

Министерство образования Республики Беларусь  
Учреждение образования «Белорусский государственный университет  
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Дисциплина: «Операционные системы и системное программирование»

К ЗАЩИТЕ ДОПУСТИТЬ

\_\_\_\_\_ Л. П. Поденок

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему

«МНОГОПОТОЧНЫЙ HTTP-СЕРВЕР»

БГУИР КП 1-40 02 01 203 ПЗ

Студент:

Бейнар А.В.

Руководитель:

Л. П. Поденок

Минск 2023

Учреждение образования  
«Белорусский государственный университет информатики и радиоэлектроники»  
Факультет компьютерных систем и сетей

УТВЕРЖДАЮ

Заведующий кафедрой ЭВМ

\_\_\_\_\_ Б. В. Никульшин

(подпись)

\_\_\_\_\_ 2023г.

ЗАДАНИЕ

по курсовому проектированию  
Бейнара Александра Владимировича

- 1 Тема проекта: «Многопоточный HTTP-сервер».
- 2 Срок сдачи студентом законченного проекта: 5 июня 2023 г.
- 3 Исходные данные к проекту:
  - 3.1 Протокол взаимодействия: HTTP.
  - 3.2 Формат обмена данными: JSON.
  - 3.3 Среда разработки: Visual Studio Code.
  - 3.4 Язык программирования: C.
  - 3.5 Операционная система: Linux.
- 4 Содержание пояснительной записки (перечень подлежащих разработке вопросов):

Введение 1. Обзор литературы. 2. Системное проектирование. 3. Функциональное проектирование. 4. Разработка программных модулей. 5. Руководство пользователя. Заключение. Список использованных источников. Приложения.
- 5 Перечень графического материала (с точным указанием обязательных чертежей):
  - 5.1 Многопоточный HTTP-сервер.  
Схема структурная.
  - 5.2 Многопоточный HTTP-сервер.  
Схема программы.

### КАЛЕНДАРНЫЙ ПЛАН

Наименование этапов дипломного проекта	Объем этапа, %	Срок выполнения этапа	Примечания
Подбор и изучение литературы. Сравнение аналогов.	10	23.03 – 05.04	
Структурное проектирование	15	05.04 – 12.04	
Функциональное проектирование	25	12.04 – 24.04	
Разработка программных модулей	20	24.04 – 08.05	
Программа и методика испытаний	10	8.05 – 15.05	
Оформление пояснительной записки	15	20.05 – 30.05	

Дата выдачи задания: 15 февраля 2023 г.

Защита курсового проекта с 23 мая 2023 г. по 12 июня 2023 г.

РУКОВОДИТЕЛЬ

Л.П.Поденок

ЗАДАНИЕ ПРИНЯЛ К ИСПОЛНЕНИЮ \_\_\_\_\_

А. В. Бейнар

(дата и подпись студента)

## СОДЕРЖАНИЕ

ВВЕДЕНИЕ .....	5
1 ОБЗОР ЛИТЕРАТУРЫ .....	6
1.1 Основные термины.....	6
1.2 Анализ существующих аналогов.....	9
1.2.1 Веб-сервер и почтовый прокси-сервер «Nginx» .....	9
1.2.2 Веб-сервер «Apache».....	10
1.2.3 Веб-сервер «Cherokee». ....	11
1.3 Выбор технологий для создания проекта .....	12
1.4 Описание основных функций создаваемого программного обеспечения ....	12
2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ.....	13
2.1 Общее структурное описание состава программного обеспечения .....	13
2.2 Краткое описание модулей.....	13
2.2.1 Блок для работы с базой данных .....	13
2.2.2 Блок для работы с сервером.....	13
2.2.3 Блок отладочной информации .....	13
2.2.4 Блок для работы с сокетами и сетью.....	13
3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ .....	14
3.1 Описание основных функций программы.....	14
4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ .....	15
4.1 Алгоритм чтения файлов и каталогов.....	15
4.2 Алгоритм извлечения основной информации из запроса.....	15
4.3 Алгоритм отправки ответа .....	16
5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ.....	17
5.1 Минимальные программно-аппаратные требования к установке и запуску программы.....	17
5.2 Краткое описание основных действий пользователя при пользовании программой .....	17
5.3 Тестирование и проверка работоспособности программного средства .....	22
ЗАКЛЮЧЕНИЕ .....	24
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ.....	25
ПРИЛОЖЕНИЕ А .....	26
ПРИЛОЖЕНИЕ Б.....	27
ПРИЛОЖЕНИЕ В .....	28
ПРИЛОЖЕНИЕ Г .....	29

## ВВЕДЕНИЕ

В современном мире HTTP-сервер является важной частью в жизни нынешнего поколения. Люди каждый день пользуются HTTP-сервером. Данный сервер подразумевает под собой программу, которая понимает, что необходимо клиенту и выдаёт ему ответы на запрос в виде HTML страниц, на которых может содержаться различная информация: изображения, тексты, файлы и многое другое.

Целью курсового проекта является создание сервера на языке C. Овладения практическими навыками проектирования и разработки программного приложения. Закрепить теоретические знания, полученные при изучении курса «Операционные системы и системное программирование».

С ростом масштабов современных предприятий сложно контролировать их работу используя лишь бумагу и ручку. Для сложных многоуровневых процессов, необходимы простые в использовании, но многозадачные автоматизирующие программы.

В данном курсовом проекте будет представлено моё виденье простого в использовании многопоточного HTTP-сервера.

Данная тема является актуальной, так как разработанный программный продукт предоставляет пользователю возможность получить HTTP-сервер для операционной системы Linux.

# 1 ОБЗОР ЛИТЕРАТУРЫ

## 1.1 Основные термины

Многопоточность — способность центрального процессора (CPU) или одного ядра в многоядерном процессоре одновременно выполнять несколько процессов или потоков, соответствующим образом поддерживаемых операционной системой. Этот подход отличается от многопроцессорности, так как многопоточность процессов и потоков совместно использует ресурсы одного или нескольких ядер: вычислительных блоков, кэш-памяти ЦПУ или буфера перевода с преобразованием (TLB).

Многопоточные приложения используются в следующих случаях:

- для управления вводом от нескольких окон;
- для управления вводом от нескольких периферийных устройств или средств взаимодействия;
- для эффективного и гибкого использования системы приоритетов;
- для повышения интерактивности взаимодействия приложения и пользователя;
- для обработки данных в «фоновом» режиме.

Сокет — это средство связи, позволяющее разрабатывать клиент-серверные системы для локального, на одной машине, или сетевого использования.

HTTP (HyperText Transfer Protocol — «протокол передачи гипертекста») — протокол прикладного уровня передачи данных изначально — в виде гипертекстовых документов в формате «HTML», в настоящий момент используется для передачи произвольных данных. Каждое взаимодействие через HTTP включает в себя запрос и ответ. По своей природе HTTP не имеет состояний. Без состояния означает, что все запросы отделены друг от друга, а значит — каждый запрос должен содержать достаточно информации, чтобы полностью выполняться. Это означает, что каждая транзакция HTTP-модели, основанной на сообщениях, обрабатывается отдельно от остальных.

HTTP это прикладной протокол, который относится к семейству протоколов TCP/IP. Протокол TCP (или TLS - защищённый TCP) используется для пересылки своих сообщений, однако любой другой надёжный транспортный протокол теоретически может быть использован для доставки таких сообщений. Как правило, серверное программное обеспечение обычно использует TCP-порт 80 (и, если порт не указан явно, то обычно клиентское программное обеспечение по умолчанию использует именно 80-й порт для открываемых HTTP-соединений), хотя может использовать и любой другой.

Основой HTTP является технология «клиент-сервер», то есть предполагается существование:

- потребителей (клиентов), которые инициируют соединение и посылают запрос;

- поставщиков (серверов), которые ожидают соединения для получения запроса, производят необходимые действия и возвращают обратно сообщение с результатом.

URL (англ. Uniform Resource Locator — унифицированный указатель ресурса — система унифицированных адресов электронных ресурсов, или единообразный определитель местонахождения ресурса (файла). На рисунке 1.1 приведен пример URL.

`http://www.example.com/search?item=vw+beetle`

Protocol	Domain	Path	Parameters
----------	--------	------	------------

Рисунок 1.1 – Пример URL

Домен (Domain) — имя для идентификации одного или нескольких IP-адресов, на которых расположен ресурс.

Путь (Path) — указывает местоположение ресурса на сервере. Использует ту же логику, что и расположение ресурса на устройстве (например, /search/cars/VWBeetle.pdf или C:/mycars/VWBeetle.pdf).

Параметры (Parameters) — дополнительные данные, используемые для идентификации или фильтрации ресурса на сервере.

В HTTP каждый запрос должен иметь URL-адрес. К тому же, запросу необходим метод. Метод (запрос) HTTP — последовательность из любых символов, кроме управляющих и разделителей, указывающая на основную операцию над ресурсом. Обычно метод представляет собой короткое английское слово, записанное заглавными буквами.

Основные методы работы с HTTP:

- GET — используется для запроса содержимого указанного ресурса;
- HEAD — аналогичен методу GET, за исключением того, что в ответе сервера отсутствует тело;
- POST — применяется для передачи пользовательских данных заданному ресурсу (например, HTML-форму);
- PUT — применяется для загрузки содержимого запроса на указанный в запросе URI;
- DELETE — удаляет указанный ресурс;

Все HTTP-сообщения имеют один или несколько заголовков, за которыми следует необязательное тело сообщения. Тело содержит данные, которые будут отправлены с запросом, или данные, полученные с ответом.

Первая часть каждого HTTP-запроса содержит три элемента:

1. Первая часть — это метод, который сообщает, какой метод HTTP используется. Чаще всего используется метод GET. Метод GET извлекает ресурс с веб-сервера, и поскольку у GET нет тела сообщения, после заголовка ничего не требуется

2. Вторая часть — запрошенный URL.

3. Третья часть — используемая версия HTTP. Версия 1.1. является наиболее распространенной для большинства браузеров, однако, версия 2.0 становится популярнее.

Код состояния HTTP — часть первой строки ответа сервера при запросах по протоколу HTTP. Он представляет собой целое число из трёх десятичных цифр. Первая цифра указывает на класс состояния. За кодом ответа обычно следует отделённая пробелом поясняющая фраза на английском языке, которая разъясняет человеку причину именно такого ответа.

Клиент узнаёт по коду ответа о результатах его запроса и определяет, какие действия ему предпринимать дальше.

В настоящее время выделено пять классов кодов состояния:

- 1xx - информационный;
- 2xx - успешный;
- 3xx - перенаправление;
- 4xx - ошибка клиента;
- 5xx - ошибка сервера.

Заголовки HTTP — это строки в HTTP-сообщении, содержащие разделённую двоеточием пару параметр-значение. Формат заголовков соответствует общему формату заголовков текстовых сетевых сообщений ARPA.

API многих программных продуктов также подразумевает использование HTTP для передачи данных — сами данные при этом могут иметь любой формат, например, XML или JSON.

Клиент (user agent) — это любой инструмент или устройство, действующие от лица пользователя. Эту задачу преимущественно выполняет веб-браузер; в некоторых случаях участниками выступают программы, которые используются инженерами и веб-разработчиками для отладки своих приложений.

Чтобы отобразить веб-страницу, браузер отправляет начальный запрос для получения HTML-документа этой страницы. После этого браузер изучает этот документ и запрашивает дополнительные файлы, необходимые для отображения содержания веб-страницы (исполняемые скрипты, информацию о макете страницы - CSS таблицы стилей, дополнительные ресурсы в виде изображений и видео-файлов), которые непосредственно являются частью исходного документа, но расположены в других местах сети. Далее браузер соединяет все эти ресурсы для отображения их пользователю в виде единого документа — веб-страницы. Скрипты, выполняемые самим браузером, могут получать по сети дополнительные ресурсы на последующих этапах обработки веб-страницы, и браузер соответствующим образом обновляет отображение этой страницы для пользователя.

Браузер отправляет введённый URL-адрес в DNS, преобразователь URL-адресов в IP-адреса. DNS расшифровывается как «доменная система имён» (Domain Name System), и его можно представить как огромную таблицу со всеми зарегистрированными именами для сайтов и их IP-адресами. DNS возвращает



браузеру IP-адрес, с которым тот уже умеет работать. Теперь браузер начинает составлять HTTP-запрос с вложенным в него IP-адресом.

## 1.2 Анализ существующих аналогов

В ходе анализа существующих программных аналогов было выяснено, что на персональный компьютер есть возможность приобрести или установить приложение, объединяющее в себе все функции, представленные в данном проекте. Из них некоторые уже встроены в различные проводники, а другие работают как кроссплатформенные приложения.

### 1.2.1 Веб-сервер и почтовый прокси-сервер «Nginx»

Данный сервис находится в открытом доступе и скачать его можно на сайте [nginx.org](http://nginx.org). Nginx — веб сервер и почтовый прокси-сервер, разработанный российским программистом, который его активно продвигает, сейчас даже появилась компания Nginx, Inc [3].

Nginx работает на unix-подобных операционных системах, таких как: FreeBSD, OpenBSD, Linux, Solaris, Mac OS X, AIX, HP-UX, но также есть и версии под Windows, хотя как Вы уже поняли, разрабатывался этот web сервер именно под unix. Последней версией на данный момент является версия 1.2.4.

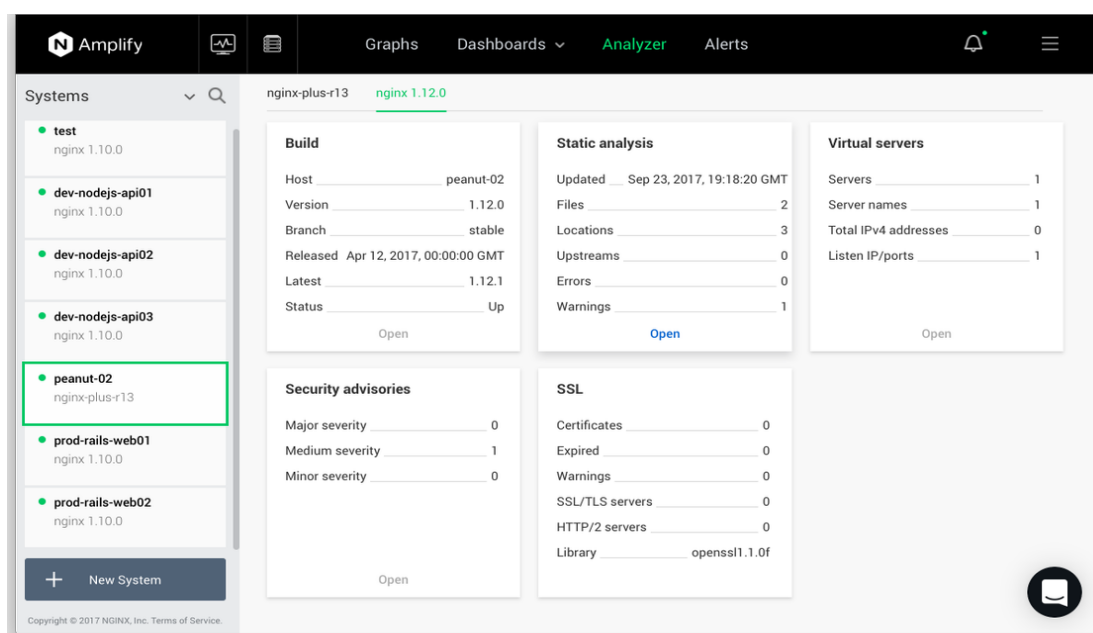


Рисунок 1.1 — Результат работы веб-сервера «Nginx»

Практическое применение:

- Отдельный порт/IP. При наличии большого количества статического контента или файлов для загрузки, можно настроить на отдельном порту или IP,

чтобы осуществлять раздачу. При большом количестве запросов рекомендуется ставить отдельный сервер и подключать к нему Nginx.

- Акселерированное проксирование. В таком случае все пользовательские запросы на статичный контент (картинки, простой HTML, JavaScript, CSS-файлы) поступают сначала на Nginx, а он их обрабатывает самостоятельно. При этом никаких изменений исходного кода не требуется.

- Nginx и FastCGI. Если поддерживается технология FastCGI, Apache вообще можно не использовать. Но в таком случае может потребоваться модификация кодов скриптов.

Сервис Nginx имеет свои преимущества и недостатки.

Недостатки:

- нет встроенной поддержки технологии Web-сокеты;
- встроенная поддержка PHP. Лечится подключением PHP через fastcgi;
- существуют пиратские копии, имеющие в себе вредоносный код.

Преимущества:

- шифрование, сжатие, поддержка многосайтовости на IP-адресе;
- межсистемность, малый размер, простота конфигурации, масштабируемость;
- преимущества асинхронной системы ввода-вывода. Это экономия ресурсов системы при больших нагрузках.

### **1.2.2 Веб-сервер «Apache».**

Данный сервис находится в открытом доступе и скачать его можно на сайте [apache.org](http://apache.org). Apache — это бесплатный, кроссплатформенный web сервер, он поддерживает следующие операционные системы: Microsoft Windows, Linux, BSD, Mac OS, Novell NetWare, BeOS[5].

Данный веб сервер отличается своей гибкостью в конфигурирование и расширяемостью, т.е. к нему можно подключать внешние модули. На данном web сервере можно разрабатывать сайты на таких языках программирования как: PHP, Python, Ruby, Perl, ASP, Tcl.

Веб-сервер на локальном компьютере может абсолютно всё, что умеют веб-сервера на хостингах. Веб-сервер подойдёт для разработки и тестирования сайтов и веб-приложений использующих, например, AJAX. Именно веб-сервер позволит вам работать со структурой веб-сайта так, будто бы он уже размещён на хостинге.

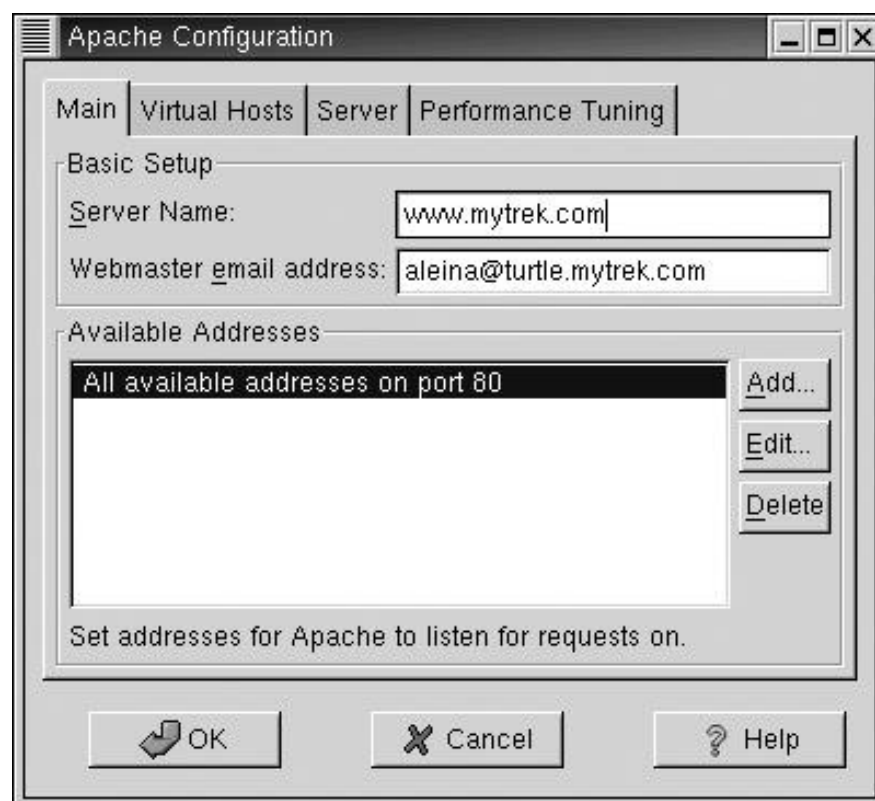


Рисунок 1.2 — Результат работы сервиса «Apache»

Недостатки:

- работает медленнее чем свои аналоги;
- медленно в отображении статического контента.

Преимущества:

- простота разрабатывать и обновлять приложения на Apache очень просто;
- обрабатывание динамического контента;
- преимущества асинхронной системы ввода-вывода;
- доступен на всех операционных системах.

### 1.2.3 Веб-сервер «Cherokee».

Особенность данного веб-сервера заключается в том, что управлять им можно только через веб-интерфейс.

Cherokee — свободный кроссплатформенный веб-сервер, написан на Си. Поддерживает все современные технологии, включая FastCGI, SCGI, PHP, CGI, SSI, HTTPS (TLS и SSL), виртуальные хосты, балансировку нагрузки и другие. Расширяем, благодаря поддержке плагинов. Основной упор при разработке делается на высокую производительность и скорость работы. Наличие веб-интерфейса позволяет упростить настройку. В ряде тестов показывает более высокую скорость работы, в сравнении с такими серверами, как Lighttpd и Nginx[7].

Основное средство конфигурирования Cherokee — программа `cherokee-admin`, открывающая на время запуска административный веб-интерфейс. Для входа в него используются имя пользователя и временный пароль, которые выводит `cherokee-admin`.

Благодаря этому интерфейсу конфигурирования Cherokee отлично подходит для быстрого развёртывания веб-сервера, например для тестирования веб-приложений.

Конфигурирование из командной строки также поддерживается, для этого используется программа `cherokee-tweak`. Либо можно вручную редактировать конфигурационный файл.

### **1.3 Выбор технологий для создания проекта**

Для разработки данного приложения использовалась Visual Studio Code. Данная интегрированная среда была выбрана для разработки курсового проекта по ряду причин. Немаловажно также наличие официальной документации с описанием наиболее важных библиотек, описан каждый метод для работы с подробными примерами.

Реализация протокола HTTP будет проходить за счет стандартных библиотек, имеющихся в языке C, такие как `sys/socket.h`, `arpa/inet.h` и `pthread.h`.

### **1.4 Описание основных функций создаваемого программного обеспечения**

В данном проекте реализованы функции:

- обработки GET и POST запросов;
- получения ответов от сервера;
- многопоточного подключения сокетов;
- работы с базой данных сервера.

## **2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ**

### **2.1 Общее структурное описание состава программного обеспечения**

Для организации корректного функционала HTTP-сервера необходимо разделить работу над ним на следующие модули.

### **2.2 Краткое описание модулей**

Работа программы разбита на 4 основных структурных элемента: блок для работы с базой данных, блок для работы с сервером, блок отладочной информации и блок работы с сокетами.

#### **2.2.1 Блок для работы с базой данных**

Блок для работы с базой данных – часть программы, хранящая находящиеся на сервере данные и отвечающая за выбор файлов HTML формата, загружаемых и обрабатываемых в зависимости от полученного запроса.

#### **2.2.2 Блок для работы с сервером**

Блок для работы с сервером – модуль, отвечающий за создание платформы работы сервера и реализацию его работы на сокетах.

#### **2.2.3 Блок отладочной информации**

Блок отладочной информации – часть программы, отвечающая за вывод вспомогательной информации ответа сервера на экран. Как пример: получение запроса, отправленного клиентом.

#### **2.2.4 Блок для работы с сокетами и сетью**

Блок для работы с сокетами и сетью – модуль, отвечающий за инициализацию сокета прослушивания и работу с сокетами-клиентами.

## 3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

В данном разделе описывается функционирование и структура разрабатываемой программы.

### 3.1 Описание основных функций программы

- `int create_socket` – создает неблокирующий сокет по заданному порту со стандартными протоколами для HTTP и устанавливает максимальное число подключений.;

- `void *get_client_addr` – данная функция получает IPv4 или IPv6 - адрес клиента.;

- `void *recv_HttpRequest` – функция, которую выполняет каждый отдельный поток, когда клиент соединяется с сервером. Получает строку запроса, логирует время обработки полученных данных и контролирует количество потоков в данный момент времени;

- `void parse_http_request` – извлекает основную информацию из запроса для формирования ответа;

- `void send_HttpResponse` – обрабатывает извлеченные данные из запроса и на их основании формирует ответ, производит различные проверки;

- `int send_file` – принимает имя файла для отправки и сокет клиента, после чего отправляет считанные данные из файла, а если файл большой, отправляет их порционно;

- `void send_404` – отправляет самую распространенную ошибку о неправильном запросе клиенту;

- `void directory_listing` – считывает все файлы и каталоги хранящиеся на сервере и отправляет в корректной и удобочитаемой форме клиенту (как веб страницу).

## **4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ**

### **4.1 Алгоритм чтения файлов и каталогов**

В данном разделе приведен алгоритм чтения всех вложенных файлов и каталогов, принадлежащих каталогу отправленному в запросе (является частью блока работы с базой данных).

Как только на сервер поступает GET запрос и URL адресуется на объект, не имеющий файловое расширение, алгоритм распознает это как команду отображения каталога. Происходит поиск всех вложенных объектов и затем отображение.

Ниже представлено более подробное описание алгоритма:

Шаг1. В функцию передаем путь каталога.

Шаг2. Блокируем доступ к каталогу другим потокам используя семафор, чтобы избежать эффекта гонок

Шаг3. Ищем в каталоге все вложенные файлы и подкаталоги используя структуру, куда помещается информация о каждом найденном объекте.

Шаг4. Сортируем отдельно каталоги и файлы в алфавитном порядке

Шаг5. Формируем html-строку ответа из отсортированных данных

Шаг6. Отправляем обработанные данные на сокет.

Шаг7. Закрываем дескриптор поиска файлов, открываем доступ остальным потокам к каталогу и очищаем выделенную память.

### **4.2 Алгоритм извлечения основной информации из запроса**

Ниже приведен алгоритм получения метода запроса, URL адреса, переданных аргументов, версии HTTP протокола, расширения запрашиваемого файла (является частью блока отладочной информации):

Шаг1. Получаем запрос клиента, содержащий необходимую информацию.

Шаг2. Далее следует найти и скопировать с полученной строки данные в отдельный буфер.

Шаг3. Инициализируем указатели начала нужных данных, в ячейки которых будут заноситься подстроки копии запроса.

Шаг4. Считываем строки с копии запроса с помощью функции strtok, которой в качестве аргумента передаем символы согласно стандарту RFC.

Шаг5. Записываем пустую строку в случае возврата функции нулевого указателя.

Шаг6. Выводим полученные данные.

### 4.3 Алгоритм отправки ответа

Ниже приведен алгоритм формирования заголовков ответа, передача управления функциям из блока работы с базой данных и отладочной информации (является частью блока работы с сокетами и сетью):

Шаг1. Получаем запрос клиента и его сокет.

Шаг2. Извлекаем версию протокола по которой был отправлен запрос.

Шаг3. Производим проверку на корректную версию HTTP протокола и в случае не совпадения с ни одной из поддерживаемых, отправляем 404 ошибку.

Шаг4. Извлекаем метод запроса.

Шаг5. Если метод запроса GET, отправляем запрошенный файл или выводим каталог с соответствующими заголовками для каждого типа данных.

Шаг6. Если метод запроса POST, ищем токен с типом action и отправляем извлеченные данные обратно клиенту.

Шаг7. Иначе выводим 404 ошибку.



## 5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

### 5.1 Минимальные программно-аппаратные требования к установке и запуску программы

Для корректной работы разработанного программного средства необходим ПК с установленной на нем операционной системы Linux. Соединение с интернетом для работы с программой так же требуется.

### 5.2 Краткое описание основных действий пользователя при использовании программой

Разработанное программное средство легко в использовании. Ниже будут даны описания последовательностей действий, необходимых для успешной работы с приложением.

После запуска программы откроется консоль, в которой будет указан адрес, по которому можно отправлять запросы. Адрес это ссылка, поэтому ее можно открыть с помощью любого из установленных браузеров предварительно перед нажатием зажав CTRL.

```
user@user-IdeaPad:~/BSUIR/0СиСП/Бейнап A.B./course_project$ ./server  
Server started http://127.0.0.1:8080
```

В браузере в строке запроса также можно написать сообщение формата `http://localhost:<порт>/<название файла или каталога>`. При запросе без указания файла (`http://localhost:8080`), сервер будет пытаться открыть и отобразить главную страницу – `index.html` (рисунок 5.1).



Рисунок 5.1 – Открываем главную страницу

Как только вы перешли на HTML страницу сервер получает запрос, который отображается в консоли. Также происходит логирование времени, когда поступил запрос, и затраченное каждым потоком время на обработку. Любой переход с одной страницы на другую сопровождается GET запросом. Увидеть это можно в данном выводе:

```
+ [GET] / HTTP/1.1  
send file: ./index.html  
Request processing time = 0.000263 seconds
```

```
Server: got connection from 127.0.0.1
Server: got connection from 127.0.0.1
Tue May 16 23:05:55 2023
Tue May 16 23:05:55 2023
Server: got connection from 127.0.0.1
+ [GET] /css/aos.css HTTP/1.1 HTTP/1.1
Tue May 16 23:05:55 2023
send file: ./css/aos.css
Request processing time = 0.000716 seconds
+ [GET] /css/bootstrap.min.css HTTP/1.1 HTTP/1.1
send file: ./css/bootstrap.min.css
Request processing time = 0.000927 seconds
+ [GET] /css/main.css HTTP/1.1 HTTP/1.1
send file: ./css/main.css
Request processing time = 0.000980 seconds
Server: got connection from 127.0.0.1
Tue May 16 23:05:55 2023
```

Далее приведён пример основного HTML файла, который запускался для демонстрации работы программы (рисунок 5.2 и 5.3).

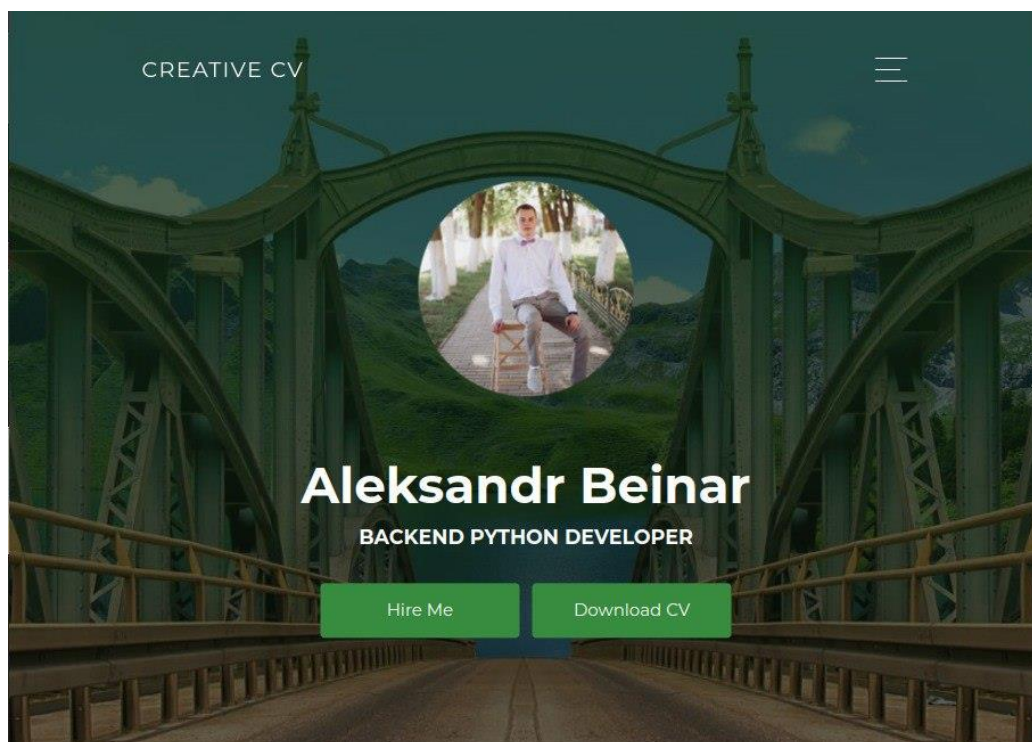


Рисунок 5.2 – Открытый HTML файл

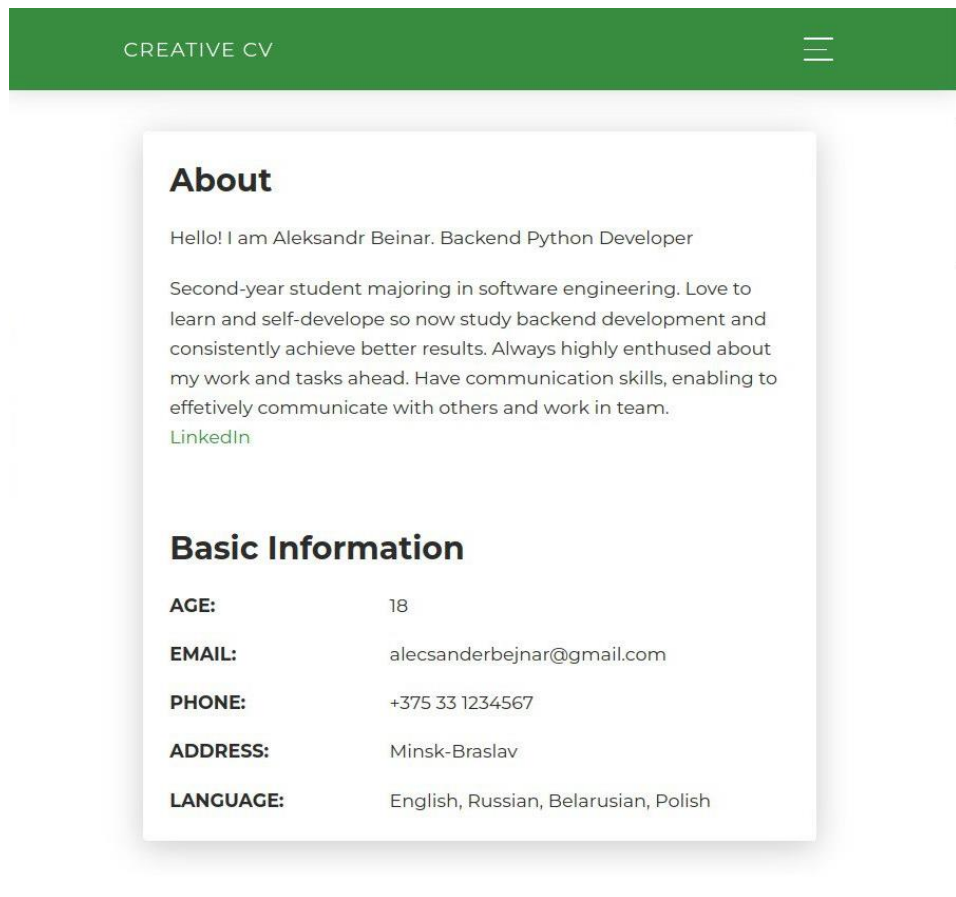


Рисунок 5.3 – Открытый HTML файл

Также был добавлен дополнительный эндпоинт `/home`, при запросе на который отображаются каталоги и файлы, содержащиеся в текущем каталоге, из которого запускался сервер (рисунок 5.4).

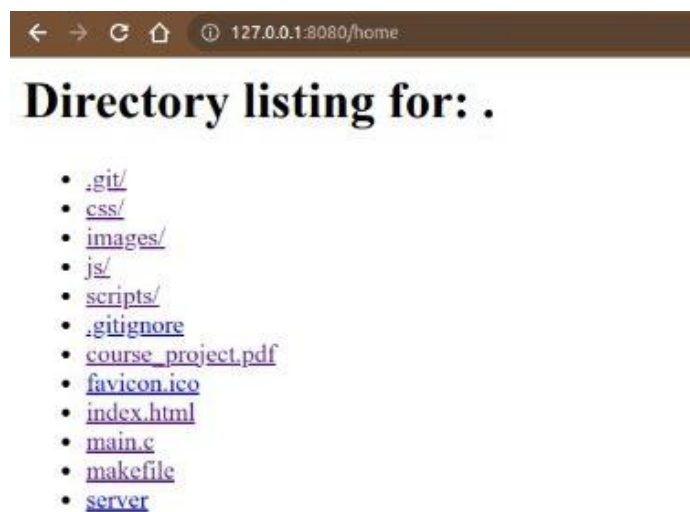


Рисунок 5.4 – Открытая HTML страница по эндпоинту `/home`

На этой странице есть возможность перейти в любой каталог, на который указывает ссылка заканчивающаяся символом `</>`. Это продемонстрировано на рисунке 5.5.

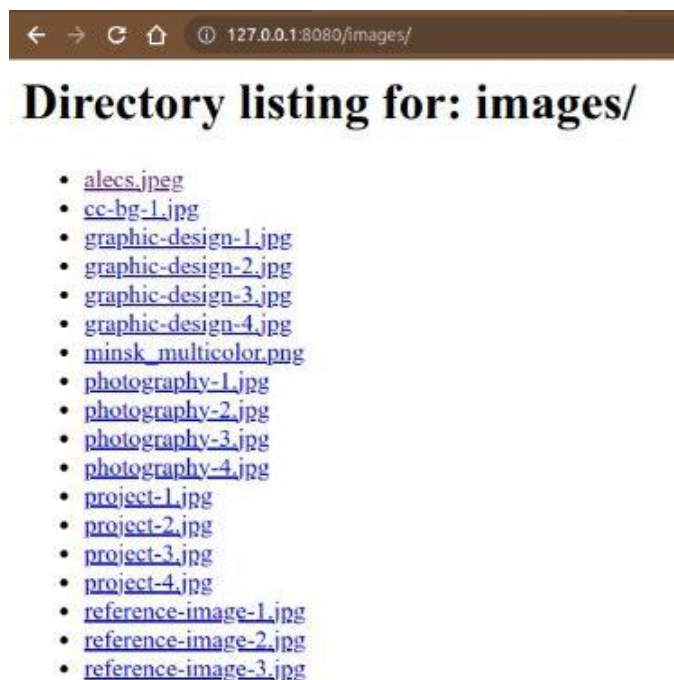


Рисунок 5.5 – Переход из /home в подкаталог

Если же на домашней странице нажать на любой файл, то на веб-странице будет отображен текст внутри этого файла. Представлено на рисунке 5.6.

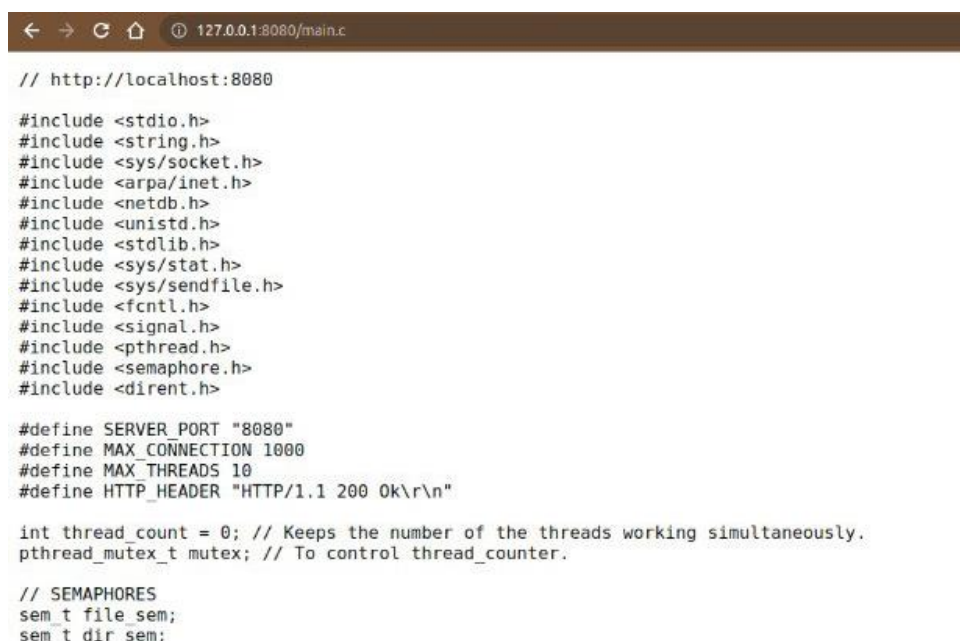


Рисунок 5.6 – Отображение файлов

На главной странице имеется возможность отправки POST запроса на сервер. Для этого необходимо заполнить форму приведенную на рисунке 5.7.

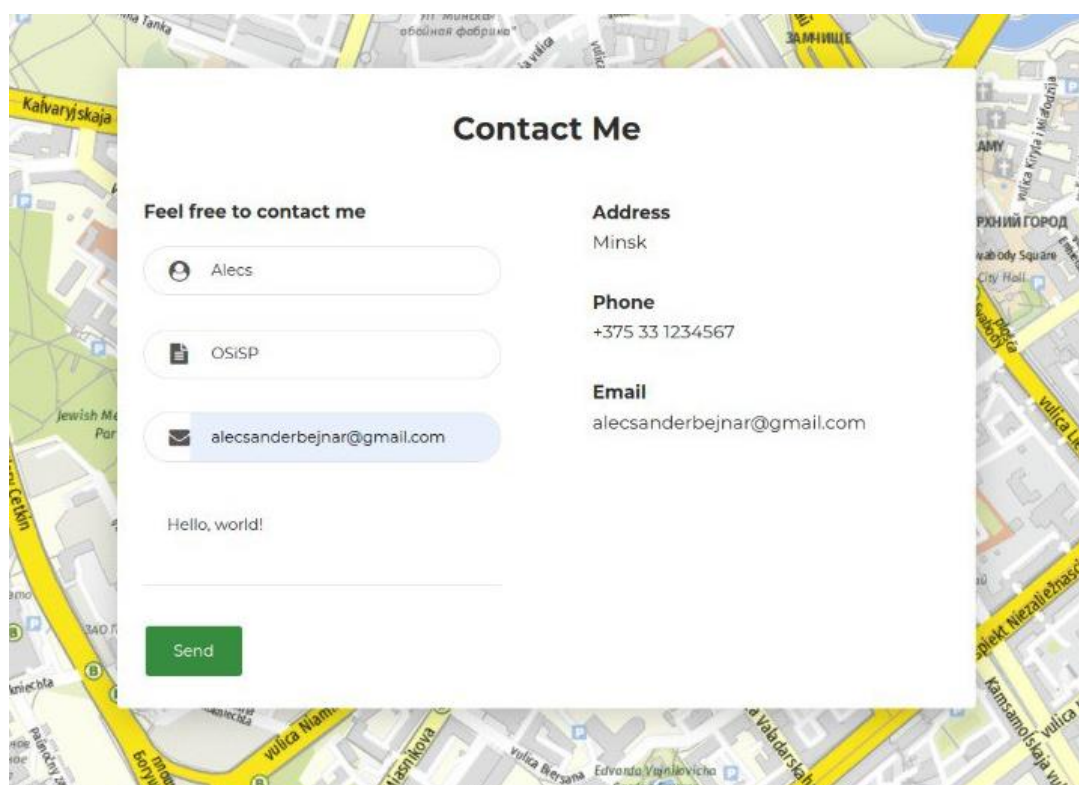


Рисунок 5.7 – Заполнение формы для отправки POST запроса

После нажатия кнопки SEND, на сервер придут данные, которые заполнял пользователь, в том числе и токены. Это можно увидеть в следующем выводе:

```
Server: got connection from 127.0.0.1
+ [POST] /action HTTP/1.1
--Token: POST /action HTTP/1.1
find string: POST /action HTTP/1.1
Request processing time = 0.000058 seconds
Mon May 1 02:41:08 2023
```

В браузере же отобразится извлеченный из запроса токен UserAction (рисунок 5.8).



Рисунок 5.8 – Вывод в браузере после отправки POST запроса

### 5.3 Тестирование и проверка работоспособности программного средства

Исключительные ситуации – ситуации, возникающие во время работы системы, выходящие за пределы, предусмотренные спецификацией. Для того, чтобы предотвратить такие случаи необходимо в коде программы предусмотреть защиту от неправильных действий.

В первую очередь стоит уделить внимание на отправку запроса неправильного формата, отсутствие основного HTML-файла index.html в том же каталоге откуда запускается сервер, ситуацию когда пришло одновременно большое количество запросов и все доступные потоки заняты, клиент неожиданно отсоединился. Такие моменты стоит обязательно обрабатывать для получения программы устойчивой к неправильным действиям.

На рисунке 5.9 представлена обработка ситуации получения запроса неправильного формата.

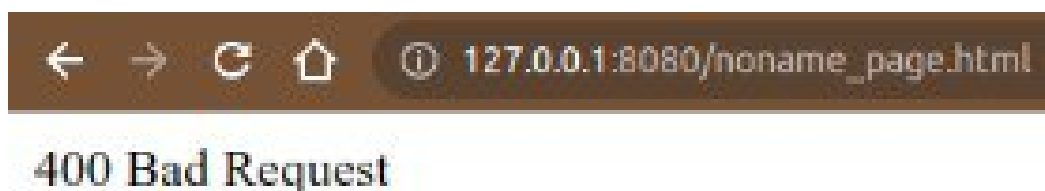


Рисунок 5.9 – Обработка некорректного запроса

При этом сервер в консоль выведет следующее:

```
Server: got connection from 127.0.0.1  
Mon May 1 02:29:25 2023  
+ [GET] /noname_page.html HTTP/1.1  
Cannot Open file path : ./noname_page.html with error -1  
Request processing time = 0.000344 seconds
```

На рисунке 5.10 представлена обработка ситуации отсутствия основного HTML-файла index.html в том же каталоге откуда запускается сервер.



Рисунок 5.10 – Отсутствие файла index.html



При этом сервер в консоль выведет следующее:

```
Server started http://127.0.0.1:8080
Server: got connection from 127.0.0.1
Mon May 1 02:29:25 2023
+ [GET] / HTTP/1.1
Cannot Open file path : ./index.html with error -1
Request processing time = 0.000344 seconds
Add index.html file!
```

Обработка ситуации, когда пришло одновременно большое количество запросов и все доступные потоки заняты, приводит к выводу в консоли следующего сообщения и при этом запрос не обрабатывается:

```
Server: got connection from 127.0.0.1
Mon May 1 02:26:47 2023
System is busy right now!
```

В данном выводе представлена обработка ситуации неожиданного рассоединения с клиентом:

```
+ [GET] /some.html HTTP/1.1
Cannot Open file path : ./some.html with error -1
Request processing time = 0.000125 seconds
Client desconnected unexpectedly.
```

## **ЗАКЛЮЧЕНИЕ**

В процессе выполнения курсового проекта были выполнены все поставленные задачи:

- создание простого прототипа многопоточного HTTP-сервера.
- способность обрабатывать основные запросы, передаваемые на сервер, получение ответа от сервера.
- освоение навыков работы с файловой системой, заложенной на сервере.

Результатом курсового проекта стало создание приложения многопоточного HTTP-сервера, выполняющего все задуманные функции. Следует отметить, что данная программа может быть усовершенствована путем добавления сохранения ответов сервера в файл, добавления обработок не только двух видов запросов, но и большего числа.

В дальнейшем планируется усовершенствование программы, а именно, усовершенствование графического интерфейса и расширения функционала, добавление разных виджетов с расширенным набором действий.



## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1]Страуструп, Б. Программирование. Принципы и практика использования С/ Б. Страуструп, Вильямс, 2018 г. – 991 с.

[2]Стивен П. Язык программирования С. Лекции и упражнения, 6-е изд. : Пер. с англ. – М. : И.Д. Вильямс, 2012. – 1248 с.

[3]Amplify [Электронный ресурс]. – Режим доступа: <https://amplify.nginx.com> – Дата доступа: 11.05.2023.

[4]Apache [Электронный ресурс]. – Режим доступа: <https://apache-windows.ru> – Дата доступа: 11.05.2023.

[5]Cherokee [Электронный ресурс]. – Режим доступа: <http://cherokee-project.com> – Дата доступа: 11.05.2023.

**ПРИЛОЖЕНИЕ А**  
(обязательное)

Ведомость документов

## **ПРИЛОЖЕНИЕ Б**

*(обязательное)*

Схема программы

## **ПРИЛОЖЕНИЕ В**

*(обязательное)*

Алгоритм функции «recv\_HttpRequest»

**ПРИЛОЖЕНИЕ Г**  
(обязательное)

Листинг кода