTTP. Чаще всего это файлы форматов HTML, JS, CSS, однако возможно передавать и другие.

Целью данной работы является разработка статического сервера с использованием языка программирования Си без сторонних библиотек. Сервер должен использовать шаблон управления потоками threadpool и системный вызов select для мультиплексирования. Для достижения поставленной цели необходимо выполнить следующие задачи:

1) провести анализ предметной области;
2) провести анализ шаблона threadpool и системного вызова select;
3) спроектировать и разработать сервер раздачи статической информации;
4) провести сравнение результатов нагрузочного тестирования при помощи apache benchmark разработанного сервера с nginx;

1 Аналитическая раздел

В данном разделе проводится анализ предметной области, анализ шаблона управления потоками threadpool, модели асинхронного блокирующего ввода-вывода, системного вызова select, а также протокола передачи данных http. Формулируются требования к разрабатываемому приложению.

1.1 Статический сервер

Статический сервер — программа, принимающая запросы по протоколу http и возвращающая на них ответы со статической информацией.

Статическая информация — информация, которая вообще или редко подвергается изменениям. В данной работе предметом раздачи сервера будут файлы разных форматов:

1. HTML (от англ. HyperText Markup Language) — гипертекстовые документы;
2. CSS (от англ. Cascading Style Sheets) — файлы стилей;
3. JS (от англ. JavaScript) — файлы с кодом на языке java script;
4. PNG (от англ. Portable Network Graphics) — файлы растровых изображений;
5. JPEG (от англ. Join Photographic Experts Group) — файлы растровых изображений;
6. SWF (от англ. Small Web Format) — файлы векторной графики;
7. GIF (от англ. Graphics Interchange Format) — файлы растровых изображений;
8. TXT (от англ. text) — файлы с текстом.

1.2 Готовые решения

Так как любой веб-сайт нуждается в сервере, а с момента запуска сети Интернет прошло уже более 30 лет, то на данный момент существует множество

вариантов статических серверов.

NGINX (от англ. Engine X) [1] — это HTTP-сервер и обратный прокси-сервер, почтовый прокси-сервер, а также TCP/UDP прокси-сервер общего назначения, написанный Игорем Сысоевым, выпускником МГТУ им. Н.Э.Баумана. Согласно статистике Netcraft [2] nginx обслуживал или проксировал 20.72% самых нагруженных сайтов в декабре 2023 года.

Apache (от англ. a patchy server) [3] — веб-сервер с открытым исходным кодом. Является одним из первых решений в данной области и до появления nginx обслуживал до 70% всех приложений, отслеживаемых Netcraft [2]. Основным достоинством является гибкость конфигурации, позволяющая подключать внешние модули для предоставления данных, модифицировать данные об ошибках и т. д. На данный момент обслуживает около 22% веб-серверов.

1.3 Шаблон пула потоков

Пул потоков (англ. threadpool) — шаблон управления потоками для достижения параллелизма выполнения в компьютерной программе. Основывается он на создании набора (пула) потоков при запуске программы и распределения задач между ними в течение всей работы.

Для использования пула потоков необходимо:

1. создать рабочие потоки;
2. создать общую очередь задач, из которой рабочие потоки будут брать задачи на выполнение;
3. управлять общим доступом к очереди при помощи примитивов синхронизации.
4. предоставить внешний интерфейс для создания, удаления пула потоков, а также добавления работ в очередь.

На рисунке 1.1 представлена схема работы пула потоков.

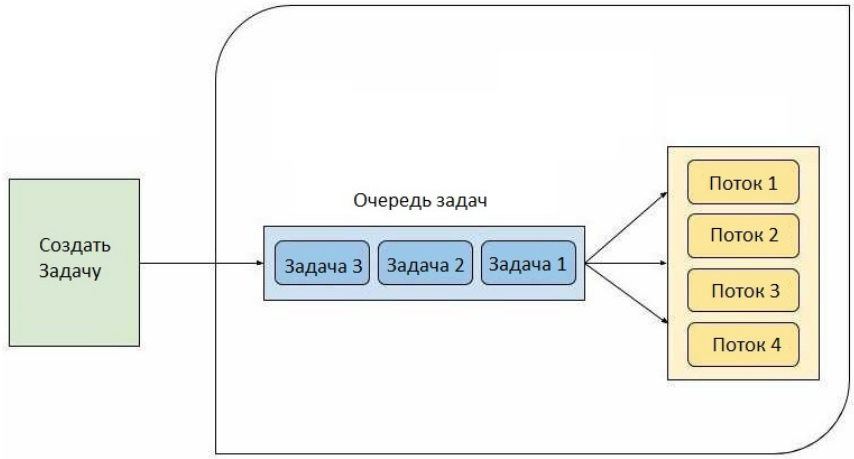

Рисунок 1.1 – Пул потоков

Применяя эту концепция к веб-серверу, главный поток будет соединяться с клиентом и передавать это соединение одному из выделенных потоков через очередь задач, где оно уже будет обработано и создан ответ на запрос. А основной поток в это время сможет дальше функционировать и принимать новых клиентов.

1.4 Мультиплексирование

Мультиплексирование — модель асинхронного блокирующего ввода-вывода. Основывается на опросе набора источников о готовности. Модель является блокирующей, так как главный процесс блокируется в ожидании готовности одного из источников. Асинхронность достигается за счёт того, что главный поток производит одновременный опрос сразу нескольких источников, и блокируется только до готовности одного из них.

1.5 Системный вызов select

Системный вызов select — функция, определённая стандартом POSIX, предназначенная для опроса файловых дескрипторов открытых каналов ввода-вывода. Заголовок данной функции определён в файле *sys/select.h* на языке программирования Си.

На рисунке 1.2 представлена реализация мультиплексирования при помощи функции select.

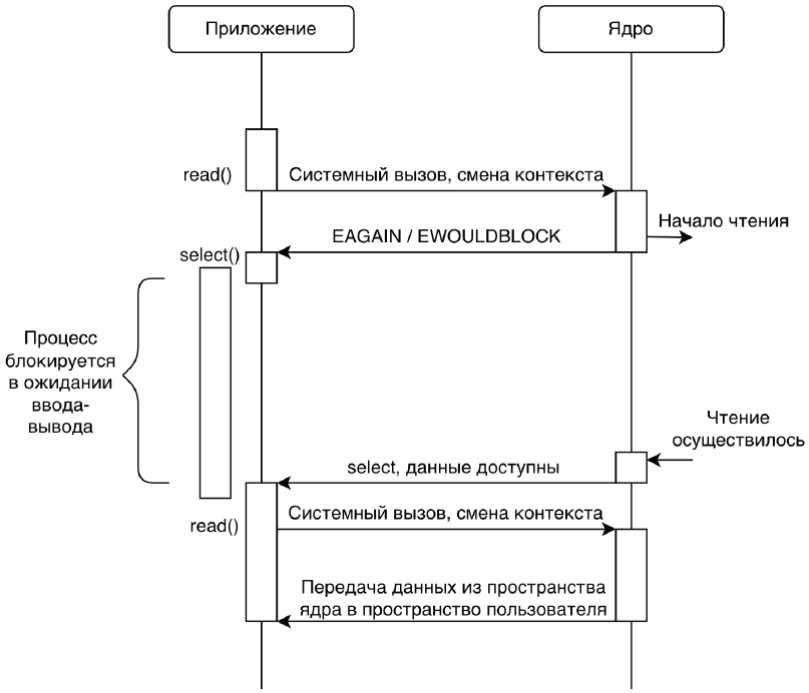

Рисунок 1.2 – Мультиплексирование с помощью select

Функция select принимает на вход 5 аргументов:

1. nfds — число, на единицу больше максимального файлового дескриптора.
2. readfds — набор файловых дескрипторов типа fd_set, предназначенный для опроса дескрипторов на чтение.
3. writefds — набор файловых дескрипторов типа fd_set, предназначенный для опроса дескрипторов на запись.
4. exceptfds — набор файловых дескрипторов типа fd_set, предназначенный для опроса дескрипторов на появление исключительных ситуаций.
5. timeout — структура времени типа timeval с секундами и микросекундами, в которой хранится время, которое процесс будет заблокирован.

Возвращает функция количество файловых дескрипторов, готовых к операции ввода-вывода.

1.6 HTTP

HTTP (от англ. HyperText Transfer Protocol) — протокол уровня приложений сетевой модели OSI [4], предложенной международной организацией

по стандартизации ISO [5]. Это текстовый протокол, изначально предназначенный для передачи гипертекстовых документов.

Каждое HTTP-сообщение состоит из трёх частей: стартовая строка, заголовки, тело сообщения. В стартовой строке указывается один из методов запроса: OPTIONS, GET, HEAD, PUT, POST, PATCH, DELETE, TRACE, CONNECT. В данной работе будут рассмотрены только два из них, GET и HEAD.

Метод GET используется для запроса содержимого ресурса. Запросы этого метода считаются идемпотентными, то есть на один и тот же запрос всегда выдаётся один и тот же ответ.

Метод HEAD аналогичен запросу GET, за исключением того, что в ответе на этот запрос отсутствует тело.

В заголовках HTTP-сообщения указываются определённые свойства и характеристики как тела сообщения, так и установленного соединения. Наример, один из заголовков, content-type, устанавливает тип данных, передаваемых в теле. Для каждого формата файла он свой:

1. HTML — text/html;
2. CSS — text/css;
3. JS — text/javascript;
4. PNG — image/png;
5. JPEG — image/jpeg;
6. SWF — application/x-shockwave-flash;
7. GIF — image/gif;
8. TXT — text/plain.

1.7 Требования к разрабатываемой программе

На основе задания в курсовой работе и вышеперечисленного разрабатываемая программа должна соответствовать следующим требованиям:

1. поддержка запросов GET и HEAD (поддержка статусов 200, 403, 404);

2. ответ на неподдерживаемые запросы статусом 405;
3. выставление content type в зависимости от типа файла (поддержка .html, .css, .js, .png, .jpg, .jpeg, .swf, .gif);
4. корректная передача файлов размером в 100мб;
5. сервер по умолчанию должен возвращать html-страницу на выбранную тему с css-стилем;
6. учесть минимальные требования к безопасности статик-серверов (предусмотреть ошибку в случае если адрес будет выходить за root директорию сервера);
7. реализовать логгер;
8. использовать язык Си. Сторонние библиотеки запрещены;
9. реализовать архитектуру с использованием threadpool и select;
10. статик сервер должен работать стабильно.

Вывод

В данном разделе был проведён анализ предметной области, анализ шаблона управления потоками threadpool, модели асинхронного блокирующего ввода-вывода, системного вызова select, а также протокола передачи данных http. Сформулированы требования к разрабатываемому приложению.

2 Конструкторский раздел

В данном разделе будут представлены схемы алгоритмов работы сервера и одного из потоков в пуле.

2.1 Разработка алгоритмов

На рисунке 2.1 представлена схема алгоритма работы сервера.

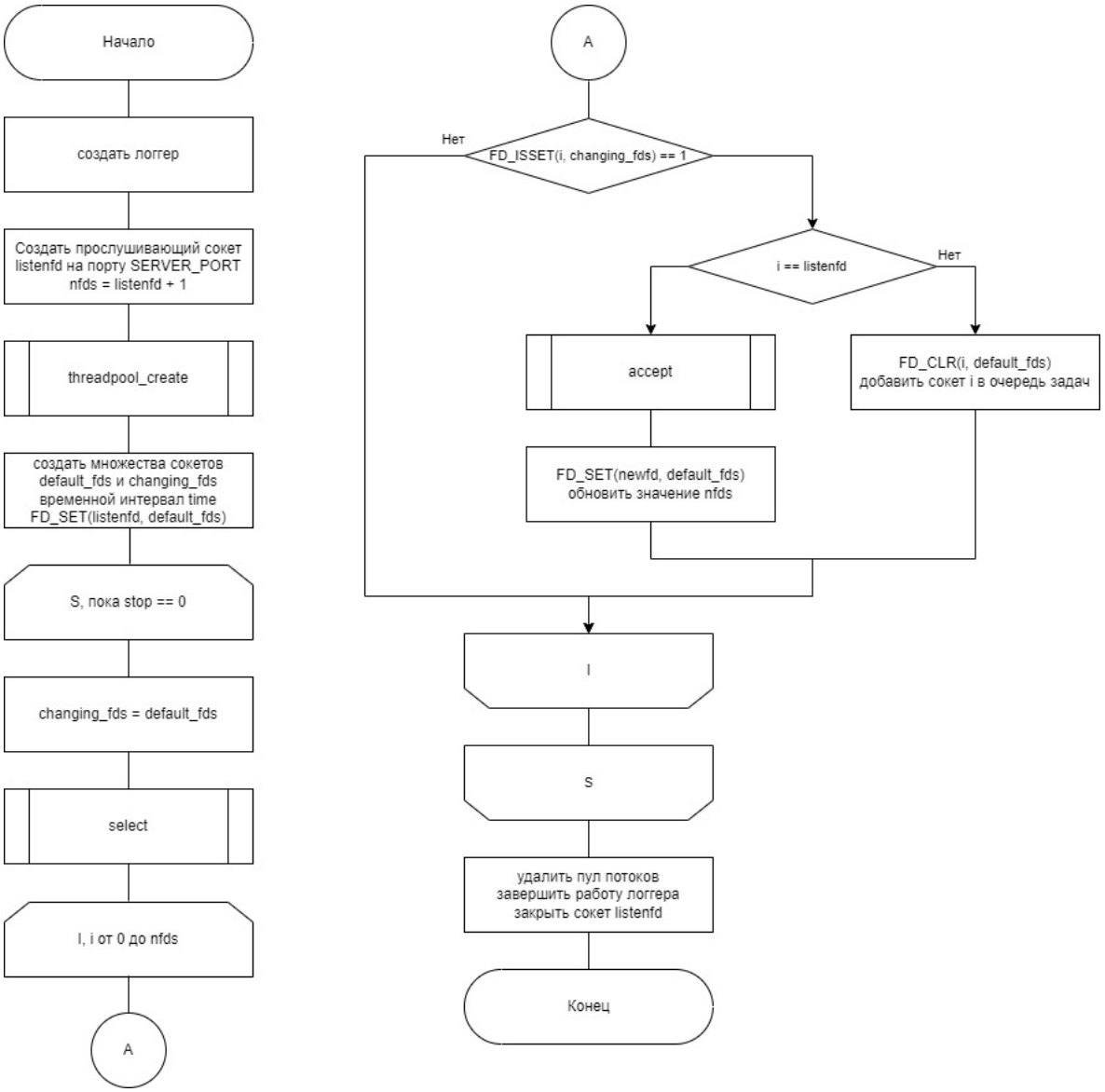

Рисунок 2.1 – Схема работы сервера

На рисунке 2.2 представлена схема алгоритма работы одного из потоков в пуле.

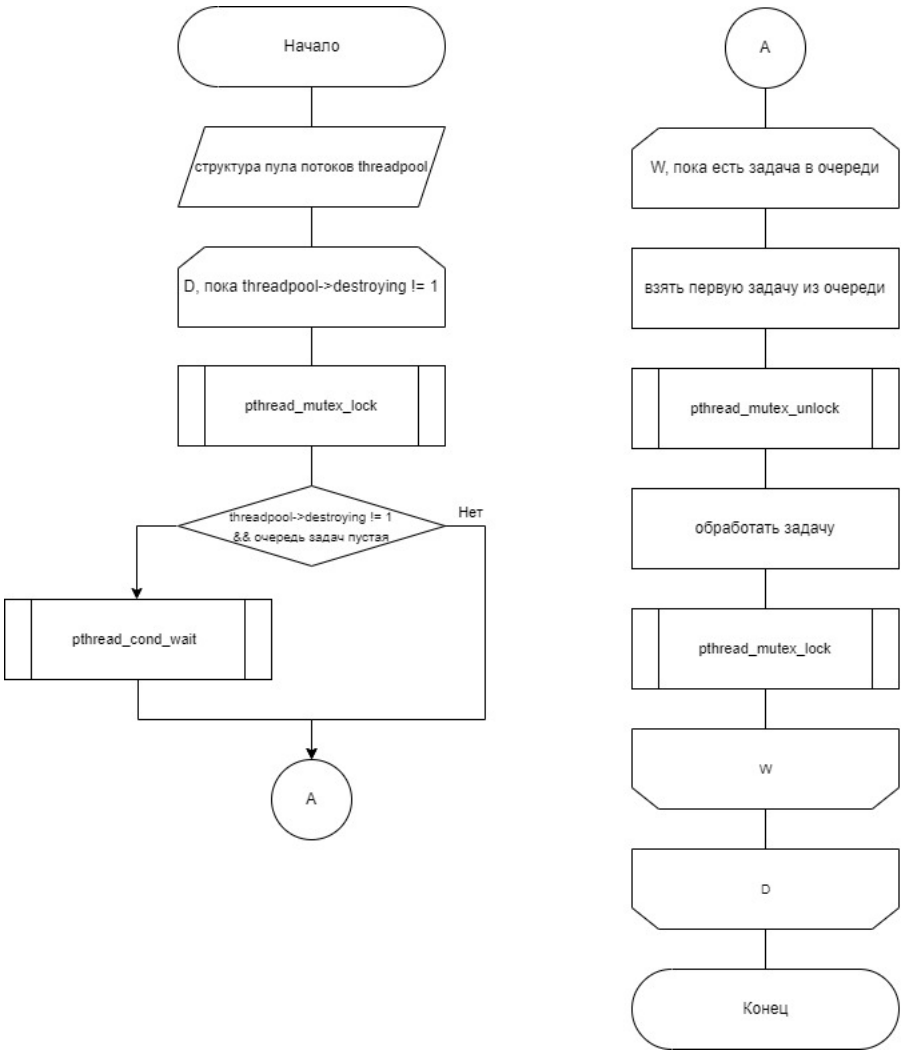

Рисунок 2.2 – Схема алгоритма работы потока из пула

Вывод

В данном разделе были представлены схемы алгоритмов работы сервера и одного из потоков в пуле.

3 Технологический раздел

В данном разделе будут приведены средства реализации программного обеспечения, рассмотрены модули программы, представлены листинги исходного кода.

3.1 Средства реализации

Для реализации ПО был выбран язык C, согласно требованиям к выполнению курсовой работы.

3.2 Структура программы

Программа разделена на несколько модулей, представленных в виде отдельных файлов.

1. Main — модуль, запускающий сервер.
2. Server — модуль, предоставляющий основные функции работы сервера.
3. Threadpool — модуль, предоставляющий функции работы с пулом потоков.
4. Logger — модуль, предосталяющий функции логирования.

3.3 Реализация

В листингах 3.1-3.3 представлена реализация основного алгоритма работы сервера. А в листингах 3.4-3.5 представлена реализация обработки запросов на сервер.

Листинг 3.1 – Основной алгоритм сервера (часть 1)

```
1 void run_server(char *filename)
2 {
3     signal(SIGINT, close_server);
4     signal(SIGPIPE, SIG_IGN);
5     logger_create(filename);
```

Листинг 3.2 – Основной алгоритм сервера (часть 2)

```
1     long number_of_processors = sysconf(_SC_NPROCESSORS_ONLN);
2     char log_buffer[BUFFER_LEN * 2 + 1] = {0};
3     setbuf(stdout, NULL);
4
5     int listenfd = 0;
6     if ((listenfd = get_listen()) == -1)
7     return;
8
9     threadpool_t *threadpool = threadpool_create(
          number_of_processors);
10    if (!threadpool)
11    {
12        close(listenfd);
13        return;
14    }
15
16    fd_set default_fds, changing_fds;
17    FD_ZERO(&default_fds);
18    FD_ZERO(&changing_fds);
19    FD_SET(listenfd, &default_fds);
20    int nfds = listenfd + 1;
21
22    struct timeval time;
23    while (!stop)
24    {
25        changing_fds = default_fds;
26        time.tv_sec = 30;
27        time.tv_usec = 0;
28        if ((select(nfds, &changing_fds, NULL, NULL, &time)) == -1)
29        {
30            continue;
31        }
32        for (int i = 0; i < nfds; ++i)
33        {
34            if (FD_ISSET(i, &changing_fds))
35            {
36                if (i == listenfd)
37                {
38                    int newfd;
39                    if ((newfd = accept(listenfd, NULL, NULL))!=-1)
```

Листинг 3.3 – Основной алгоритм сервера (часть 3)

```
                    {
                        FD_SET(newfd, &default_fds);
                        if (newfd >= nfds)
                        nfds = newfd + 1;
                    }
                }
                else
                {
                    FD_CLR(i, &default_fds);
                    threadpool_work_add(threadpool, http_handler, (
                        void *) i);
                }
            }
        }
    }
    threadpool_destroy(threadpool);
    close(listenfd);
    logger_destroy();
}
```

Листинг 3.4 – Обработка запроса на сервер (часть 1)

```
void http_handler(void *args)
{
    int socket = (int) args;

    char header[BUFFER_LEN] = {0};
    char path[BUFFER_LEN + 1] = {0};

    char request_data[MAX_GET_REQUEST + 1] = {0};
    long bytes = read(socket, request_data, MAX_GET_REQUEST);
    char log_buffer[MAX_GET_REQUEST * 2 + 1] = {0};
    if (bytes > 0)
    {
        http_request_t request_type = GET;
        http_response_t response_type = parse_request_header(&
            request_type, request_data, path);
        if (response_type != OK \
            || (response_type = parse_path(path)) != OK)
        {
```

Листинг 3.5 – Обработка запроса на сервер (часть 2)

```
            memset(path, 0, BUFFER_LEN + 1);
            get_default_path(path, response_type);
        }
        int header_len = get_response_header(header, response_type);
        char content_type_header[BUFFER_LEN + 1] = {0};
        int content_type_len = get_content_type_header(
           content_type_header, path);
        memcpy(header + header_len, content_type_header,
           content_type_len);
        header_len += content_type_len;
        int asd = write(socket, header, header_len);

        FILE *file;
        if (request_type == GET && (file = fopen(path, "r")))
        {
            char data[BUFFER_LEN] = {0};
            size_t number_of_data = 0;
            while ((number_of_data = fread(data, sizeof(char),
               BUFFER_LEN, file)))
            write(socket, data, number_of_data);
            fclose(file);
        }
    }
    close(socket);
}
```

3.4 Пример работы программы

На рисунке 3.1 представлен пример работы программы, а именно основная страница, отображаемая при обращении к серверу.

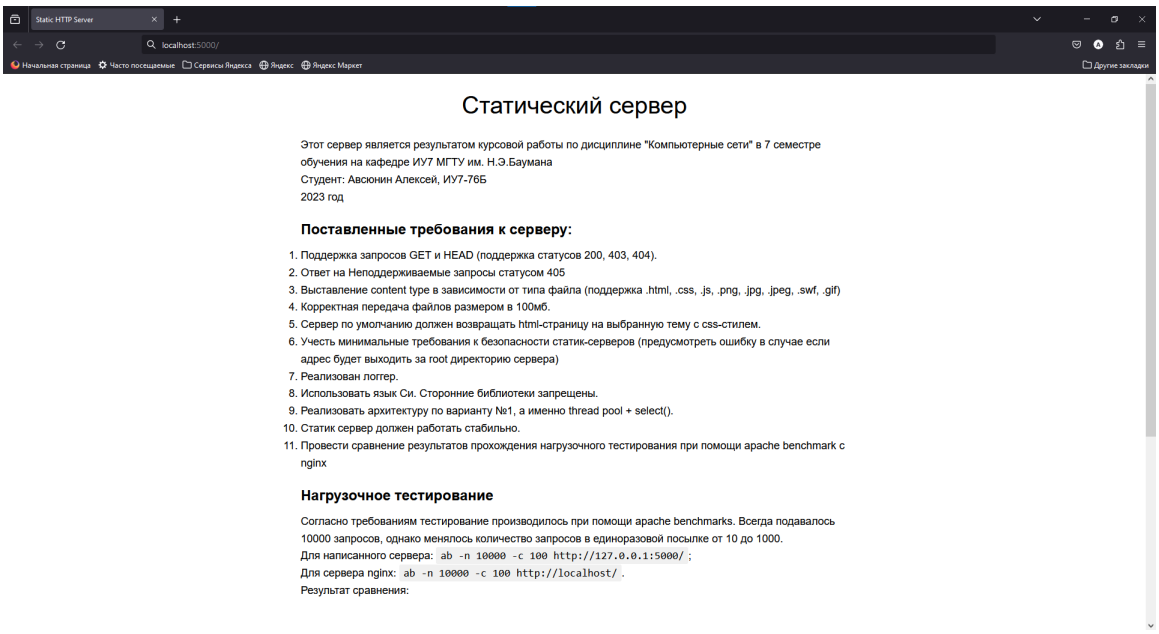

Рисунок 3.1 – Пример работы программы

Вывод

В этом разделе был выбран язык программирования, рассмотрены модули программы, представлены листинги исходного кода.

4 Исследовательский раздел

В данном разделе будет проведено нагрузочное тестирование разработанного ПО и сервера nginx при помощи Apache Benchmark и сравнение результатов.

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялись измерения:

1. операционная система Ubuntu, 20.04.4 [6];
2. память 8 ГБ;
3. процессор 2,4 ГГц 4-ядерный процессор Intel Core i5-1135G7 [7].

Во время замеров ноутбук был включен в сеть электропитания, нагружен только встроенными приложениями окружения и разработанным сервером.

4.2 Нагрузочное тестирование

Нагрузочное тестирование проводилось при помощи утилиты apache benchmark. На главную страницу сервера посылалось единовременно различное количество запросов от 10 до 1000. Рассматривалось среднее время обработки одного запроса.

Команда для единовременной отправки 100 запросов на разработанный сервер:
ab -n 10000 -c 100 http://localhost:5000.

Команда для единовременной отправки 100 запросов на сервер nginx:
ab -n 10000 -c 100 http://localhost.

На рисунке 4.1 представлены результаты измерений.

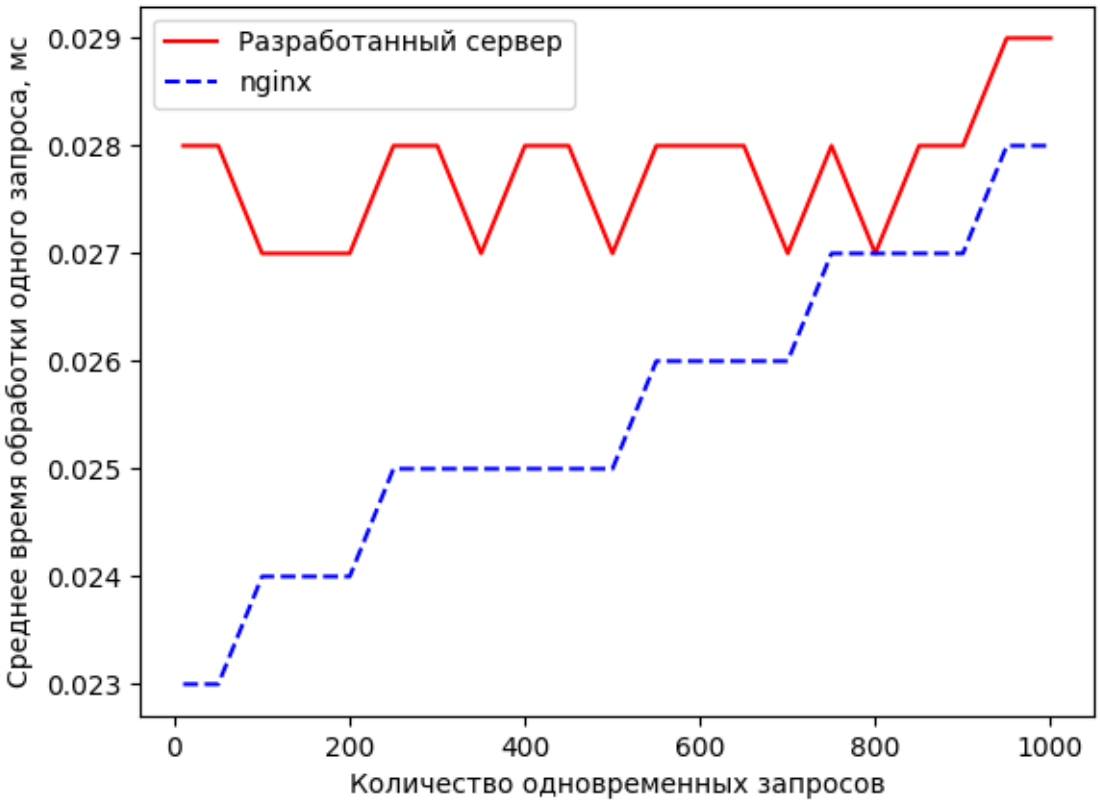

Рисунок 4.1 – Результаты тетсирования

По результатам тестирования разработанный сервер уступает nginx на 20% при одновременном обращении до 500 клиентов. С увеличением количества одновременных обращений среднее время выдачи ответа у разработанного сервера растёт медленнее, чем у сервера nginx и к 1000 клиентам разница уже составляет около 5% или 1 микросекунду.

Вывод

В данном разделе было проведено нагрузочное тестирование разработанного ПО и сервера nginx при помощи Apache Benchmark и сравнение результатов. По результатам тестирования разработанный сервер немного уступает в среднем времени обработки запроса серверу nginx.

Заключение

Цель, поставленная в начале, была достигнута: разработан статический сервер с использованием языка программирования Си без сторонних библиотек с использованием шаблона управления потоками threadpool и системного вызова select для мультиплексирования.

В ходе выполнения курсовой работы были решены следующие задачи:

1) проведён анализ предметной области;

2) проведён анализ шаблона threadpool и системного вызова select;

3) спроектирован и разработан сервер раздачи статической информации;

4) проведено сравнение результатов нагрузочного тестирования при помощи apache benchmark разработанного сервера с nginx;

Список использованных источников

1. Главная страница NGINX [Электронный ресурс]. Режим доступа: https://nginx.org/ru/ (дата обращения: 15.12.2023).

2. December 2023 Web Server Survey [Электронный ресурс]. Режим доступа: https://www.netcraft.com/blog/december-2023-web-server-survey/ (дата обращения: 15.12.2023).

3. Главная страница Apache [Электронный ресурс]. Режим доступа: https://www.apache.org/ (дата обращения: 15.12.2023).

4. Ю.А. Семенов. Протоколы Internet. Энциклопедия. — Москва: «Горячая линия — Телеком», 2003.

5. International Organization for Standardization [Электронный ресурс]. Режим доступа: https://www.iso.org/home.html (дата обращения: 15.12.2023).

6. Ubuntu [Электронный ресурс]. Режим доступа:https://ubuntu.com/ (дата обращения: 15.12.2023).

7. Процессор Intel® Core™ i5 [Электронный ресурс]. Режим доступа: https://www.intel.com/processors/core/i5/docs (дата обращения: 15.12.2023).