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 ИУ7-72Б _____ Карпова Е.О
(Группа) (Подпись, дата) (И. О. Фамилия)

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

2024 г.

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)

УТВЕРЖДАЮ
Заведующий кафедрой ИУ-7
_____ И. В. Рудаков
«16» сентября 2023 г.

З А Д А Н И Е
на выполнение курсовой работы

по дисциплине

Компьютерные сети

по теме

«Разработка статического сервера»

Студент группы **ИУ7-72Б**

Карпова Екатерина Олеговна

Направленность КР

учебная

Источник тематики

кафедра

График выполнения: 25% к 6 нед., 50% к 9 нед., 75% к 12 нед., 100% к 15 нед.

Техническое задание

Провести обзор функциональности статического сервера. Описать алгоритм работы статического сервера. Реализовать статический сервер. Провести нагрузочное тестирование разработанного сервера и сервера Nginx. Сравнить результаты.

Оформление курсовой работы:

Расчетно-пояснительная записка на **12-20** листах формата А4.

Дата выдачи задания «16» сентября 2023 г.

Руководитель курсовой работы	_____	**Исполатов Ф. О.**
	(Подпись, дата)	(Фамилия И. О.)
Студент	_____	**Карпова Е. О.**
	(Подпись, дата)	(Фамилия И. О.)

СОДЕРЖАНИЕ

РЕФЕРАТ

Расчетно-пояснительная записка 20 с., 5 рис., 0 табл., 0 источн., 0 прил.

Ключевые слова: TCP, HTTP, статический сервер, Nginx, нагрузочное тестирование.

В данной работе был проведен обзор функциональности и составляющих статического сервера, был описан алгоритм его работы и реализован непосредственно статический сервер. Было проведено нагрузочное тестирование полученного продукта, результаты которого сравнивались с аналогичными результатами для Nginx [**dejonghe2020nginx**].

ОПРЕДЕЛЕНИЯ

В настоящей расчетно-пояснительной записке применяют следующие термины с соответствующими определениями.

Статический сервер — сервер, возвращающий клиенту файлы, хранящиеся на диске, не изменяя содержимого этих файлов [**РҳРњРчРчРѳСЃР«Рў2021РѳРўСЂР«РёРѳСЂРчРчРчСЃР«РўРѳР„Р„**

Сокет — абстакция конечной точки соединения [**ryaznu**].

Протокол — набор соглашений логического уровня [**tikh**].

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

В настоящей расчетно-пояснительной записке применяют следующие сокращения и обозначения.

HTTP — HyperText Transfer Protocol [**fielding2022rfc**].

TCP — Transmission Control Protocol [**eddy2022rfc**].

UDP — User Datagram Protocol [**tcp**].

ВВЕДЕНИЕ

Статический сервер — программное обеспечение, позволяющее клиентам получать доступ к файлам на сервере с учетом контроля доступа [**intro**].

Целью работы является разработка статического сервера.

Для достижения поставленной цели необходимо решить следующие задачи:

1. провести обзор функциональности статического сервера;
2. провести обзор компонентов, из которых состоит статический сервер;
3. описать алгоритм работы статического сервера.
4. реализовать статический сервер;
5. провести нагрузочное тестирования разработанного программного обеспечения, сравнить результаты с нагрузочным тестированием Nginx.

1 Аналитический раздел

В данном разделе приведен обзор функциональности и составляющих статического сервера.

1.1 Протоколы TCP и UDP

Протокол дейтаграмм пользователя UDP не ориентирован на создание соединения, его главное отличие — отсутствие гарантии доставки и поддержки упорядоченности передаваемых сообщений до места назначения [**tcp**]. Поэтому в приложениях, использующих UDP, разработчики должны реализовывать функции, компенсирующие ненадежность этого протокола: таймауты, повторную передачу, обработку потерянных дейтаграмм и порядковые номера для сопоставления ответов запросам.

Протокол TCP — протокол транспортного уровня модели OSI/ISO. Перед началом передачи данных в обязательном порядке устанавливается соединение и использующим его приложениям обеспечивается надежный упорядоченный двухсторонний байтовый поток. Протокол поддерживает отправку и прием подтверждений, обработку таймаутов, повторную передачу, управление потоком и прочие возможности [**tcp**].

Все отправленные данные подлежат обязательному подтверждению встречной стороной, причем формируются подтверждения не для каждого конкретного успешно полученного пакета, а для всех данных от начала посылки до некоторого порядкового номера. Если подтверждение не приходит в течение времени RTO (Retransmission Time Out), то протокол TCP автоматически передает данные повторно и перезапускает таймер вновь. Величина таймера RTO динамически меняется и зависит от времени двухсторонней задержки, определяемой с помощью специальных алгоритмов, типа сети и конкретной реализации протокола.

1.2 Протокол HTTP

HTTP — широко распространённый протокол передачи данных, изначально предназначенный для передачи гипертекстовых документов. Клиенты и серверы взаимодействуют, обмениваясь одиночными сообщениями, а не потоком дан-

ных [**http**].

HTTP-сообщения — это формат обмена данными между сервером и клиентом. Есть два типа сообщений:

1. запросы, отправляемые клиентом, чтобы инициировать реакцию со стороны сервера;

2. ответы от сервера.

Сообщения HTTP состоят из текстовой информации в кодировке ASCII, записанной в несколько строк. В HTTP/1.1 и более ранних версиях они пересылались в качестве обычного текста. В HTTP/2 текстовое сообщение разделяется на фреймы, что позволяет выполнить оптимизацию и повысить производительность.

Веб разработчики не создают текстовые сообщения HTTP самостоятельно — это делает программа, браузер, прокси или веб-сервер. Они обеспечивают создание HTTP-сообщений через конфигурационные файлы (для прокси и серверов), APIs (для браузеров) или другие интерфейсы.

Так как HTTP это клиент-серверный протокол, соединение всегда устанавливается клиентом. Открыть соединение в HTTP — значит установить соединение через соответствующий транспортный протокол, обычно TCP.

В случае с TCP, в качестве порта HTTP-сервера по умолчанию на компьютере используется порт 80, хотя другие также часто используются, например 8000 или 8080. URL загружаемой страницы содержит доменное имя и порт, который можно и не указывать, если он соответствует порту по умолчанию.

1.3 Сокеты

Под сокетом понимают абстракцию конечной точки соединения. Сокеты — универсальное средство взаимодействия параллельных процессов, применяемое как на локальной машине, так и в распределенной системе [**ryaznu**].

Сокеты создаются системным вызовом `socket` со следующими параметрами.

1) Family/Domain — домен соединения: Некоторые значения для домена:

- AF_UNIX, AF_LOCAL — локальное соединение;
- AF_INET — протокол IPv4;
- AF_INET6 — протокол IPv6;
- AF_UNSPEC — неопределенный;
- AF_NETLINK — используется для взаимодействия с ядром.

2) Type — задает семантику коммуникации. Некоторые типы:

   - SOCK_STREAM — определяет ориентированное на потоки, надежное, упорядоченное, полудуплексное соединение между двумя сокетами.
   - SOCK_DGRAM — определяет ненадежную службу datagram без установления логического соединения, где пакеты могут передаваться без сохранения порядка.
   - SOCK_RAW — обеспечивает доступ к низкоуровневому сетевому протоколу.

3) Protocol — задает конкретный протокол, который работает с сокетом.

   - IPPPROTO_TCP (по умолчанию для SOCK_STREAM).
   - IPPPROTO_UDP (по умолчанию для SOCK_DGRAM).

Взаимодействие на сокетах осуществляется по модели «клиент-сервер»: сервер предоставляет ресурсы и службы одному или нескольким клиентам, которые обращаются к серверу за обслуживанием. Схема взаимодействия представлена на рисунке 1.1.

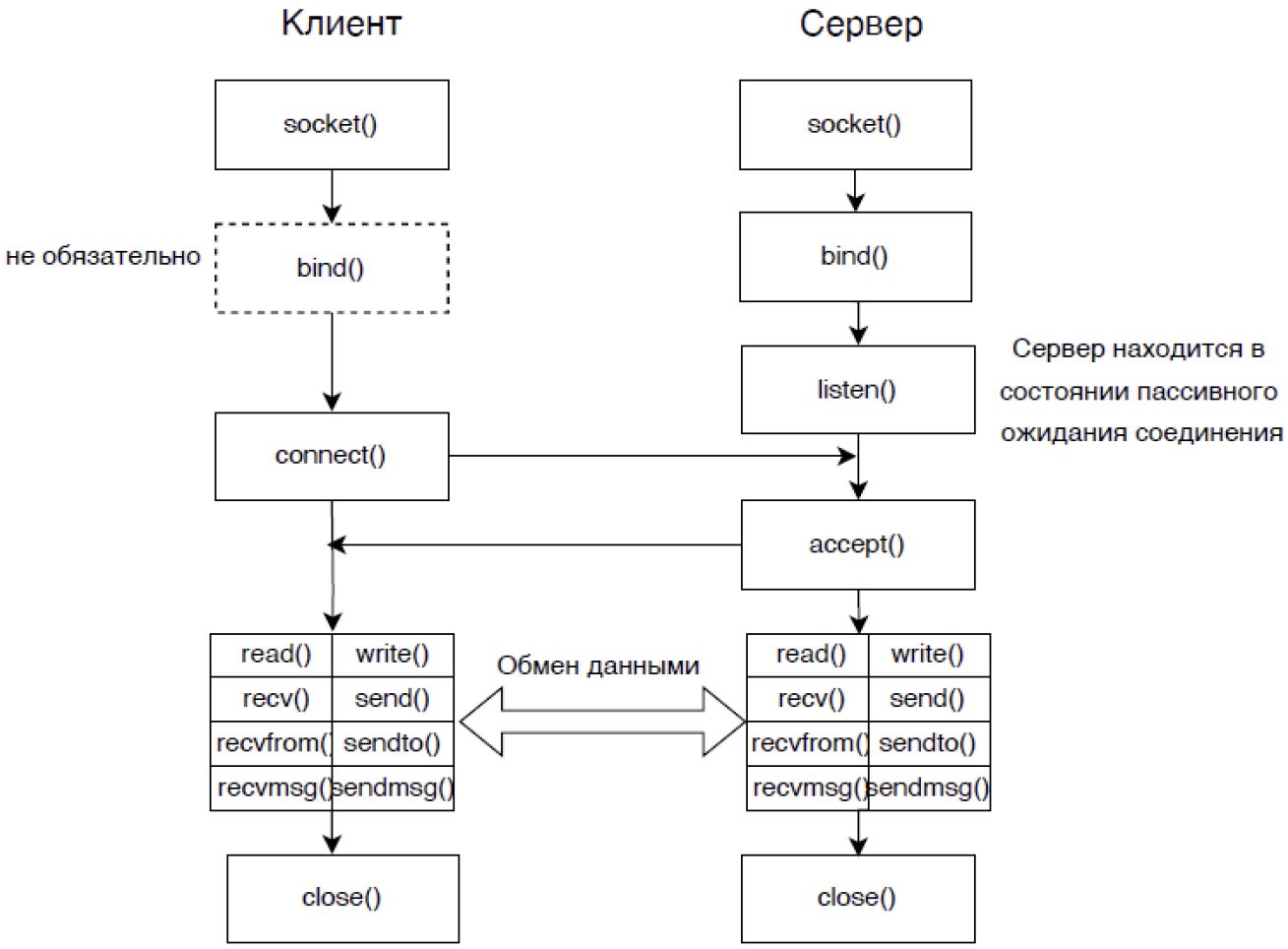

Рисунок 1.1 – Схема взаимодействия на сокетах по модели «клиент-сервер»

1.4 Мультиплексирование

Для уменьшения времени блокировки сервера на ожидании соединения используется мультиплексирование, т. к. время установления соединения с первым готовым к взаимодействию меньше, чем с каждым конкретным клиентом в определенной последовательности. Мультиплексор опрашивает соединения. Первое готовое соединение фиксируется ядром. Мультиплексирование является менее затратным вариантом многопоточности. Для мультиплексирования ОС предоставляет системные вызовы, с помощью которых можно обратиться к одному из доступных мультиплексоров. Существующие мультиплексоры: select, poll, pselect, dev/poll, epoll [**ryaznu**].

На рисунке 1.2 представлена схема мультиплексирования.

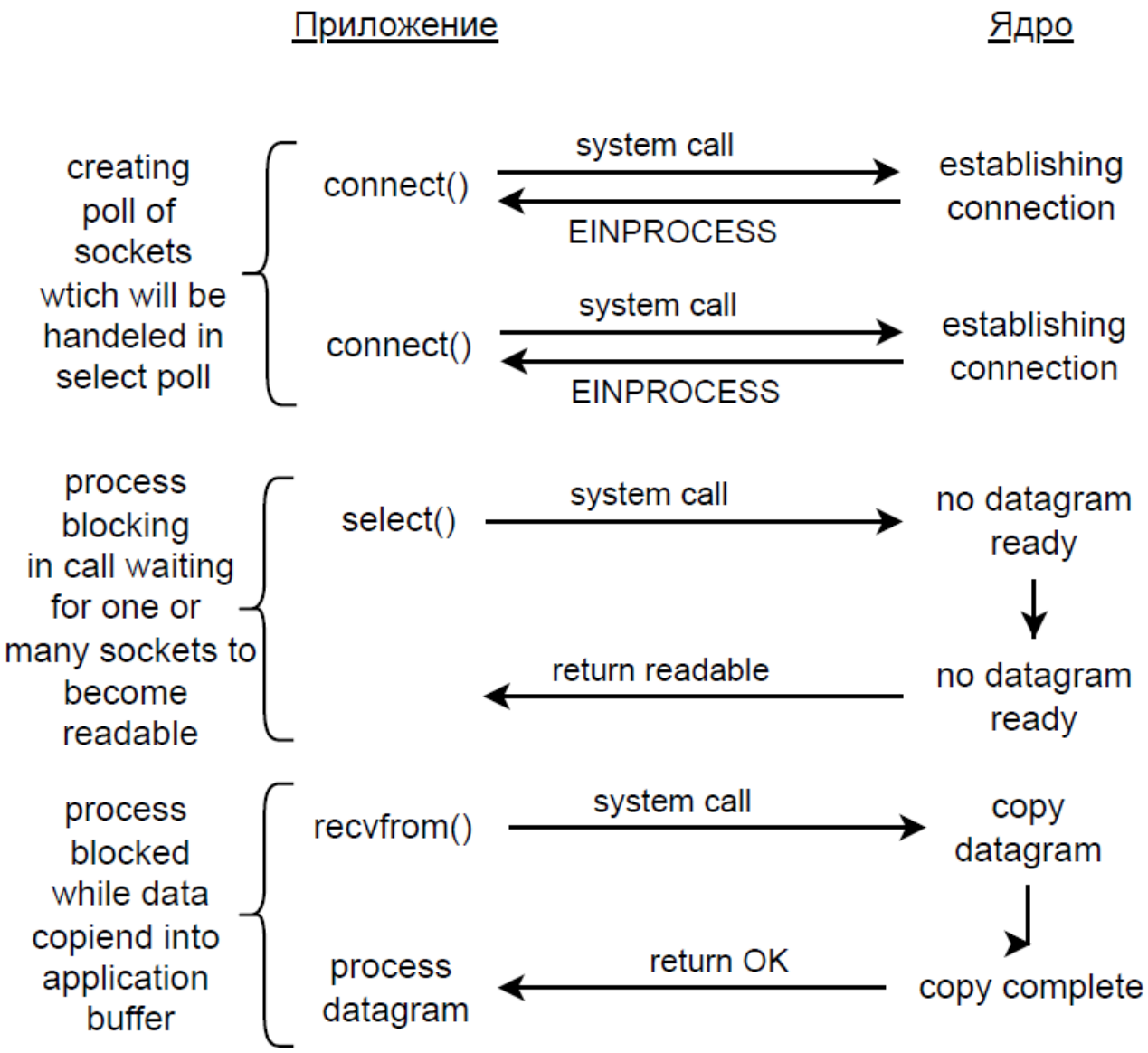

Рисунок 1.2 – Схема мультиплексирования

1.5 Пул потоков

Пул потоков — это фиксированный набор потоков, одновременно выполняющих независимые друг от друга задачи, помещенные в некоторый массив. Массив задач обычно представляется в виде очереди [**ryazanov**].

Перед поступлением заявок в очередь создаются все потоки, которые могут принимать участие в обработке, и переводятся в режим ожидания. Вновь прибывшую задачу достает любой свободный поток из пула и начинает ее

выполнять. Закончив обработку, поток возвращается в режим ожидания.

Данный подход позволяет повысить производительность программы, так как избавляет от необходимости в процессе обработки заявок из очереди создавать новые потоки, что было бы трудоемко для ОС.

2 Конструкторский раздел

В данном разделе описан алгоритм работы статического сервера.

2.1 Алгоритм работы статического сервера

На рисунке 2.1 показана схема алгоритма мультиплексирования статического сервера с пулом потоков.

Рисунок 2.1 – Схема алгоритма мультиплексирования статического сервера с пулом потоков

2.2 Алгоритм работы потока

На рисунке 2.2 показана схема алгоритма работы потока на статическом сервере.

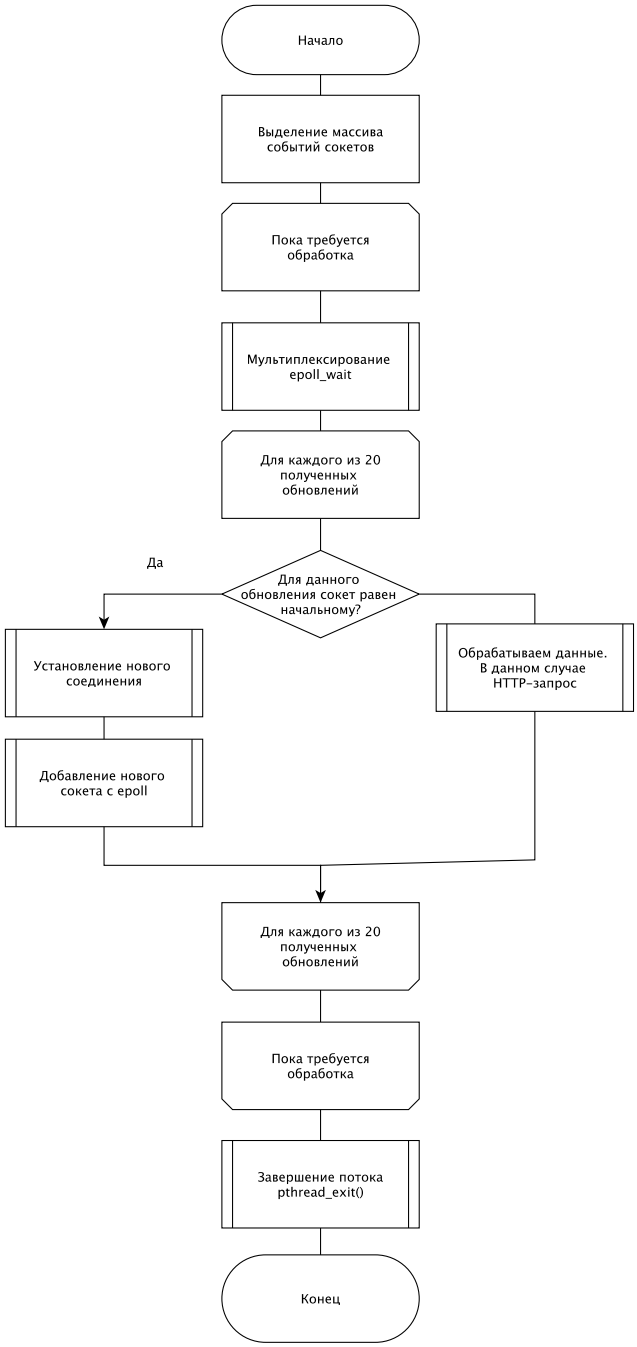

Рисунок 2.2 – Схема алгоритма работы потока на статическом сервере

3 Технологический раздел

В данном разделе приведена реализация статического сервера.

3.1 Листинг алгоритма работы статического сервера

В листинге 3.1 приведен программный код работы статического сервера.

Листинг 3.1 – Листинг алгоритма работы статического сервера

```
1  void accept_connection() {
2      int fd = accept(sockfd, NULL, NULL);
3      if (fd < 0) {
4          log_error("accept failed");
5          return;
6      }
7
8      struct epoll_event e;
9      e.events = EPOLLIN | EPOLLET;
10     e.data.fd = fd;
11     if (epoll_ctl(epollfd, EPOLL_CTL_ADD, fd, &e) < 0) {
12         log_fatal("epoll_ctl: failed to add socket");
13     }
14
15     log_info("got new connection");
16 }
17
18 void use_fd(int fd) {
19     char buf[BUF_SIZE] = {0};
20
21     ssize_t n = read(fd, buf, sizeof(buf));
22     if (n == 0) {
23         log_info("connection is closed");
24         return;
25     }
26
27     http_request_t req;
28     req.fd = fd;
29     int rc = http_parse_request(buf, &req);
```

```
30      if (rc != OK) {
31          log_error("cannot parse request: %s", error_messages[rc]);
32          if (rc == ERR_UNSUPPORTED_METHOD) {
33              http_write_error(&req,
                    RESPONSE_CODE_METHOD_NOT_ALLOWED);
34          }
35          return;
36      }
37
38      rc = http_handle(&req);
39      if (rc != OK) {
40          log_error("cannot handle request: %s", error_messages[rc]);
41          return;
42      }
43
44      log_debug("%s \"%s\"", http_methods_map[req.method], req.uri);
45  }
46
47  void *handle_connections(void *data) {
48      int rc;
49
50      log_debug("starting processing");
51
52      struct epoll_event events[MAX_EVENTS];
53
54      while (running) {
55          int n = epoll_wait(epollfd, events, MAX_EVENTS, -1);
56          if (n < 0) {
57              if (!running) {
58                  break;
59              }
60              log_fatal("epoll failed");
61          }
62
63          for (int i = 0; i < n; i++) {
64              if (events[i].data.fd == sockfd) {
65                  accept_connection();
66                  continue;
67              }
```

```
68
69                  use_fd(events[i].data.fd);
70              }
71          }
72
73          pthread_exit((void *) 0);
74      }
```

4 Исследовательский раздел

Предметом исследований является скорость и пропускная способность сервера при запросе картинки. Нагрузочное тестирование проводилось на примере изображения в формате PNG размером 2.8 Мбайт. Сравнивались разработанный сервер и Nginx [**dejonghe2020nginx**].

Характеристики устройства, на котором проводилось исследование, следующие [**macbook**]:

- оперативная память 16Гб;
- процессор Apple M2 Air;
- операционная система macOS Ventura 13.0.1.

Оба сервера были запущены в Docker [**docker**] со следующими ограничениями по ресурсам: оперативная память — 2Гб, ресурс процессора — 2000m.

Зависимости количества запросов, обрабатываемых серверов в секунду, от количества потоков представлена на рисунке 4.1.

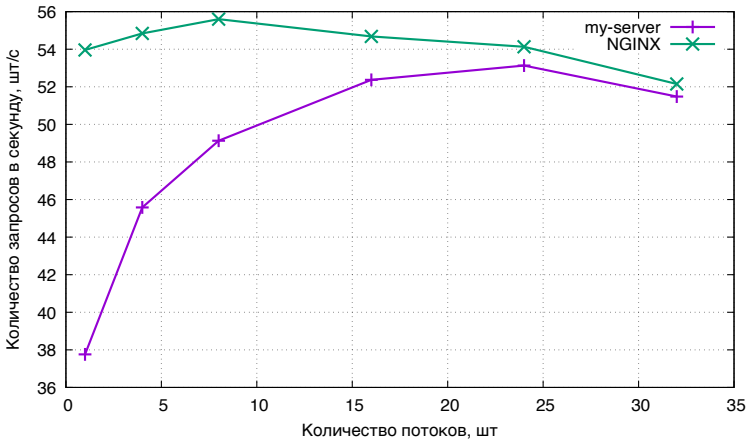

Рисунок 4.1 – Зависимости ... от количества потоков

По графику видно, что NGINX обрабатывает большее количество запросов в секунду, чем разработанный сервер. При 1 потоке он быстрее в 1.43 раз, при 32 потоках в 1.01 раз.

ЗАКЛЮЧЕНИЕ

Цель работы была выполнена: разработан статический сервер.

Для достижения поставленной цели были решены следующие задачи:

1. проведен обзор функциональности статического сервера;
2. проведен обзор компонентов, из которых состоит статический сервер;
3. описан алгоритм работы статического сервера.
4. реализован статический сервер;
5. проведено нагрузочное тестирования разработанного программного обеспечения, результаты сравнены с нагрузочным тестированием Nginx.