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 _____ Виноградов А. О.
(Подпись, дата) (Фамилия И. О.)

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

2023 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 теме

«Разработка статического сервера»

Студент группы ИУ7-76Б

Виноградов Алексей Олегович

Направленность КР

учебная

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

Техническое задание

Разработать статический веб-сервер для отдачи контента с диска. В качестве мультиплексора использовать pselect. Сервер должен реализовывать многопоточную обработку запросов с использованием технологии prefork.

Оформление научно-исследовательской работы:

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

Дата выдачи задания «16» сентября 2023 г.

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

Студент _____ Виноградов А. О.
(Подпись, дата) (Фамилия И. О.)

СОДЕРЖАНИЕ

ВВЕДЕНИЕ **4**

1 Аналитический раздел **5**

1.1 Статический сервер 5

1.2 Готовые решения 6

1.3 Шаблон prefork 6

1.4 Мультиплексирование 7

1.4.1 Системный вызов pselect 7

1.5 HTTP 8

1.6 Требования к разрабатываемой программе 9

2 Конструкторский раздел **11**

2.1 Разработка алгоритмов 11

3 Технологический раздел **13**

3.1 Средства реализации 13

3.2 Реализация алгоритмов 13

3.3 Пример работы программы 17

4 Исследовательский раздел **19**

4.1 Технические характеристики 19

4.2 Нагрузочное тестирование 19

ЗАКЛЮЧЕНИЕ **22**

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 prefork и системный вызов pselect для мультиплексирования. Для достижения поставленной цели необходимо выполнить следующие задачи:

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

1 Аналитический 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) [?] — 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 [?] nginx обслуживал или проксировал 20.72% самых нагруженных сайтов в декабре 2023 года.

Apache (от англ. a patchy server) [?] — 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 [?]. 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 prefork

Шаблон управления потоками prefork (предварительный запуск процессов) является альтернативой пулу потоков и используется в том числе в веб-серверах.

В отличие от пула потоков, где создается набор потоков при запуске программы, в шаблоне pefork процессы создаются заранее. Каждый процесс может обрабатывать входящие запросы, и родительский процесс управляет количеством процессов в пуле.

Этот подход подходит для серверов, которым необходимо избегать потоков из-за несовместимости с непотоково-безопасными библиотеками.

Шаблон prefork на статическом сервере работает путем создания заранее определенного количества процессов, которые ожидают и обслуживают входящие запросы на статические ресурсы, такие как HTML, CSS и изображения. Когда запрос поступает, один из доступных процессов берет на себя обработку запроса, что позволяет обрабатывать несколько запро-

сов параллельно. Это позволяет увеличить производительность сервера за счет параллельной обработки запросов.

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.

1.4.1 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 — максимальное 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.

1.5 HTTP

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

1.6 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 prefork и pselect;
10. статик сервер должен работать стабильно.

Вывод

В данном разделе был проведён анализ предметной области, анализ шаблона управления потоками prefork, модели асинхронного блокирующего ввода-вывода, системного вызова pselect, 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 1 представлена схема алгоритма работы сервера.

Вывод

В данном разделе были представлена схема алгоритма работы сервера.

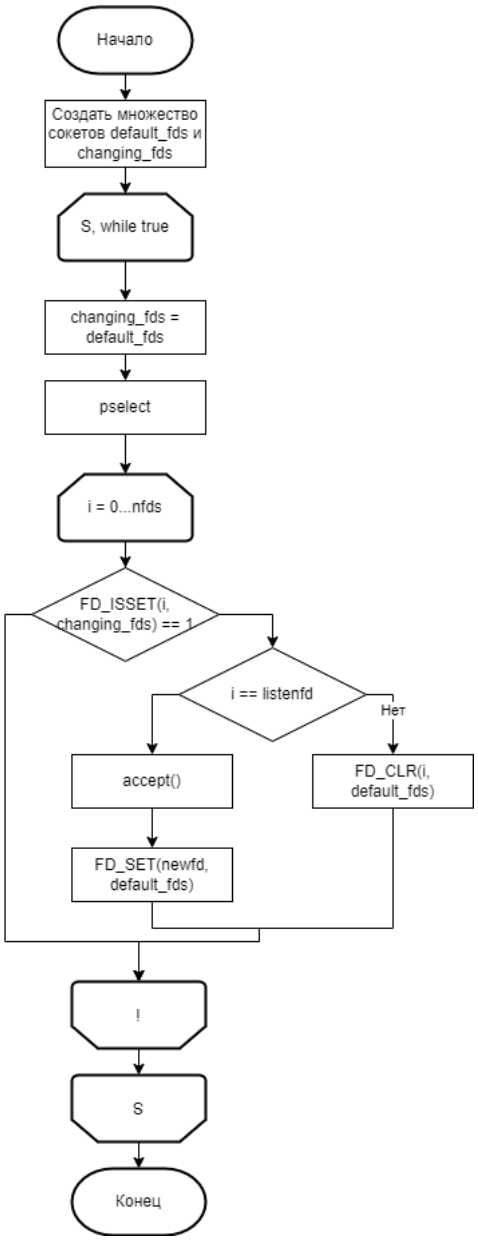

Рисунок 1 – Схема алгоритма работы сервера

3 Технологический раздел

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

3.1 Средства реализации

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

3.2 Реализация алгоритмов

В листингах ??-?? представлена реализация основного алгоритма работы сервера. А в листингах ??-?? представлена реализация обработки запросов на сервер.

Листинг 1 – Основной алгоритм работы сервера

```
1  int init_server() {
2      close(STDIN_FILENO);
3      close(STDERR_FILENO);
4      signal(SIGCHLD, sigchld_handler);
5      worker_count = get_nprocs();
6
7      servsock = socket(AF_INET, SOCK_STREAM, 0);
8      if (servsock == -1) {
9          log_error("Cannot create socket");
10         return EXIT_FAILURE;
11     }
12     if (setsockopt(servsock, SOL_SOCKET, SO_REUSEADDR, &(int){ 1
           }, sizeof(int)) == -1) {
13         close(servsock);
```

Листинг 2 – Основной алгоритм работы сервера (продолжение)

```
1         log_error("Cannot␣set␣socket␣to␣reuse␣address");
2         return EXIT_FAILURE;
3     }
4     struct sockaddr_in servaddr;
5     servaddr.sin_family = AF_INET;
6     servaddr.sin_addr.s_addr = inet_addr(HOST);
7     servaddr.sin_port = htons(PORT);
8     if (bind(servsock, (struct sockaddr *)&servaddr,
         sizeof(servaddr)) == -1) {
9         close(servsock);
10        log_error("Cannot␣bind␣address");
11        return EXIT_FAILURE;
12    }
13
14    max_fd = servsock;
15    rfds = malloc(sizeof(fd_set));
16    FD_ZERO(rfds);
17    FD_SET(servsock, rfds);
18    workers = malloc(sizeof(worker_t) * worker_count);
19    for (size_t i = 0; i < worker_count; i++) {
20        workers[i].servsock = servsock;
21        if (fork_worker(workers + i) == EXIT_FAILURE) {
22            free(workers);
23            return EXIT_FAILURE;
24        }
25    }
26    return EXIT_SUCCESS;
27 }
```

Листинг 3 – Обработка запроса сервером

```
1  void handle_connection(connection_t *connection) {
2      connection->is_served = 1;
3      int fd = connection->fd;
4      memset(connection->request.buffer, 0, REQUEST_SIZE_LIMIT + 1);
5      read(fd, connection->request.buffer, REQUEST_SIZE_LIMIT - 1);
6
7
8      sscanf(
9      connection->request.buffer, "%s %s",
           connection->request.method, connection->request.path);
10
11     if (strcmp(connection->request.method, "GET") == 0) {
12         log_info("GET request");
13         struct stat sb;
14         char actual_path[PATH_SIZE_LIMIT + 3];
15         if (strcmp(connection->request.path, "/") == 0) {
16             sprintf(actual_path, "./index.html");
17         } else {
18             sprintf(actual_path, ".%s", connection->request.path);
19         }
20
21         int rc;
22         if ((rc = stat(actual_path, &sb)) == 0) {
23             const char *mime_type = detect_mime_type(actual_path);
24             sprintf(connection->response.status,
                   RESPONSE_TEMPLATE, 200, "OK");
25             sprintf(connection->response.headers,
                   CONTENT_TYPE_TEMPLATE, mime_type, sb.st_size);
26         } else {
27             sprintf(connection->response.status,
                   RESPONSE_TEMPLATE, 404, "Not Found");
28             write(fd, connection->response.status,
                   strlen(connection->response.status));
29         }
30         if (rc == 0) {
31             FILE *file = fopen(actual_path, "rb");
32             if (file == NULL && errno == EACCES) {
33                 sprintf(connection->response.status,
                       RESPONSE_TEMPLATE, 403, "Forbidden");
34             }
```

Листинг 4 – Обработка запроса сервером (продолжение)

```
1             write(fd, connection->response.status,
                 strlen(connection->response.status));
2             write(fd, connection->response.headers,
                 strlen(connection->response.headers));
3             write_file(file, fd);
4             fclose(file);
5         }
6
7     } else if (strcmp(connection->request.method, "HEAD") == 0) {
8         log_info("HEAD request");
9         struct stat sb;
10        char actual_path[PATH_SIZE_LIMIT + 3];
11        sprintf(actual_path, ".%s", connection->request.path);
12        if (stat(actual_path, &sb) == 0) {
13            const char *mime_type = detect_mime_type(actual_path);
14            sprintf(connection->response.status,
                 RESPONSE_TEMPLATE, 200, "OK");
15            sprintf(connection->response.headers,
                 CONTENT_TYPE_TEMPLATE, mime_type, sb.st_size);
16        } else {
17            sprintf(connection->response.status,
                 RESPONSE_TEMPLATE, 404, "Not Found");
18        }
19
20        write(fd, connection->response.status,
             strlen(connection->response.status));
21        write(fd, connection->response.headers,
             strlen(connection->response.headers));
22    } else {
23        log_info("Unknown request");
24        sprintf(connection->response.status, RESPONSE_TEMPLATE,
             405, "Method Not Allowed");
25        write(fd, connection->response.status,
             strlen(connection->response.status));
26    }
27}
```

3.3 Пример работы программы

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

Вывод

В этом разделе был выбран язык программирования, 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 [**?**];
2. память 8 ГБ;
3. процессор 2,4 ГГц 4-ядерный процессор Intel Core i5-1135G7 [**?**].

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:8080.

Команда для единовременной отправки 100 запросов на сервер nginx: **ab -n 10000 -c 100 http://localhost:8080**.

На рисунке **??** представлены результаты измерений.

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 prefork и системного вызова pselect для мультиплексирования.

В ходе выполнения курсовой работы были решены следующие задачи:

1) проведён анализ предметной области;
2) проведён анализ шаблона prefork и системного вызова pselect;
3) спроектирован и разработан сервер раздачи статической информации;
4) проведено сравнение результатов нагрузочного тестирования при помощи apache benchmark разработанного сервера с nginx;