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 «**Информатика и системы управления**»

КАФЕДРА «**Программное обеспечение ЭВМ и информационные технологии**»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

«Разработка статического сервера»

Студент группы **ИУ7-76Б** _____ **П. А. Калашков**

(Подпись, дата) (И.О. Фамилия)

Руководитель курсовой работы _____ _____

(Подпись, дата) (И.О. Фамилия)

***2023** г.*

РЕФЕРАТ

Расчетно-пояснительная записка 33 с., 8 рис., 7 ист.

СЕРВЕР РАЗДАЧИ СТАТИЧЕСКОЙ ИНФОРМАЦИИ, HTTP, НАГРУЗОЧНОЕ ТЕСТИРОВАНИЕ, NGINX

Цель работы — разработка сервера раздачи статческой информации на языке Си без использования сторонних библиотек, построенного при помощи паттерна thread pool и использующего системный вызов pselect.

Результаты: разработанная программа предназначена для раздачи статической информации в несколько потоков, реализовано логирование запросов (в отдельном потоке), обработка некоторых типов файлов, поддержка статусов HTTP.

Проведено сравнение результатов нагрузочного тестирования (приведённого при помощи Apache Benchmarks) разработанной программы с NGINX.

Исходя из полученных данных, при небольшом (до 100) количестве клиентов разработанная программа показывает меньшее среднее время обработки запросов, чем NGINX, при этом чем меньше клиентов подсоединено к серверу, тем больше эта разница (до 7 раз при 5 клиентах).

СОДЕРЖАНИЕ

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

В работе используются следующие термины с соответствующими определениями:

HTML — Hypertext Markdown Language, язык гипертексовой разметки

CSS — Cascading Style Sheet, каскадные таблицы стилей

JS — JavaScript

PNG — Portable Network Graphics

JPEG (JPG) — Joint Photographic Experts Group

SWF — Small Web Format

GIF — Graphics Interchange Format

HTTP — Hypertext Transfer Protocol, протокол передачи гипертекста

ВВЕДЕНИЕ

По данным DatePortal, к январю 2022 года в Росссийской Федерации насчитывалось 129.8 миллионов интернет-пользователей, примерно 89% популяции [1]. При этом количество доменов в области российского интернета к декабрю 2022 года составляет 4.93 миллиона, при этом на большинстве этих сайтов (73.8%) зарегистрированы физические лица и индивидуальные предприниматели [2]. При этом объём трафика в рунете к 2022 году составил 91 эксабайт (квинтиллион байтов), а к концу 2023 года превысит 100 экабайт [3].

В большинстве современных веб-приложений используются такие технологии, как HTML, JS и CSS, часто используются различные форматы файлов для отображения информации определённого типа (например, PNG, JPG или JPEG для изображений).

Целью работы является разработка сервера раздачи статческой информации на языке Си без использования сторонних библиотек, построенного при помощи паттерна *thread pool* и использующего системный вызов *pselect*. Для достижения поставленной цели необходимо выполнить следующие задачи:

1) провести анализ предметной области;
2) определить функционал, реализуемый сервером раздачи статической инфоормации;
3) провести анализ паттерна *thread pool* и системного вызова *pselect* ;
4) спроектировать и разработать сервер раздачи статическй информации;
5) провести сравнение результатов нагрузочного тестирования при помощи Apache Benchmarks с NGINX.

1 Аналитическая часть

В данном разделе проводится анализ предметной области, анализ паттерна *thread pool*, асинхронной блокирующей модели ввода-вывда и системного вызова *pselect*, а также протокол HTTP и формализация требований к разрабатываемой программе.

1.1 Статическая информация

Статическая информация — информация, которая редко меняется с течением времени. В данной работе под статической информацией будем подразумевать файлы определённых форматоов, которые, предположительно, редко меняются с течением времени.

Согласно поставленной задаче, разрабатываемый сервер разадчи статической инфоормации должен предоставлять доступ к файлам следующих форматов:

1) файлы HTML (имеющие расширение .html), содержащие гипертекстовые документы;
2) скрипты JS (имеющие расширение .js);
3) файлы стилей CSS (имеющие расширение .css) ;
4) файлы изображений форматов PNG и JPEG (имеющие расширения .png и .jpeg или .jpg соответственно);
5) файлы SWF для векторной анимации (имеющие расширение .swf);
6) файлы GIF, содержащие графические изображения или анимации (имеющие расширение .gif).

Также необходимо учесть, что рассмотренные файлы могут иметь размер до нескольких сотен мегабайт, и передача таких файлов должна осуществляться корректно.

1.2 Существующие решения

Поскольку проблема раздачи статической информации возникла вместе с появлением интернета, к 2023 году существует набор готвых решений, как уни-

версальных, так и направленных на соответствие определённым технологиям.

NGINX (от англ. *Engine X*) — веб-сервер и прокси-сервер, работающий как на Unix-подобных системах, так и на системах Miscrosoft Windows (с версии 0.7.72). Разработан в 2004 году российским программистом Игорем Сысоевым, выпускником МГТУ им. Н. Э. Баумана. NGINX позиционируется производителем как простой, быстрый и надёжный сервер, использование корого целесообразно в том числе для раздачи статической информации. Помимо раздачи статической информации, NGINX предоставляет возможности кэширования запросов, сжатия при помощи технологии gzip, балансировки нагрузки между серверами, а также многие другие функции [4]. По данным Netcrafts, к ноябрю 2023 года NGINX используется для поддержки 249 миллионов сайтов, что составляет большую часть (22.83%) опрошенных сайтов [5].

Apache (от англ. *a patchy server*) — ближайший конкурент NGINX в области веб-серверов, является веб-сервером с открытым исходным кодом, поддержкивает такие операционные системы, как Linux, BSD, macOS и Microsoft Windows. С апреля 1996 года и по июль 2016 года являлся самым популярным веб-сервером в мире, поскольку с момента появления интернета являлся самым надёжным сервером, сейчас же используется 22.74% сайтов, лишь немнго уступая NGINX [5].

Cloudflare — американская компания, предоставляющая услуги по раздаче статической информации, защите от атак, предоставлению серверов DNS и проксированию сайтов. Услугами данной компании пользуются 10.62% опрошенных (115 миллионов сайтов), при этом раздача статической информации не стоит у компании на первом месте, отдаётся приоритет защите от DDoS атак: так, в 2014 году Cloudflare выдержала атаку мощностью 400 Гбит / с.

1.3 Паттерн пула потоков

Пул потоков (англ. *thread pool*) — архитектурный паттерн, предназначенный для многопоточной обработи информации [6]. В то время как создавать новый поток под новый запрос является неэффективно, предлагается создать

набор потоков (пул) определённй величины, и поддерживать его на протяжении работы программы. Когда на сервер приходит запрос, предлагается использовать свободный поток из пула потоков. В данном потоке будет произведена обработка пришедшего запроса, после чего поток будет возвращён в пул потоков, готовый к обработке дальнейших запросов.

Исследования [6] показали, что использование пула потоков может значительно улучшить производительность системы и уменьшить время обработки запросов. Тем не менее, существуют следующие проблемы, возникающие при использовании пула потоков:

— определение числа потоков (размера пула);
— контроль целостности очереди и предотвращения обработки одного запроса двумя потоками;
— оптимальное использование вычислительных ресурсов и предотвращение ”простаивания”потоков.

В данной работе предлагается оставить возможность указывать количество потоков при запуске сервера. Это позволит использовать разработанную программу наиболее гибко.

Последние две проблемы предлагается решить при помощи средств взаимоисключения (например, mutex-ов), а также использования каждым потом своего сокета, обеспечив неблокирующий ввод-вывод посредством сокетов. На рисунке 1 представлена концептуальная модель пула потоков: очередь запросов, ожидающих обработку, пул потоков и обработанные запросы.

1.4 Асинхронный блокирующий ввод-вывод

Прежде, чем перейти к описанию системного вызова *pselect*, рассмотрим модель асинхронного блокирующего ввоода-вывода, в котой этот вызов используется.

Модель асинхронного блокирующего ввода-вывода основана на опросе набора источников ввода-вывода на предмет готовности. Блокировка происходит на моменте опроса: главный процесс блокируется в ожидании готовности

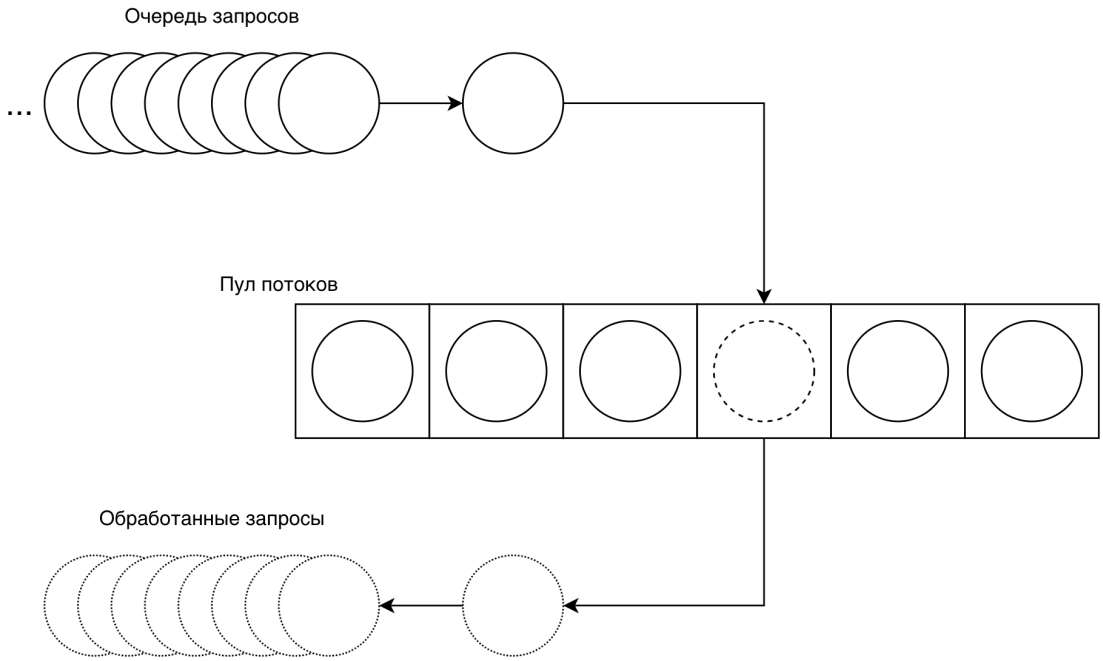

Рисунок 1 – Концептуальная модель пула потоков

соединения. Как только соединение установлено, можно осуществлять передачу и дальнейшую обработку данных. Асинхронным данный метод называется потому, что возможен одновременный опрос нескольких источников ввода-вывода и блокировка лишь до того момента, когда первый из источников сообщит о готовности.

На рисунке 2 представлена модель асинхронного блокирующего ввода-вывода с ипользованием select: запрос на чтение, блокировка в ожидании результата чтения, передача данных из пространства ядра в пространство пользователя.

1.5 Системный вызов pselect

Системный вызов *pselect* — функция, существующая в Unix-подобных и POSIX системах и предназначенная для опроса файловых дескрипторов открытых каналов ввода-вывода. Подключение данной функции происходит при помощи заголовочного файла sys/select.h на языке программирования Си.

Функция pselect принимает на вход 6 аргументов:

1) nfds, число типа int, значение котого вычисляется как $n+1$, где n — макси-

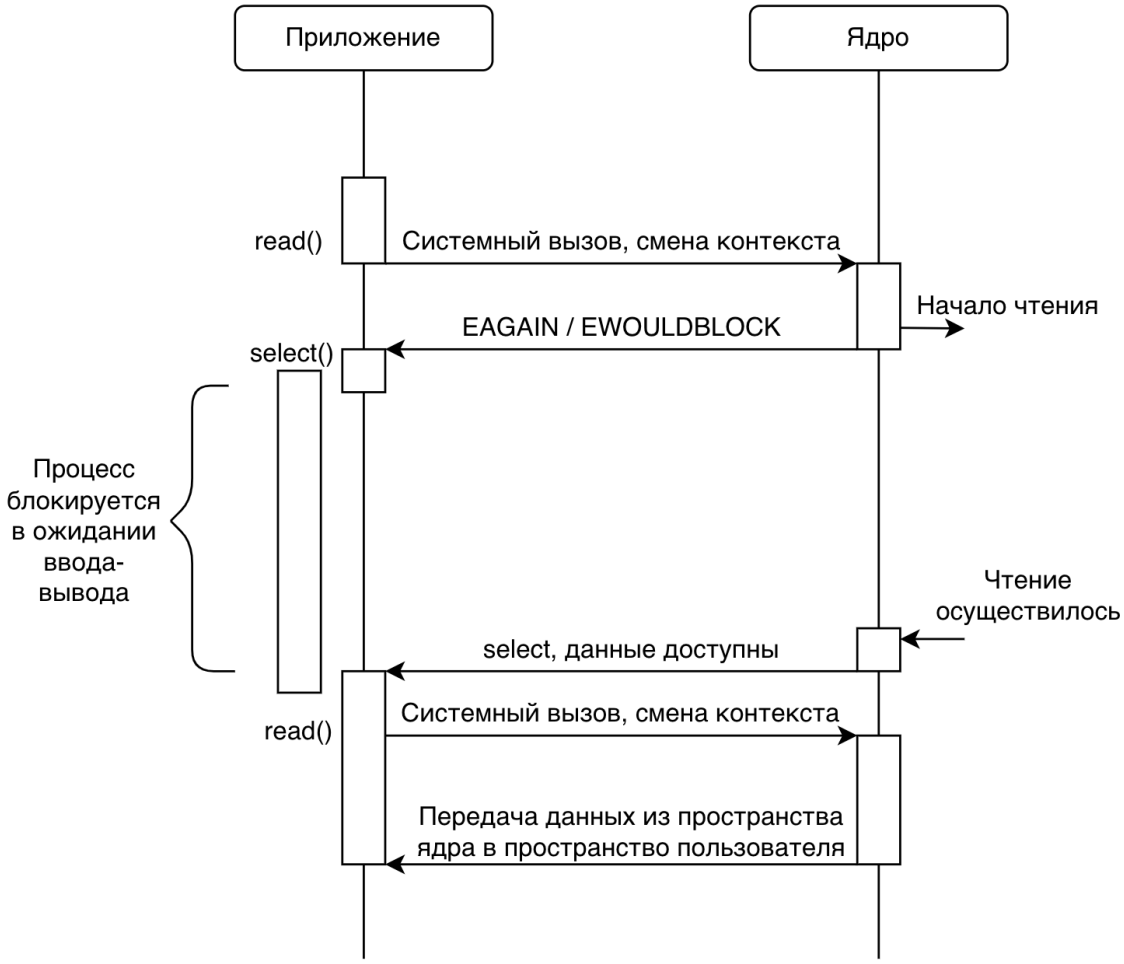

Рисунок 2 – Модель асинхронного блокирующего ввода-вывода с ипользованием select

мальное значение файлового дескриптора из наборов файловых дескрипторов, опрос которых осуществляется;

2) readfds, набор файловых дескрипторов типа fd_set, предназначенных для опроса на предмет чтения из них;
3) writefds, набор файловых дескрипторов типа fd_set, предназначенных для опроса на предмет записи в них;
4) exceptfds, набор файловых дескрипторов типа fd_set, предназначенных для опроса на предмет появления исключительных ситуаций;
5) timeout, структура типа struct timespec (содержит миллисекунды и наносекунды), определяет максимальный интервал времени, в течение которого будет осуществлено ожидание;

6) sigmask, маска типа sigset_t сигналов, позволяющая изменить маску сигналов на время работы функции select.

Возвращает функция количество файловых дескрипторов, готовых к вводу-выводу, 0 в случае, если за интервал времени timeout таких дескрипторов нет и -1 в случае ошибки.

При использовании в разрабатываемой программе необходимыми будут лишь параметры nfds, readfds и timeout, поскольку при раздаче статической информации требуется чтение с диска в определённый промежуток времени, а не запись на диск или мониторинг исключительных ситуаций на сокетах.

1.6 HTTP

HTTP (англ. *Hypertext Transfer Protocol*) — протокол прикладного уровня OSI ISO, изначально предназначенный для передачи гипертекстовых документов и сейчас являющийся универсальным средством взаимодействия между узлами в интернете.

HTTP сообщение состоит из трёх частей: стартовой строки, определяющей тип соообщения, заголовков, характеризующих тело сообщения, параметры передачи и прочие сведения, а также тела сообщения.

Одним из обязательных заголовков в HTTP сообщении является метод запроса: OPTIONS, GET, HEAD, PUT, POST, PATCH, DELETE, TRACE или CONNECT. В данной работе будут рассмотрены лишь два метода: GET и HEAD, поскольку именно они имеют смысл при реализации сервера раздачи статической инфрмации.

Метод GET используется для запроса содержимого указанного ресурса. Запросы GET считаются идемпотентными, т. е. результат запроса всегда будет одним и тем же в случае статической информации. В тело запроса при этом отсутствует, а в теле ответа будет находиться содержимое запрошенного ресурса, в случае разрабатываемого сервера — содержимое запрошенного файла.

Метод HEAD аналогичен методу GET за исключением того, что в ответе HEAD отсутствует тело сообщения. Обычно HEAD-запросы применяются для

извлечения метаданных или проверки существования ресурса.

Ещё один заголовок, который необходимо рассмотреть для правильной передачи данных, это Content-Type. Он предназначен для описания типа передаваемых данных для правильной интерпретации на стороне получателя. Для каждого из рассмотренных выше типов значение заколовка будет своим:

1) файлы HTML — text/html
2) скрипты JS — text/javascript;
3) файлы стилей CSS — text/css;
4) файлы изображений форматов PNG и JPEG — image/png и image/jpeg;
5) файлы SWF для векторной анимации — application/x-shockwave-flash;
6) файлы GIF — image/gif.

1.7 Формализация требований к разрабатываемой программе

На основе рассмотренной выше информации и задания к работе разрабатываемая программа должна соответстовать следующим требваниям:

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 без сторонних библиотек;
9) реализовать архитектуру при помощи пула потоков с использованием системного вызова pselect;
10) обеспечить стабильную работу сервера.

Вывод

В данном разделе проводится анализ предметной области, анализ паттерна *thread pool*, асинхронной блокирующей модели ввода-вывда и системного вызова *pselect*, а также протокол HTTP и формализация требований к разрабатываемой программе. Необходимо будет провести нагрузочное тестирование при помощи Apache Benchmarks и сравнить результаты тестирования разработанной программы с результатами тестирования NGINX-сервера. Как архитектурный паттерн будет использоваться пул потоков, как модель ввода-вывода асинхронный блокирующий ввод-вывод, осуществляемый при помощи системного вызова *pselect*.

2 Конструкторская часть

В данном разделе рассматриваются схемы алгоритмов: схема алгоритма работы сервера раздачи статической инфорации и схема алгоритма потока пула, обрабатывающего соединение.

2.1 Схемы алгоритмов

На рисунке 3 приведена схема алгоритма рабты сервера, а на рисунке 4 — схема алгоритма потока пула, обрабатвающего соединение.

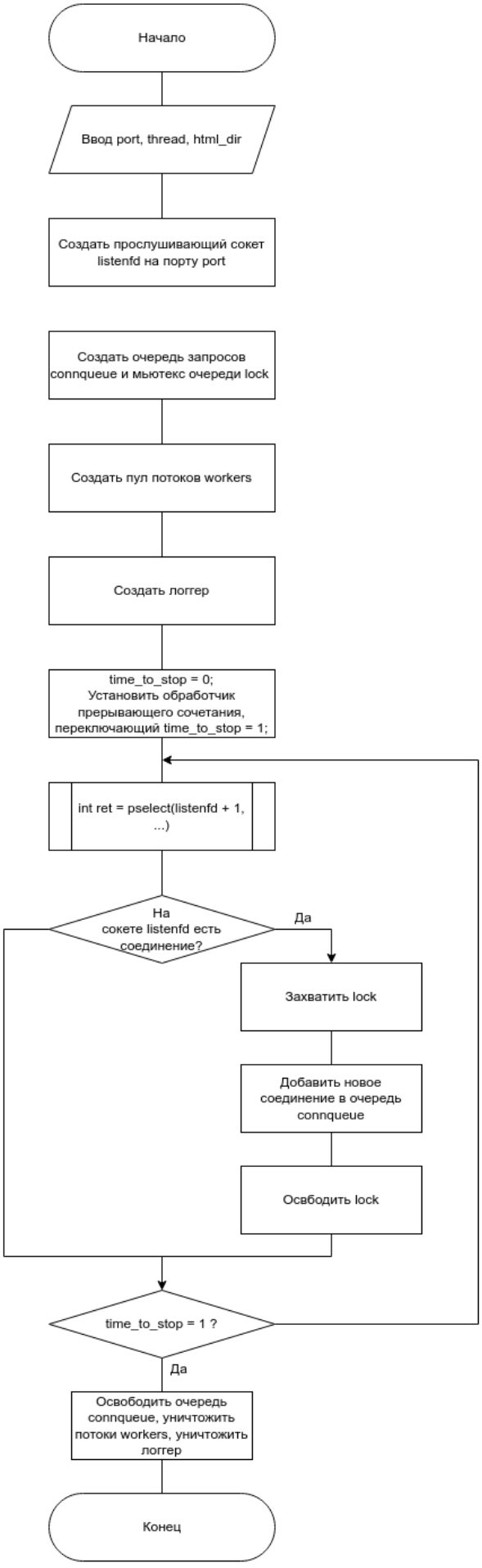

Рисунок 3 – Схема алгоритма работы сервера

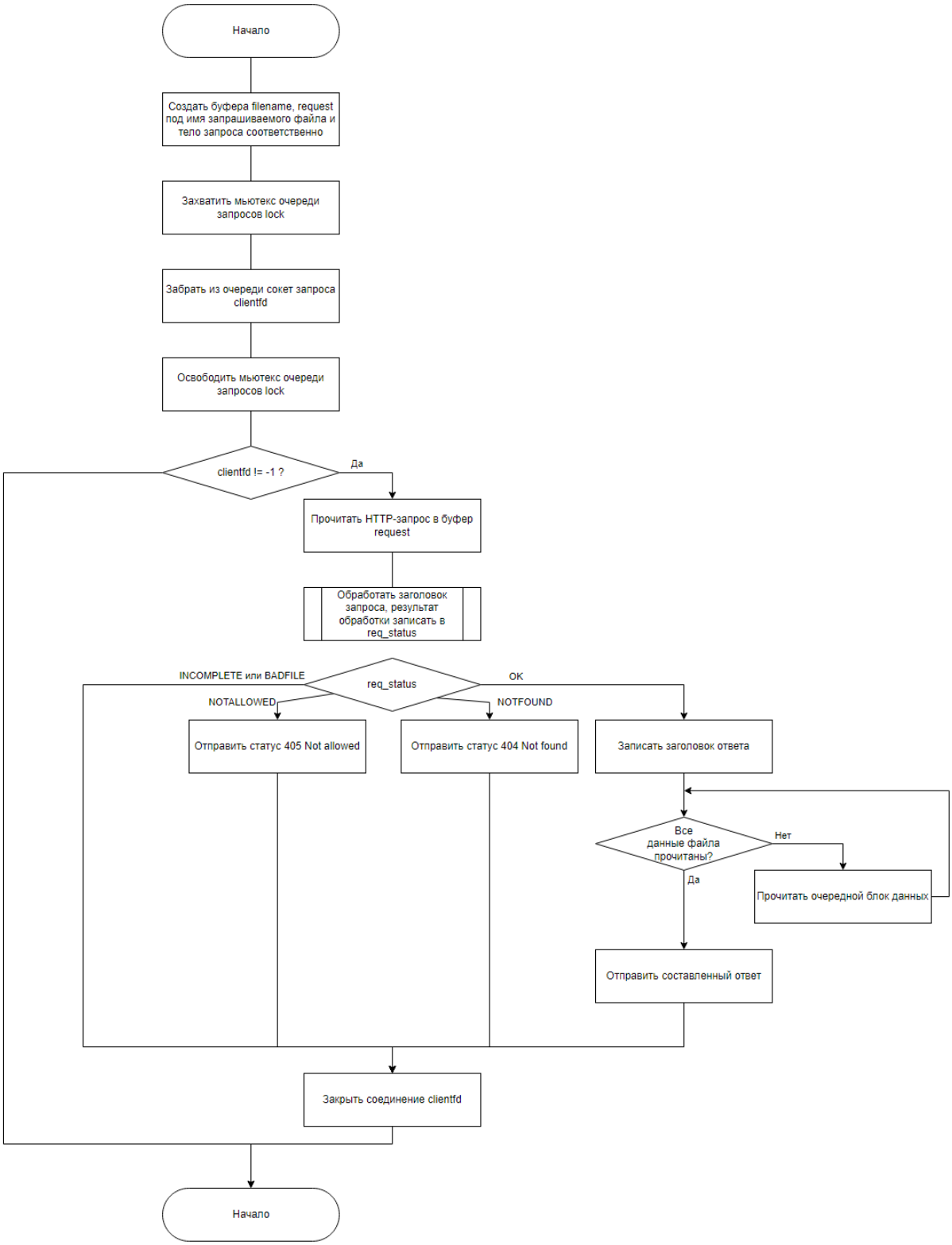

Рисунок 4 – Схема алгоритма потока пула, обрабатвающего соединение

Вывод

В данном разделе были рассмотрены схемы алгоритмов: схема алгоритма работы сервера раздачи статической инфорации и схема алгоритма потока пула, обрабатывающего соединение.

3 Технологическая часть

В данном разделе рассмотрены средства разработки программы, модули разработанного сервера и листинги исходных кодов.

Приведённые листинги исходных кодов включают в себя функции

3.1 Средства реализации

Как основное средство реализации и разработки ПО был выбран язык программирования Си. Причиной выбора данного языка является следующие причины: во-первых, согласно требованиям к работе должен быть использован этот язык, во-вторых, язык Си компилируется напрямую в ассемблер, команды которого напрямую соответствуют машинным командам, что делает язык Си предпочтительным для написания сервера, поскольку это обеспечивает наименьшее время обработки запроса.

3.2 Модули программы

Реализованная программа состоит из 5 следующих модулей:

1) main, содержащий точку входа в программу, а также основной цикл обработки сервера;
2) server, содержащий набор функций для обработки запросов;
3) thread_pool, содержащий функцию создания пула потоков ;
4) logger, содержащий функции для логирования запросов;
5) http, содержащий необходимые для обработки HTTP запросов константы.

3.3 Реализация сервера

В расположенных ниже листингах 1 – 4 приведены реализации основного цикла сервера, а в листинге 5 —

Листинг 1 – Реализация основного цикла сервера (часть 1)

```
int main(int argc, char *argv[])
{
    int port = DEFAULT_PORT;
    html_dir = DEFAULT_STATIC;

    if(parse_args(argc, argv, &port, \
        &threads, &html_dir) == -1)
            return EXIT_SUCCESS;
    if(!check_html_dir())
        exit(EXIT_FAILURE);
    html_dir_len = strlen(html_dir);
    client_sockfd = calloc(threads, sizeof(int));
    int listenfd;
    if((listenfd = create_listen_socket(port)) < 0)
        exit(EXIT_FAILURE);
    if((connqueue = create_queue(MAX_CONNECTIONS)) == NULL)
    {
        close(listenfd);
        exit(EXIT_FAILURE);
    }
    pthread_t *workers;
    if((workers = create_threadpool(threads, \
        (void *)handle_connection)) == NULL)
    {
        close(listenfd);
        pthread_mutex_destroy(&(connqueue->lock));
        pthread_cond_destroy(&(connqueue->cond_var));
        exit(EXIT_FAILURE);
    }
```

Листинг 2 – Реализация основного цикла сервера (часть 2)

```
    create_logger();
    bytes_read = calloc(threads, sizeof(unsigned int));
    bytes_wrote = calloc(threads, sizeof(unsigned int));

    printf("Server running on port: %d\nthreads: \
        %d\nstatic directory: %s\nType Ctrl + C to exit\n",\
        port, threads, html_dir);

    signal(SIGPIPE, SIG_IGN);
    signal(SIGINT, stop_server);

    int clientfd = -1;
    struct sockaddr_in client_addr;
    socklen_t len;
    int i;
    struct timespec tv;
    bzero((void*)client_sockfd, sizeof(client_sockfd));
    int ret = 0;
    while (1)
    {
        if (time_to_stop)
            break;
        FD_ZERO(&client_fdset);
        FD_SET(listenfd, &client_fdset);
        tv.tv_sec = 30;
        tv.tv_nsec = 0;
        ret = pselect(listenfd+1, &client_fdset, \
            NULL, NULL, &tv, NULL);
        if(ret < 0 && time_to_stop) break;
```

Листинг 3 – Реализация основного цикла сервера (часть 3)

```
        if(FD_ISSET(listenfd, &client_fdset))
        {
            memset(&client_addr, 0, sizeof(client_addr));
            len = sizeof(client_addr);
            clientfd = accept(listenfd, \
                (struct sockaddr*)&client_addr, &len);
            if(clientfd < 0)
            {
                perror("accept error!\n");
                continue;
            }
            pthread_mutex_lock(&(connqueue->lock));
            if(enqueue(connqueue, clientfd) < 0)
            {
                printf("Connection capacity \
                    reached. Dropped new connection!\n");
                close(clientfd);
            } else
                pthread_cond_signal(&(connqueue->cond_var));
            pthread_mutex_unlock(&(connqueue->lock));
        }
    }
    printf("Closing socket...\n");
    close(listenfd);
    printf("Stopping server\n");
    for(i = 0; i < threads; i++)
        pthread_cancel(workers[i]);
    freequeue(connqueue);
    delete_logger();
    unsigned int total_wrote = 0;
```

Листинг 4 – Реализация основного цикла сервера (часть 4)

```
    unsigned int total_read = 0;
    for(i = 0; i < threads; i++)
    {
        printf("thread %u\twrote: %u\tread: %u\n"\,\
            i, bytes_wrote[i], bytes_read[i]);
        total_wrote += bytes_wrote[i];
        total_read += bytes_read[i];
    }
    printf("Total bytes received: %u Bytes\n Total bytes \
    sent: %u Bytes\n", total_read, total_wrote);

    return EXIT_SUCCESS;
}
```

Листинг 5 – Реализация функции обработки запроса в пуле потоков (часть 1)

```
void* handle_connection(void *args)
{
    int tindex = *((int*)args);
    while(1)
    {
        int clientfd = -1;
        int htmlfd = -1;
        enum http_status req_status;
        int br,bw;
```

Листинг 6 – Реализация функции обработки запроса в пуле потоков (часть 2)

```
            int total_wrote = 0;
            char *response_header;
            char filebuffer[FILE_BUFFER_SIZE] = {'\0'};
            char req_buffer[REQUEST_BUFFER_SIZE] = {'\0'};

            pthread_mutex_lock(&(connqueue->lock));
            clientfd = dequeue(connqueue);
            pthread_mutex_unlock(&(connqueue->lock));

            if(clientfd == -1)
                continue;

            br = read(clientfd, req_buffer, REQUEST_BUFFER_SIZE-1);
            if(br <= 0)
            {
                close_clientfd(clientfd, tindex, total_wrote);
                continue;
            }
            req_buffer[br] = '\0';
            bytes_read[tindex] += br;

            req_status = handle_http_request(req_buffer, \
                &htmlfd, &response_header);
            switch(req_status)
            {
                case INCOMPLETE:
                case BADFILE:
                {
                    close_clientfd(clientfd, tindex, total_wrote);
                    continue;
                }
```

Листинг 7 – Реализация функции обработки запроса в пуле потоков (часть 3)

```
                case NOTALLOWED:
                {
                    bw = write(clientfd, HTTP_405, HTTP_405_len);
                    if(bw > 0)
                        total_wrote += bw;
                    close_clientfd(clientfd, tindex, total_wrote);
                    continue;
                }
                case NOTFOUND:
                {
                    bw = write(clientfd, HTTP_404, HTTP_404_len);
                    if(bw > 0)
                        total_wrote += bw;
                    close_clientfd(clientfd, tindex, total_wrote);
                    continue;
                }
                case OK:
                {
                    bw = write(clientfd, response_header, \
                        strlen(response_header));
                    char filename[FILE_NAME_SIZE] = {'\0'};
                    int method = get_file_name(req_buffer, \
                        filename);
                    if(bw <= 0)
                    {
                        close_clientfd(clientfd, tindex, \
                            total_wrote);
                        continue;
                    }
                    total_wrote += bw;
```

Листинг 8 – Реализация функции обработки запроса в пуле потоков (часть 4)

```
                int flag = 0;
                while(method != 2 && (br = \
                read(htmlfd, filebuffer, FILE_BUFFER_SIZE-1)))
                {
                    filebuffer[br] = '\0';
                    bw = write(clientfd, filebuffer, br);
                    if(bw <= 0)
                    {
                        close_clientfd(clientfd, tindex, \
                            total_wrote);
                        flag = 1;
                        break;
                    }
                    total_wrote += bw;
                    memset(filebuffer, 0, bw);
                }
                close(htmlfd);
                if (flag)
                    break;
            }
        }
        close_clientfd(clientfd, tindex, total_wrote);
    }
    return NULL;
}
```

Вывод

В данном разделе были рассмотрены средства разработки программы, модули разработанного сервера и листинги исходных кодов. Разработанную про-

грамму следует протестировать при помощи нагрузочного тестирования с использованием Apache Benchmarks, а также сравнить результаты тестирования с тестированием сервера раздачи статической информации, реализованного при помощи NGINX.

4 Исследовательская часть

В данном разделе будут приведены примеры работы разработанной программы и проведено сравнение результатв нагрузочного тестирования, проведённого при помощи Apache Benchmarks с NGINX.

4.1 Пример работы программы

На рисунке 5 приведён пример работы программы: базовая страница, отображаемая при обращении к серверу.

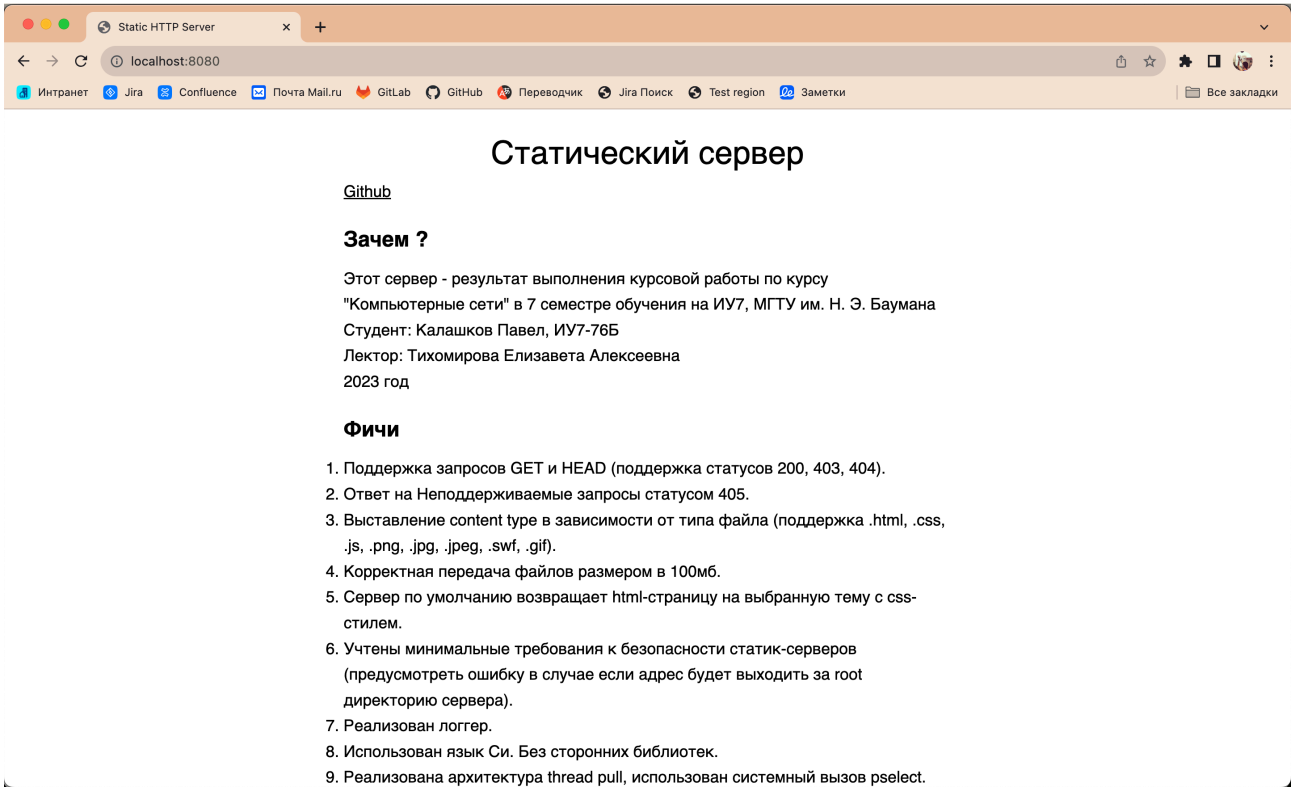

Рисунок 5 – Базовая страница

4.2 Нагрузочное тестирование

Нагрузочное тестирование было осуществлено при помощи Apache Benchmarks следующим образом: замерялось время обработки от 100 до 1000 (с шагом 100) запросов при подключении 5, 50 и 100 клиентами для разработанного сервера и для NGINX.

Технические характеристики устройства, на котором выполнялось тестирование:

— Операционная система: macOS Ventura 13.2.1 (22D68) [**?**];

— Память: 16 Гб с тактовой частотой 2133 МГц LPDDR3 [**?**];

— Процессор: Intel Core™ i7-8559U [**?**] с тактовой частотой 2.70 ГГц;

— Видеокарта: Intel Iris Plus Graphics 655 [**?**] с объёмом памяти 1536 Мб.

Тестирование проводилось на ноутбуке, включенном в сеть электропитания. Во время тестирования ноутбук был нагружен только системой тестирования (работающим приложением) и системным окружением операционной системы

Результаты тестирования для 5, 50 и 100 клиентов приведены на рисунках 6, 7 и 8 соответственно.

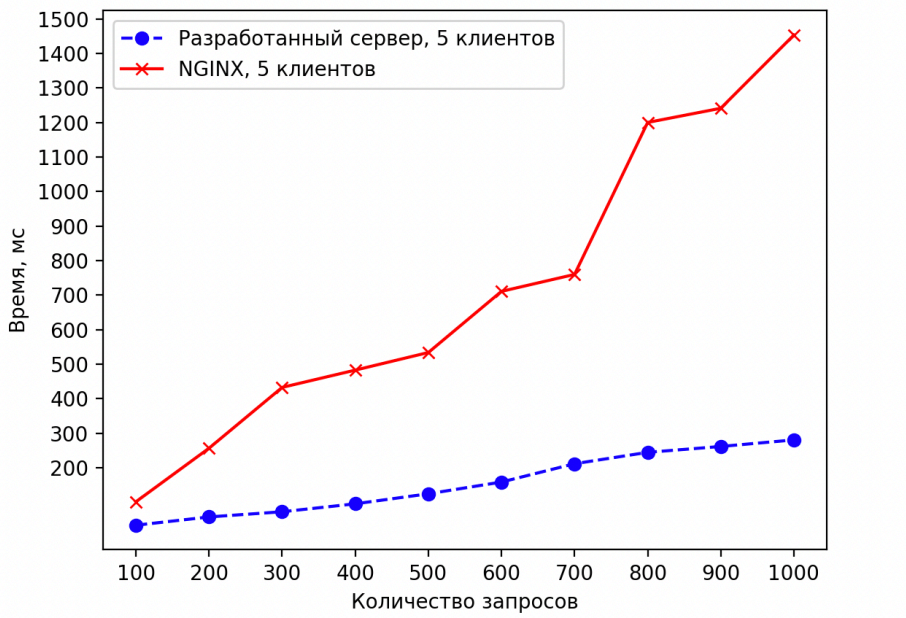

Рисунок 6 – Результаты нагрузочного тестирования в 5 клиентов

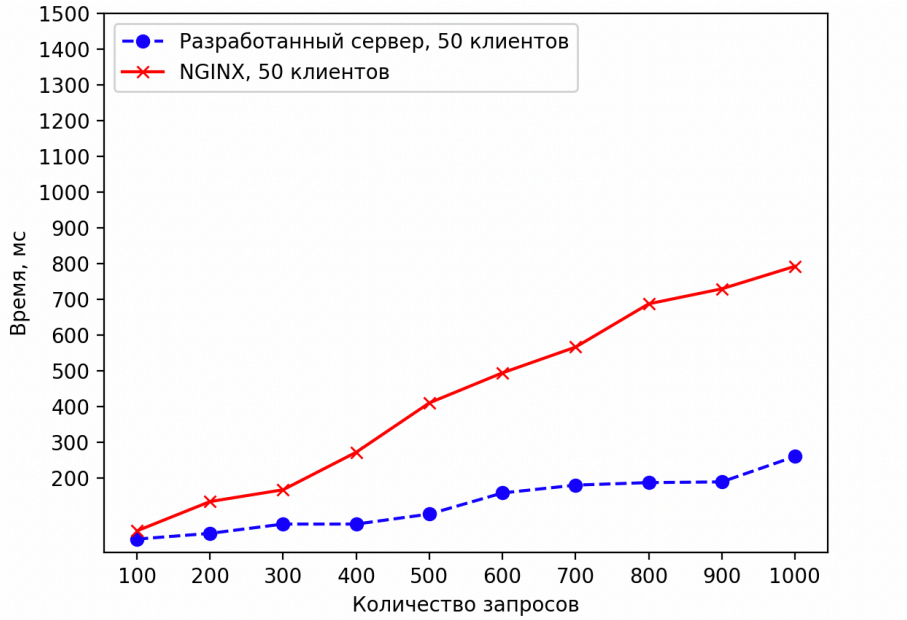

Рисунок 7 – Результаты нагрузочного тестирования в 50 клиентов

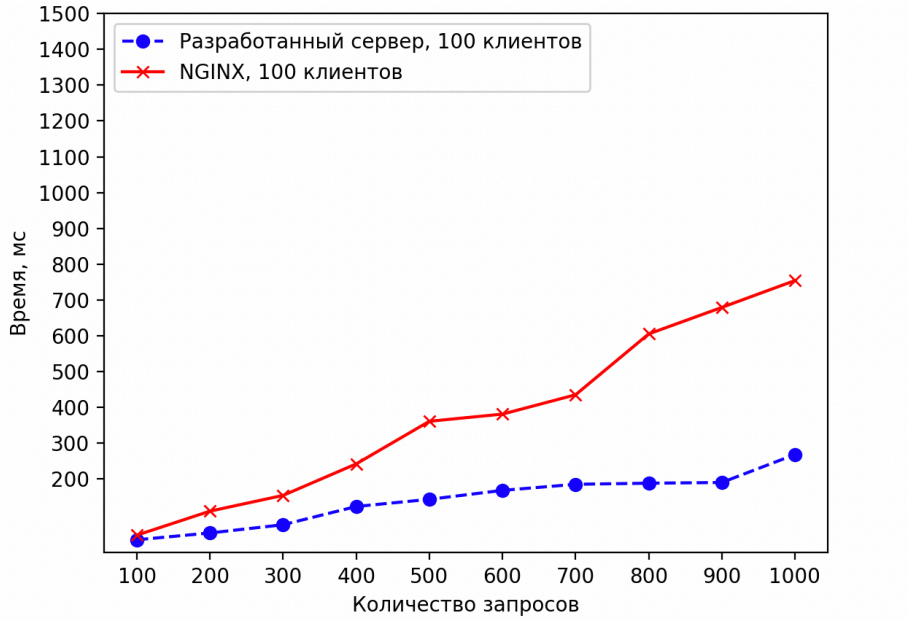

Рисунок 8 – Результаты нагрузочного тестирования в 100 клиентов

.

Результаты тестирования приводят к выводу, что время обработки запросов при раздаче статической информации разработанной программой меньше, чем при использовании NGINX. Чем меньше количество подсоединённых клиентов, тем больше разница во времени: так, при 1000 запросах и 5 клиентах разработанный сервер даёт разницу в 7 раз, в то время как при 50 клиентах этоо соотноошение сокращается до 3 раз. Это объясняется тем, что при замере времени обработки количество запросов оставалось одним и тем же (от 100 до 1000 с шагом 100), то есть суммарное число запросов, приходящееся на одного клиента, уменьшалось с увеличением числа клиентов.

Вывод

В данном разделе были приведены примеры работы разработанной программы и проведено сравнение результатв нагрузочного тестирования, проведённого при помощи Apache Benchmarks с NGINX.

Исходя из полученных данных, при небольшом (до 100) количестве клиентов разработанная программа показывает меньшее среднее время обработки запросов, чем NGINX, при этом чем меньше клиентов подсоединено к серверу, тем больше эта разница (до 7 раз при 5 клиентах).

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы был разработан сервер раздачи статической информации на языке Си (без использования сторонних библиотек), построенный при помощи паттерна *thread pool* и использующий системный вызов *pselect*. Для достижения поставленной цели были выполнены следующие задачи:

1) проведён анализ предметной области;
2) определён функционал, реализуемый сервером раздачи статической инфоормации;
3) проведён анализ паттерна *thread pool* и системного вызова *pselect* ;
4) спроектирован и разработан сервер раздачи статическй информации;
5) проведено сравнение результатов нагрузочного тестирования при помощи Apache Benchmarks с NGINX.

Из результатов следует, что при небольшом (до 100) количестве клиентов разработанная программа показывает меньшее среднее время обработки запросов, чем NGINX, при этом чем меньше клиентов подсоединено к серверу, тем больше эта разница (до 7 раз при 5 клиентах).

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. DIGITAL 2022: THE RUSSIAN FEDERATION [Электронный ресурс]. — Режим доступа: https://datareportal.com/reports/digital-2022-russian-federation (дата обращения: 07.12.2023).

2. Forbes. Число доменов зоны .ru [Электронный ресурс]. — Режим доступа: https://www.forbes.ru/tekhnologii/483876-cislo-domenov-zony-ru-po-itogam-2022-goda-opustilos (дата обращения: 07.12.2023).

3. Telecom. В 2023 году проводной трафик превысит 100 эксабайт [Электронный ресурс]. — Режим доступа: https://telecomdaily.ru/news/2023/06/15/v-2023-godu-provodnoy-trafik-prevysit-100-eksabayt (дата обращения: 07.12.2023).

4. NGINX, официальный сайт [Электронный ресурс]. — Режим доступа: https://www.nginx.com/ (дата обращения: 08.12.2023).

5. November 2023 Web Server Survey [Электронный ресурс]. — Режим доступа: https://www.netcraft.com/blog/november-2023-web-server-survey/ (дата обращения: 08.12.2023).

6. Ling Y., Mullen T,, Lin X., Analysis of optimal thread pool size — ACM SIGOPS Operating Systems Review, 200. – 55 с..

7. select(2) — Linux manual page [Электронный ресурс]. — Режим доступа: https://man7.org/linux/man-pages/man2/select.2.html (дата обращения: 08.12.2023).