Лабораторная работа №4, «Создание HTTP-сервера»
Группа: Б23-534
ФИО: Калашников Владимир Алексеевич
Номер в журнале: 5
Год в Москве: 2025

Описание лабораторной работы

Ссылка на GitHub

Работа выполнена на языке Си и представляет из себя файл *main.c*, в котором прописана работа HTTP-сервера.

Также проект оснащён небольшим скриптиком *Build.sh* для сборки с помощью CMake и последующего запуска оболочки. И favicon.ico, если кто-то решит обратиться к серверу с помощью браузера.

Функция main

```
int main(int argc, char* argv[]) {
  struct sigaction sa;
  sa.sa_handler = handle_signal;
  sigemptyset(&sa.sa_mask);
  sa.sa_flags = 0;
  sigaction(SIGINT, &sa, NULL);
  sigaction(SIGTERM, &sa, NULL);


  init_server(argc, argv);
  init_server_dir();

  char buffer[BUFFER_SIZE];

  while (KEEP_RUNNING == 1) {
    int client_fd = accept(SERVER_FD, NULL, NULL);
    if (client_fd < 0) {
      if (KEEP_RUNNING != 1) {
        break;
      }
      perror("accept");
      continue;
    }

    ssize_t n = recv(client_fd, buffer, sizeof(buffer)-1, 0);
    if (n > 0) {
      buffer[n] = '\0';
      printf("\n\nReceived:\n%s\n", buffer);
      if (strncmp(buffer, "GET /", 5) == 0) {
        char *path_start = buffer + 5;
        char *path_end = strchr(path_start, ' ');
        if (path_end && path_start != path_end) {
          *path_end = '\0';
          char requested_path[PATH_SIZE];
          strcpy(requested_path, SERVER_DIR);
          strcat(requested_path, path_start);
          printf("Requested file: '%s'\n", path_start);
          printf("Requested path: '%s'\n", requested_path);
          send_file(client_fd, requested_path);
        }
        else {
          send_response(client_fd, "404 Not Found", "text/plain", "404 Not Found\n");
        }
      }
      else {
        send_response(client_fd, "400 Bad Request", "text/plain", "Only GET method is supported\n");
      }
    }
    close(client_fd);
```

```
  }
  close(SERVER_FD);
  return 0;
}
```

Первый блок отвечает за переопределение сигналов killpid и ctrl+c. При их вызове системный вызов accept() будет прерван, переменная KEEP_RUNNING приравняется к нулю функцией handle_signal, а программа завершится.

Следующими идут инициализации сервера и рабочей папки.

Затем цикл, принимающий запрос и перенаправляющий его в функции обработки.

Остальные функции слишком велики,

..чтобы их сюда вставить. Увидеть их можно по Ссылке на GitHub.

```
void handle_signal(int sig);
```

Вызывается при срабатывании вышеописанных сигналов. Ставит переменную KEEP_RUNNING в единицу.

```
void init_server(int argc, char* argv[]);
```

Проверяет argv и argc, создаёт сокет и привязывает его к порту из argv.

```
void init_server_dir();
```

Вычисляет директорию, из которой запустился процесс.

```
void send_response(int client_fd, const char *status, const char *content_type,
const char *body);
```

Отправляет ответик в формате HTTP 1.1

```
void send_file(int client_fd, const char * requested_path);
```

Проверяет, существует ли файл, можно ли к нему обеспечить доступ, отправляет его.