ЛАБОРАТОРНАЯ РАБОТА №4

Введение

Операционная система должна предоставлять разработчику интерфейс для работы с сетью, позволяющий отправлять и получать произвольные данные. В среде Linux таким интерфейсом являются *сокеты*. Сокет — это абстракция конечной точки сетевого соединения, через которую приложение может читать и писать, так же, как в файл.

Для создания и использования сокетов в языке C применяется семейство функций:

- `int socket(int domain, int type, int protocol)` — создаёт сокет.
- `int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen)` — привязывает сокет к адресу.
- `int listen(int sockfd, int backlog)` — переводит сокет в режим ожидания входящих соединений.
- `int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen)` — принимает входящее соединение.
- `ssize_t recv(int sockfd, void *buf, size_t len, int flags)` — чтение
- `ssize_t send(int sockfd, const void *buf, size_t len, int flags)` — запись.

Ниже приведён минимальный пример создания TCP-сервера, принимающего одно соединение:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>

int main() {
    int server_fd = socket(AF_INET, SOCK_STREAM, 0);
    if (server_fd < 0) perror("socket"), exit(1);

    struct sockaddr_in addr;
    memset(&addr, 0, sizeof(addr));
    addr.sin_family = AF_INET;
    addr.sin_addr.s_addr = INADDR_ANY;
    addr.sin_port = htons(80);

    if (bind(server_fd, (struct sockaddr*)&addr, sizeof(addr)) < 0)
        perror("bind"), exit(1);

    if (listen(server_fd, 5) < 0)
        perror("listen"), exit(1);

    int client_fd = accept(server_fd, NULL, NULL);
    if (client_fd < 0) perror("accept"), exit(1);
```

```
    char buffer[1024];
    ssize_t n = recv(client_fd, buffer, sizeof(buffer)-1, 0);
    if (n > 0) {
        buffer[n] = '\0';
        printf("Received:\n%s", buffer);
        const char *msg = "Hello from server!\n";
        send(client_fd, msg, strlen(msg), 0);
    }

    close(client_fd);
    close(server_fd);
    return 0;
}
```

Поверх такого интерфейса уже можно строить протоколы уровня приложений. Самый популярный из таких протоколов - протокол HTTP, поверх которого работает подавляющее большинство современных веб-приложений. HTTP (HyperText Transfer Protocol) — протокол передачи гипертекстовых документов. Сообщения делятся на 2 категории: запросы (request) и ответы(response).

Запрос состоит из четырёх частей:

1. **Строка запроса (Request Line)**:
   `<Method> <Request-URI> <HTTP-Version>\r\n`

   Где:
   - `Method` — поддерживаемый метод (`GET`, `HEAD`, `POST` и др.). Для нашего сервера достаточно `GET`.
   - `Request-URI` — путь к ресурсу, например `/index.html`. Может содержать
   - `HTTP-Version` — версия протокола, например `HTTP/1.1`.
2. **Заголовки запроса (Headers)**: набор полей в формате `Field-Name: Field-Value`, каждое заканчивается `\r\n`. Обязательные и часто используемые:
   - `Host:` — имя хоста и, опционально, порт (требуется в HTTP/1.1).
   - `User-Agent:` — строка идентификации клиента.
   - `Accept:` — список типов медиа, которые клиент готов принять.
   - `Connection:` — опции управления соединением (`keep-alive` или `close`).
3. **Пустая строка**: последовательность `\r\n`, отделяющая заголовки от тела.
4. **Тело сообщения (Message Body)**: присутствует только для некоторых методов (например, `POST`). Для `GET` тело отсутствует.

Пример HTTP-запроса:

```
GET /dir/file.txt HTTP/1.1
```

```
Host: localhost

User-Agent: curl/7.68.0

Accept: */*

Connection: close
```

Ответ сервера также состоит из четырёх частей:

1. **Строка статуса (Status Line)**:
   `<HTTP-Version> <Status-Code> <Reason-Phrase>\r\n`

   Где:

   - `HTTP-Version` — версия протокола, например `HTTP/1.1`.
   - `Status-Code` — трехзначный код состояния (200, 404, 500 и др.).
   - `Reason-Phrase` — текстовое описание кода (например, "OK" или "Not Found").
2. **Заголовки ответа (Headers)**: поля `Field-Name: Field-Value`, каждое `\r\n`:
   - `Content-Length:` — длина тела в байтах.
   - `Content-Type:` — MIME-тип содержимого (например, `text/plain`, `text/html`).
   - `Connection:` — указывает, будет ли соединение закрыто после ответа.
3. **Пустая строка**: `\r\n` после заголовков.
4. **Тело ответа (Message Body)**: содержимое ресурса.

Пример HTTP-ответа (успешный):

```
HTTP/1.1 200 OK

Content-Length: 1234

Content-Type: text/plain

Connection: close


< content here >
```

Пример HTTP-ответа (404 Not Found):

```
HTTP/1.1 404 Not Found

Content-Length: 0

Connection: close
```

Задание

Необходимо реализовать простейший http-сервер на языке C.

Сервер должен поддерживать метод `GET` и раздачу статических файлов из рабочей директории.

Рассмотрим пример. В рабочей директории сервера есть структура:

```
./wwwroot/hello.txt
cat hello.txt
Hello, HTTP!
```

Запустим сервер на порту 8080:

```
$ ./http_server 8080
Server started, listening on port 8080
```

Выполним запрос:

```
$ curl http://localhost:8080/wwwroot/hello.txt
Hello, HTTP!
```

Если файла нет:

```
$ curl http://localhost:8080/wwwroot/missing.txt
404 Not Found
```

Отчет о работе должен содержать код и описание реализации.