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 ИУ7 ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ЭВМ И ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ

КУРСОВАЯ РАБОТА

НА ТЕМУ:

Сервер для отдачи статического содержимого с диска (вариант №1)

Студент **ИУ7-73Б** (группа) _____ (подпись, дата) **Авдейкина В П** (И.О. Фамилия)

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

РЕКОМЕНДУЮ ___
_____ *М.Н. Клочков*

2024 г.

РЕФЕРАТ

Расчетно-пояснительная 23 с., 7 рис., 16 ист.

Ключевые слова: сервер, статическая информация, HTTP, нагрузочное тестирование, NGINX, мультиплексирование, пул потоков.

Цель работы — разработка сервера для отдачи статического содержимого с диска на основе паттерна пул потоков с использованием системного вызова select.

Результатом работы является разработанное программное обеспечение, реализующее сервер отдачи статической информации с использованием технологии пул потоков на основе асинхронного блокирующего ввода-вывода. Разработанный сервер поддерживает некоторые статусы HTTP, имеет возможность обрабатывать запросы GET, HEAD, выполняет логирование.

Проведено нагрузочное тестирование разработанного программного обеспечения с использованием Apache Benchmarks и выполнено сравнение результатов с результатами тестирования NGINX.

Согласно результатам сравнения, более производительным является NGINX.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью работы является разработка сервера для отдачи статического содержимого с диска на основе паттерна пул потоков (англ. `thread pool`) с использованием системного вызова `select`. Сервер должен поддерживать запросы `GET`, `HEAD` (со статусами 200, 403, 404), отвечать на неподдерживаемые запросы статусом 405, записывать информацию о событиях, осуществлять корректную передачу файлов размером до 128 Мб.

Для достижения поставленной цели требуется выполнить следующие задачи:

1) проанализировать предметную область, определить функции сервера для отдачи статического содержимого;
2) спроектировать сервер;
3) разработать спроектированное программное обеспечение;
4) провести нагрузочное тестирование реализованного сервера при помощи Apache Benchmarks;
5) сравнить результаты с результатами тестирования сервера NGINX.

1 Аналитическая часть

На основе цели работы формализуются требования к функционированию разрабатываемого сервера:

— поддержка обработки запросов `GET`, `HEAD` — осуществление отправки ответов со статусами 200, 403, 404;

— отправка ответа со статусом 405 в случае получения запроса, поддержка которого не предусмотрена;

— корректная передача файлов определенных выше форматов размером до 128 Мб;

— осуществление записи информации о событиях (логирования);

— использование паттерна пул потоков и системного вызова `select` при реализации.

1.1 Модель клиент-сервер

Клиент — процесс, который нуждается в обслуживании [1], [2].

Сервер — процесс, который предоставляет ресурсы и обслуживание одному или нескольким клиентам [1], [2]. Иногда в качестве сервера рассматривается отдельное множество процессов, запущенных на одной машине [3].

Клиент и сервер могут находиться как в одной сети, так и в разных; они обмениваются сообщениями по шаблону запрос-ответ. Взаимодействие клиента и сервера происходит поэтапно следующим образом:

1) клиент запрашивает соединение с сервером;
2) сервер принимает или отклоняет запрос на соединение (в случае отказа дальнейшие шаги не выполняются);
3) сервер устанавливает и поддерживает соединение с клиентом по определенному протоколу;
4) клиент запрашивает ресурсы или службу у сервера (отправляет запрос);
5) сервер получает и обрабатывает запрос, выполняя некоторые операции (обращение к базе данных, чтение файлов и так далее);
6) сервер возвращает ответ.

Существует два вида информации: статическая и динамическая [4]. Статической называется информация, которая относительно редко меняется с течением

времени [4]. В рамках решаемой задачи под статической информацией подразумеваются файлы следующих форматов с указанными в скобках расширениями:

— CSS (.css);

— JS (.js);

— HTML (.html);

— PNG, JPEG (.png, .jpeg, .jpg);

— SWF (.swf);

— GIF (.gif).

Сокет — абстракция конечной точки коммуникационного взаимодействия [5]. В языке Си он представляется файловым дескриптором. Соединения между клиентом и сервером устанавливаются при помощи сокетов.

Для создания сокета используется системный вызов socket [6], представленный на листинге 1 и принимающий на вход следующие параметры:

— `domain` — семейство (AF_UNIX, AF_INET и др.);

— `type` — тип (SOCK_STREAM, SOCK_RAW и др.);

— `protocol` — протокол.

Листинг 1 — Системный вызов socket

```
1 int socket(int domain, int type, int protocol);
```

1.2 Пул потоков

Пул потоков (англ. thread pool) — архитектурный паттерн, предназначенный для многопоточной обработки информации и основанный на хранении заранее созданных и используемых потоков [7].

Создание и настройка потока являются процессами, задерживающими работу приложения, поэтому для снижения накладных расходов был разработан пул потоков [7].

В данной технологии предлагается создать набор (пул) потоков определенной величины и сохранять его состояние на протяжении всей работы приложения. Для обработки каждого приходящего запроса используется очередной свободный поток из ранее созданного пула. Он помечается как занятый, а после обработки запроса — отмечается свободным (возвращается в пул).

Основные проблемы, возникающие при применении технологии [7]:

1) «простаивание» потоков;
2) обработка одним потоком нескольких запросов;
3) проблема определения числа потоков.

Для решения первых двух проблем необходимо использовать средства взаимоисключения (например, мьютексы) [5] и отдельный сокет на каждый поток.

Третья проблема решается указанием числа потоков при запуске сервера.

Схема использования пула потоков представлена на рисунке 1.

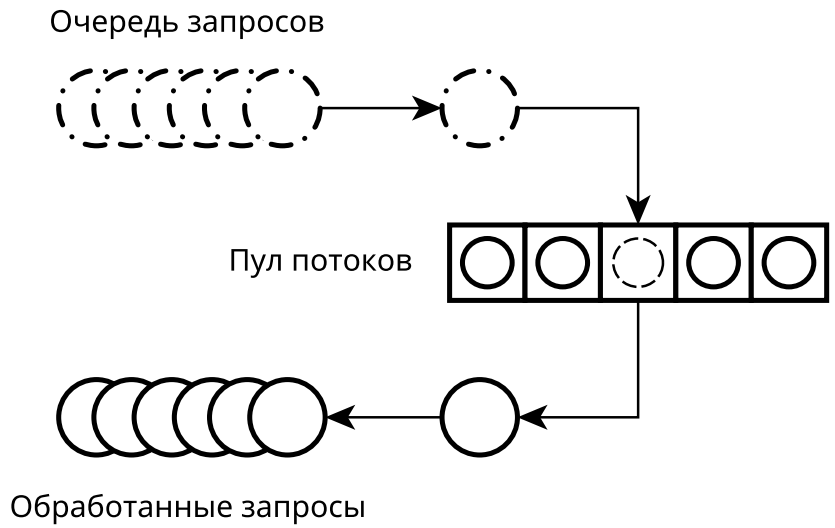

Рисунок 1 — Схема использования пула потоков

1.3 Асинхронный блокирующий ввод-вывод

Мультиплексирование (асинхронный блокирующий ввод-вывод) — процесс совмещения нескольких сообщений, передающихся одновременно, в одной логической или физической среде [5]. Оно основывается на одновременном (асинхронном) опросе и сборе информации о готовности источников. На момент опроса процесс блокируется в ожидании соединения. После установления соединения с активным источником начинается обработка данных. Следующая блокировка происходит в ожидании результата чтения, а затем выполняется передача данных из пространства ядра в пространство пользователя.

Модель асинхронного блокирующего ввода-вывода с использованием системного вызова select представлена на рисунке 2.

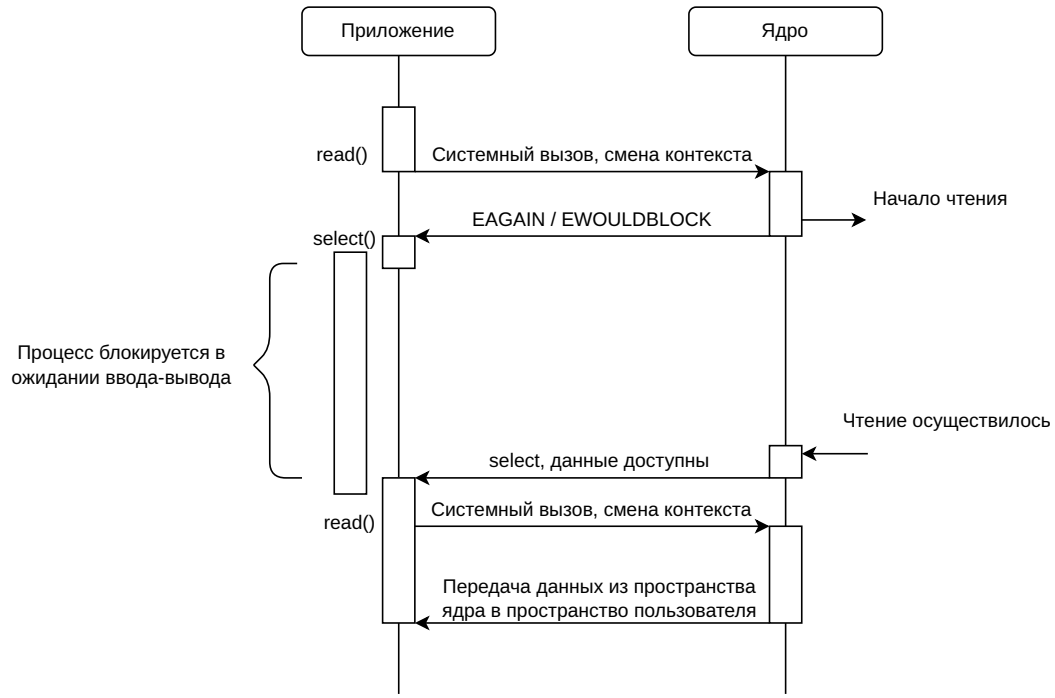

Рисунок 2 — Модель асинхронного блокирующего ввода-вывода

Системный вызов select, представленный на листинге 2, используется для отслеживания нескольких активных файловых дескрипторов. Параметры select:

— nfds — максимальное число файловых дескрипторов в перечисленных ниже массивах, увеличенное на единицу;
— readfds — массив активных файловых дескрипторов (для чтения);
— writefds — массив активных файловых дескрипторов (для записи);
— exceptfds — массив активных файловых дескрипторов (для иных ситуаций);
— timeout — структура timeval, описывающая временной интервал блокировки.

Листинг 2 — Системный вызов select

```
1 int select(int nfds, fd_set *_Nullable restrict readfds,
2            fd_set *_Nullable restrict writefds,
3            fd_set *_Nullable restrict exceptfds,
4            struct timeval *_Nullable restrict timeout);
```

1.4 Протоколы

Протокол — соглашение логического уровня между интерфейсами [8].

Представителями протоколов прикладного уровня являются HTTP, FTP, SMTP [9].

FTP — протокол передачи файлов по сети, предназначенный для обмена данными между хостами через промежуточные системы [10]. Данный протокол основан на модели клиент-сервер, гарантирует доставку данных. Соединения для управления и передачи между клиентом и сервером в нем разделены. Команды `GET`, `HEAD` данный протокол не поддерживает.

SMTP — протокол, предназначенный для передачи электронной почты [11]. Архитектура протокола соответствует модели «клиент-сервер». Принцип работы основан на обмене последовательностями, которые описываются с помощью `MAIL FROM`, `RCPT TO`, `DATA`.

HTTP — протокол для распределённых, объединённых, гипермедийных информационных систем [12]. Основой протокола является технология «клиент-сервер». Главный объект манипуляции — ресурс, на который указывает URI в запросе клиента. В рамках HTTP протокола TCP-сессия устанавливается на каждый запрос.

Таким образом, для достижения цели следует реализовать сервер, поддерживающий соединения по протоколу HTTP.

HTTP сообщение состоит из трех частей:

1) строки запроса (стартовая);
2) заголовков, описывающие параметры передачи и так далее;
3) тела сообщения.

Одним из заголовков является метод запроса, примерами которого являются `GET`, `HEAD`.

`GET` используется в запросах определенного ресурса; тело запроса в случае этого метода будет пустым, а в теле ответа будет содержаться запрошенная информация (содержимое интересующего файла).

`HEAD` аналогичен `GET`, служит для проверки наличия ресурса или извлечения метаданных, поскольку тело ответа на данный метод отсутствует.

1.5 Существующие решения

Веб-сервер — сервер, осуществляющий взаимодействие с клиентами по протоколу HTTP.

Согласно информации, представленной в [13], наиболее распространенными за последние 4 года веб-серверами являются NGINX, Cloudflare, Apache.

NGINX [14] — веб-сервер, прокси-сервер с открытым исходным кодом, поддерживающий Unix-подобные операционные системы и частично операционные системы семейства Windows. Он позиционируется как простой, быстрый и надёжный сервер, использование которого является целесообразным для раздачи статической информации. Основа работы данного сервера — обработка одним потоком нескольких запросов. По данным [13] к ноябрю 2024 года NGINX используется для поддержки 220 миллионов сайтов, что составляет 19% от общего количества.

Apache [15] — веб-сервер с открытым исходным кодом, поддерживающий большое количество операционных систем, в числе которых Unix-подобные ОС, ОС семейства Windows, OpenVMS. В ходе работы сервера каждый поток обрабатывает только один запрос. По данным [13] к ноябрю 2024 года Apache используется для поддержки 199 миллионов сайтов (17%).

Cloudflare [16] — американская компания, предоставляющая услуги по раздаче статической информации, защите от атак, предоставлению серверов DNS и проксированию сайтов. По данным [13] к ноябрю 2024 года услугами Cloudflare пользуются 134 миллионов сайтов (12%).

Таким образом, самым распространенным решением является NGINX. Сравнение разрабатываемого сервера по результатам тестирования будет проводиться с ним.

Вывод

Был проведен анализ предметной области сервера для отдачи статической информации. Для реализации будут использоваться соединения на основе протокола HTTP, пул потоков и системный вызов select (асинхронный блокирующий ввод-вывод). Результаты тестирования сервера будут сравниваться с результатами тестирования NGINX.

2 Конструкторская часть

В данном разделе рассматриваются схемы алгоритмов работы сервера отдачи статической информации и потока (из пула), который обрабатывает соединение. Схемы представлены графически на рисунках. Особенностью алгоритма сервера является бесконечный цикл и использование системного вызова select. Пул, который обрабатывает соединение, обрабатывает дополнительно HTTP коды.

2.1 Работа сервера

Схема алгоритма работы сервера представлена на рисунке 3.

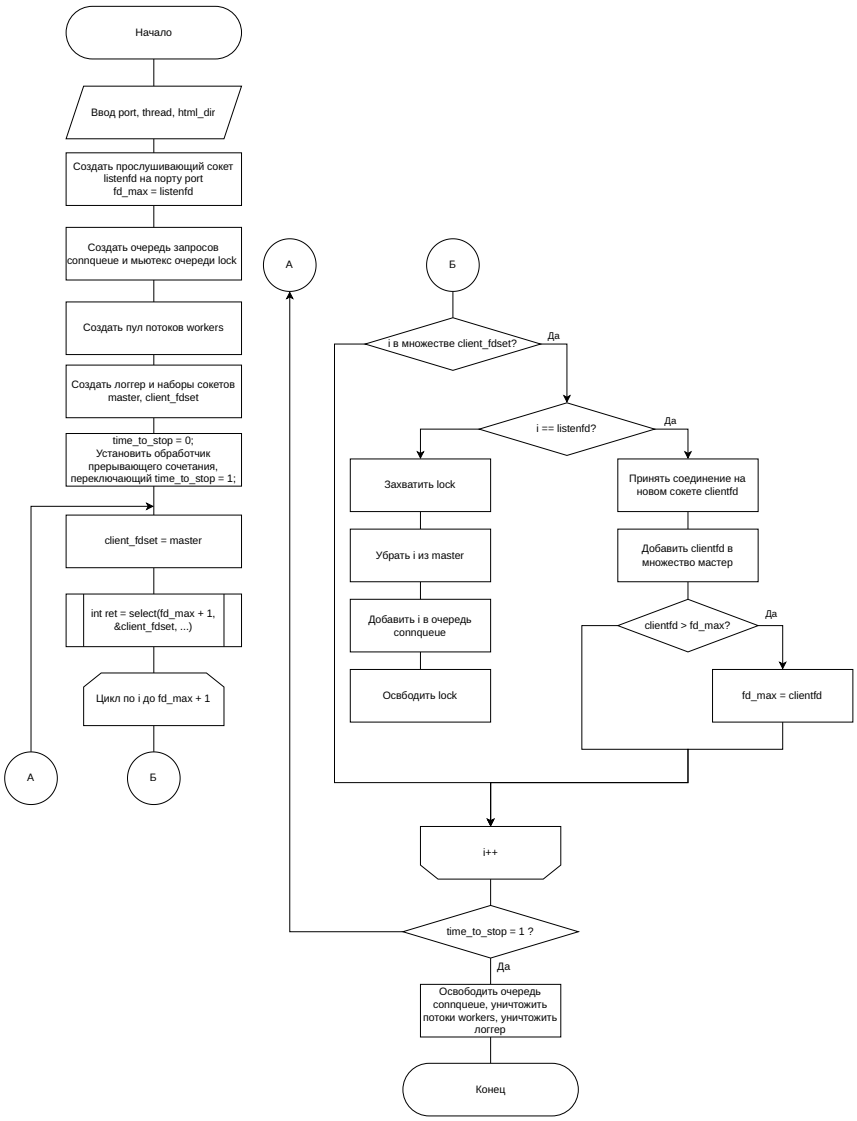

Рисунок 3 — Схема алгоритма работы сервера

2.2 Работа потока

На рисунке 4 представлена схема алгоритма работы потока.

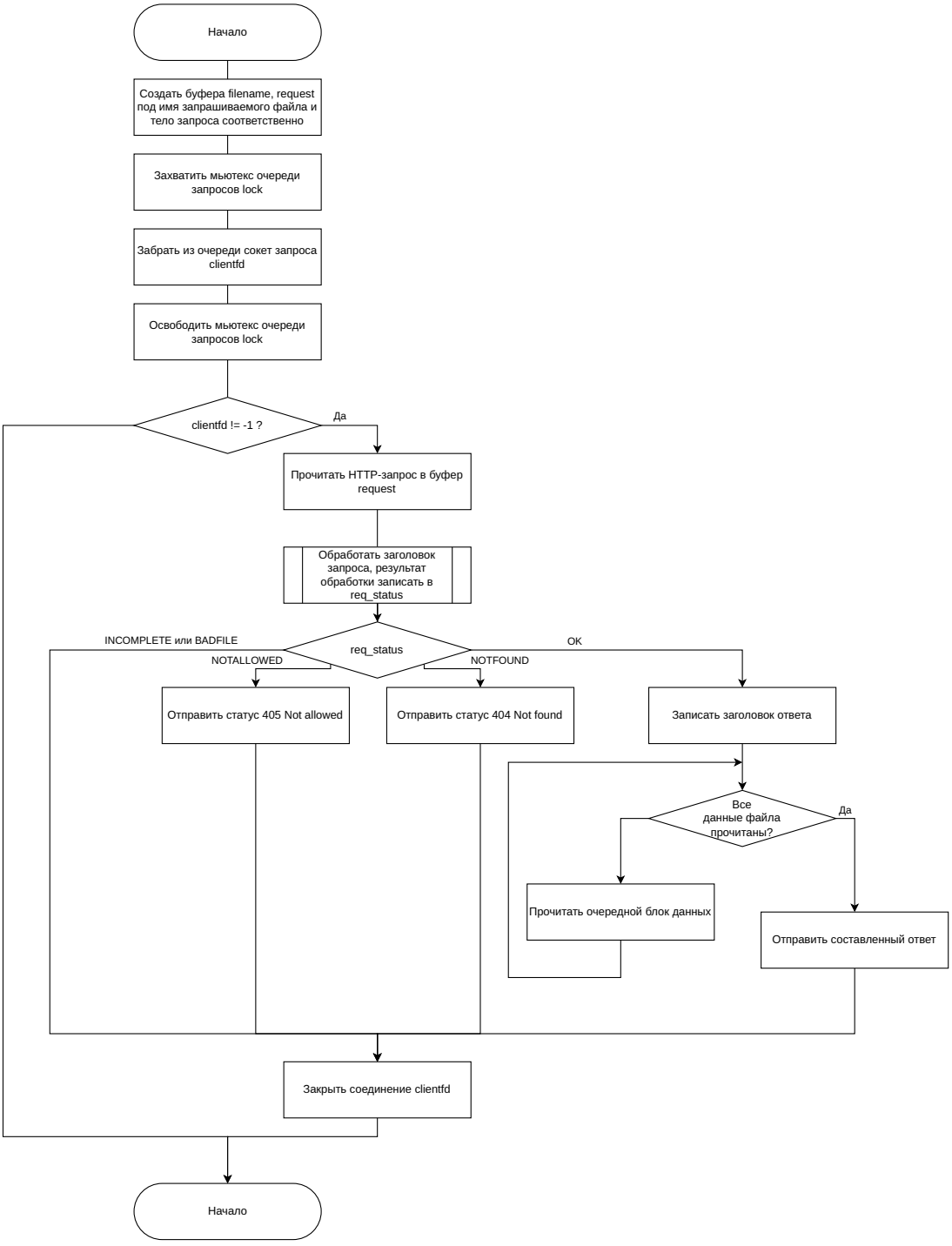

Рисунок 4 — Схема алгоритма работы потока из пула

3 Технологическая часть

В данном разделе рассматриваются выбранные средства реализации программного обеспечения. Также представлены листинги исходного кода и модули разработанного сервера. Реализация учитывает формализованные требования к разрабатываемому серверу и особенности задания на данную работу. Например, поддерживается технология пул потоков, а в основном цикле программы происходит обращение к системному вызову select.

3.1 Средства реализации и модули программы

В качестве языка программирования, используемого при реализации, выбран язык Си.

Причины выбора языка:

— наличие системного вызова select;

— соответствие требованиям задания на данную работу;

— наличие системных вызовов сетевого стека для работы с сокетами (socket, accept, listen и так далее).

Программа представлена следующими модулями:

1) main (содержит точку входа в программу, цикл с select);
2) thread_pool (содержит функцию создания пула потоков);
3) logger (содержит функции записи информации о событиях в программе);
4) server (содержит функции обработки запросов);
5) http (содержит константы HTTP протокола).

Получение исполняемого файла происходит при помощи компилятора gcc.

Сборка проекта происходит с использованием утилиты make. Пример правила из файла Makefile приведен на листинге 3.

Листинг 3 — Правило make

```
server: out/main.o out/logger.o out/thread_pool.o out/server.o
	$(CC) -o $@ $^ -pthread

out/logger.o: src/logger.c inc/logger.h
	$(CC) $(CFLAGS) -o $@ -c $<
```

3.2 Реализация сервера

На листинге 4 представлена первая часть куска реализации точки входа в программу с циклом обработки запросов, который содержит select.

Листинг 4 — Часть реализации точки входа в программу с основным циклом (часть 1)

```
int main(int argc, char *argv[])
{
  ...
    if ((workers = create_threadpool(threads, handle_connection)) ==
        NULL) {
      close(listenfd);
      pthread_mutex_destroy(&(connqueue->lock));
      exit(EXIT_FAILURE);
    }
...
  while (run) {
    if (time_to_stop) {
      break;
    }
    client_fdset = master;
    int ret = select(fd_max+1, &client_fdset, NULL, NULL, &tv);
    if (ret < 0)
      continue;
    for (int i = 0; i <= fd_max; i++) {
      if (FD_ISSET(i, &client_fdset)) {
        if (i == listenfd) {
          memset(&client_addr, 0, sizeof(client_addr));
          len = sizeof(client_addr);
          clientfd = accept(listenfd, (struct sockaddr*)&client_addr,
              &len);
          if (clientfd < 0) {
            perror("error accept\n");
            continue;
          }
          FD_SET(clientfd, &master);
          if (clientfd > fd_max)
            fd_max = clientfd;
        }
```

На листинге 5 представлена вторая часть куска реализации точки входа в программу. Реализация функции (часть 1) работы потока из пула представлена на листинге 6.

Листинг 5 — Часть реализации точки входа в программу с основным циклом (часть 2)

```
        else {
          pthread_mutex_lock(&(connqueue->lock));
          FD_CLR(i, &master);
          if (enqueue(connqueue, i) < 0) {
            printf("drop conn\n");
            close(clientfd);
          }
          pthread_mutex_unlock(&(connqueue->lock));
        }
      }
    }
  }
...
```

Листинг 6 — Часть реализации функции работы потока из пула (часть 1)

```
void* handle_connection(void *args)
{
  ...
  while (run) {
    ...
    pthread_mutex_lock(&(connqueue->lock));
    clientfd = dequeue(connqueue);
    pthread_mutex_unlock(&(connqueue->lock));
    ...
    char *message_to_log = calloc(FILE_BUFFER_SIZE, sizeof(char));
    br = read(clientfd, req_buffer, REQUEST_BUFFER_SIZE-1);
    if (br <= 0) {
      bw = write(clientfd, HTTP_403, HTTP_403_len);
      if (bw > 0)
        total_wrote += bw;
      snprintf(message_to_log, FILE_BUFFER_SIZE, "%s, %d",
          strerror(errno), clientfd);
      add_to_log(message_to_log);
      close_clientfd(clientfd, tindex, total_wrote);
      continue;
    }
```

Реализация функции (часть 2) работы потока из пула представлена на листинге 7.

Листинг 7 — Часть реализации функции работы потока из пула (часть 2)

```
    ...
    switch(req_status)
    {
      case INCOMPLETE:
        ...
      case FORBIDDEN:
        ...
      case NOTALLOWED:
        ...
      case NOTFOUND:
        ...
      case OK: {
        bw = write(clientfd, response_header, strlen(response_header));
        char filename[FILE_NAME_SIZE] = {'\0'};
        int method = get_file_name(req_buffer, filename, NULL);
        if(bw <= 0) {
          close_clientfd(clientfd, tindex, total_wrote);
          continue;
        }
        total_wrote += bw;
        int flag = 0;
        while(method != HTTP_HEAD && (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;
}
```

4 Исследовательская часть

В текущем разделе приведены результаты нагрузочного тестирования разработанного программного обеспечения с использованием Apache Benchmarks. Также приведены результаты сравнения с тестированием NGINX. Сравнение с сервером NGINX проводится по причине его первенства среди рассмотренных существующих решений. Описываются характеристики машины, на которой производится тестирование.

4.1 Описание тестирования

Нагрузочное тестирование осуществляется с использованием Apache Benchmarks. Время обработки замерялось от 100 до 1000 запросов с шагом 100 при подключении 5, 50, 100 клиентов. Тестировался разработанный сервер, а также NGINX.

Технические характеристики машины, на которой проводилось тестирование:

— операционная система — Ubuntu 24.04.1 LTS;

— процессор — 13th Gen Intel® Core™ i7-1360P × 16;

— оперативная память — 16,0 Гб LPDDR5 с тактовой частотой 4800 Гц;

— графическая карта — Intel® Iris® Xe Graphics (RPL-P).

Тестирование выполнялось на ноутбуке, являющимся подключенным к сети электропитания. Во время тестирования ноутбук был нагружен только системой тестирования (работающим приложением) и окружением операционной системы.

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

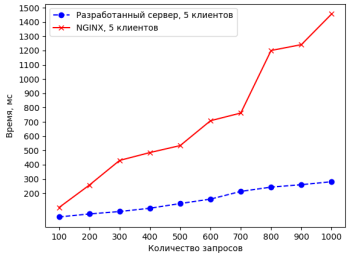

Рисунок 5 — Результаты тестирования (5)

Результаты тестирования показывают, что разработанная программа обрабатывает запросы на статическую информацию быстрее, чем NGINX. Разница во времени обработки тем больше, чем меньше количество подключённых клиентов: при 1000 запросах и 5 клиентах разработанный сервер работает в 7 раз быстрее, а при 50 клиентах — в 3 раза. Это объясняется тем, что при тестировании количество запросов оставалось постоянным (от 100 до 1000 с шагом 100), поэтому суммарное число запросов на одного клиента уменьшалось с ростом числа клиентов.

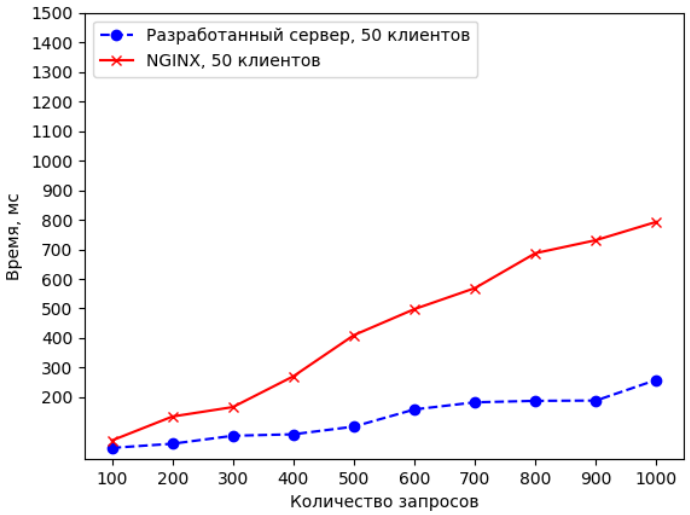

Рисунок 6 — Результаты тестирования (5)

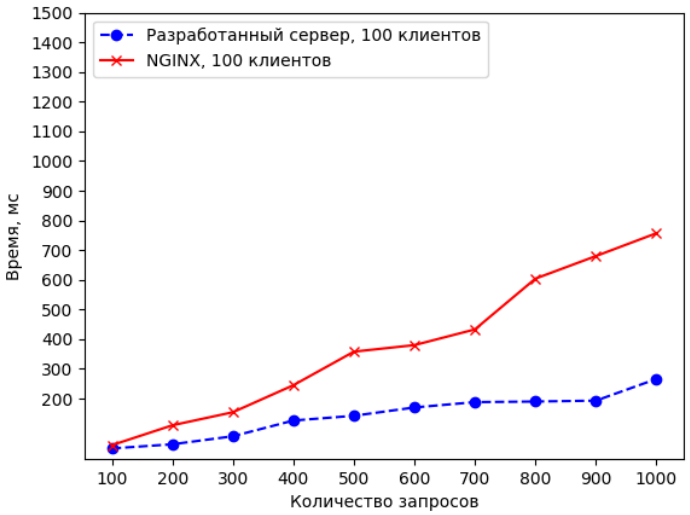

Рисунок 7 — Результаты тестирования (5)

ЗАКЛЮЧЕНИЕ

В ходе работы цель была достигнута — разработан сервер для отдачи статического содержимого с диска на основе паттерна пул потоков (англ. `thread pool`) с использованием системного вызова `select`.

Были выполнены следующие задачи:

1) проанализирована предметная область, определены функции сервера для отдачи статического содержимого;
2) спроектирован сервер;
3) разработано спроектированное программное обеспечение;
4) протестировано реализованное программное обеспечение при помощи Apache Benchmarks;
5) проведено сравнение результатов с результатами тестирования сервера NGINX.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Kumar S. A Review on Client-Server based applications and research opportunity // International Journal of Recent Scientific Research. — 2019. — Т. 10, № 7. — С. 33857—3386.
2. Oluwatosin H. S. Client-server model // IOSR Journal of Computer Engineering. — 2014. — Т. 16, № 1. — С. 67—71.
3. Huhta A. Multi-platform data processing engine. — 2021.
4. Аршакян А., Клещарь С., Ларкин Е. Оценка статических потерь информации в сканирующих устройствах // Известия Тульского государственного университета. Технические науки. — 2013. — № 3. — С. 388—395.
5. Рязанова Н. Ю. Операционные системы : неопубликованный конспект лекций. — 2023.
6. Kerrisk M. The Linux programming interface: a Linux and UNIX system programming handbook. — No Starch Press, 2010.
7. Опарин И. А. Многопоточность в языке программирования C#. Пул потоков // StudNet. — 2022. — Т. 5, № 6. — С. 7131—7139.
8. Тихомирова Е. А. Компьютерные сети : неопубликованный конспект лекций. — 2024.
9. Lombardi M., Pascale F., Santaniello D. Internet of things: A general overview between architectures, protocols and applications // Information. — 2021. — Т. 12, № 2. — С. 87.
10. Forouzan B. A. TCP/IP Protocol Suite. — McGraw-Hill Higher Education, 2002.
11. Johnson K. Internet email protocols: a developer's guide. — Addison-Wesley Longman Publishing Co., Inc., 2000.
12. Fielding R. T., Nottingham M., Reschke J. HTTP/1.1 [Электронный ресурс]. — 06.2022. — Режим доступа: https://www.rfc-editor.org/info/rfc9112 (дата обращения: 12.12.2024).
13. November 2024 Web Server Survey [Электронный ресурс]. — Режим доступа: https://www.netcraft.com/blog/november-2024-web-server-survey/ (дата обращения: 11.12.2024).

14. nginx : официальный сайт [Электронный ресурс]. — Режим доступа: https: //nginx.org/ (дата обращения: 12.12.2024).

15. The Apache HTTP Server Project : официальный сайт [Электронный ресурс]. — Режим доступа: https : / / httpd . apache . org/ (дата обращения: 12.12.2024).

16. Cloudflare : официальный сайт [Электронный ресурс]. — Режим доступа: https://www.cloudflare.com/ (дата обращения: 12.12.2024).

ПРИЛОЖЕНИЕ А

Презентация к курсовой работе состоит из 10 слайдов