Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика”

Кафедра №806 “Вычислительная математика и программирование”

Лабораторная работа №1 по курсу

«Операционные системы»

Группа: М8О-210БВ-24

Студент: Дмитренко Я.С.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 01.10.25

Москва, 2025

Постановка задачи

Вариант 4.

Пользователь вводит команды вида: «число число число<endline>». Далее эти числа передаются от родительского процесса в дочерний. Дочерний процесс производит деление первого числа, на последующие, а результат выводит в файл. Если происходит деление на 0, то тогда дочерний и родительский процесс завершают свою работу. Проверка деления на 0 должна осуществляться на стороне дочернего процесса. Числа имеют тип float. Количество чисел может быть произвольным.

Общий метод и алгоритм решения

Использованные системные вызовы:

- pid_t fork(void); – создает дочерний процесс.
- int pipe(int *fd); – создает анонимный канал, возвращает 0 при успехе, -1 при ошибке
- ssize_t read(int fd, void *buf, size_t count) - читает данные из файла по дескриптору fd в буфер buf. Возвращает количество прочитанных байт
- ssize_t write(int fd, const void *buf, size_t count) - записывает данные в файл по дескриптору fd. Возвращает число записанных байт
- int open(const char *pathname, int flag, mode_t mode) - Открывает файл, возвращает файловый дескриптор
- int close(int fd) - закрываает файловый дескриптор
- int execl(const char *path, const char *arg0, ..., NULL) – заменяет текущий процесс новым исполняемым файлом

- int dup2(int fd, int fd2) - замена файлового дескриптора
- pid_t wait(int *wstatus) - ожидает завершения дочернего процесса. Возвращает pid завершившегося процесса

Клиент и сервер взаимодействуют через два анонимных канала (pipe). Клиентский процесс создаёт дочерний процесс, в котором запускает серверную программу, предварительно перенаправив стандартные потоки ввода и вывода на созданные пайпы. Из консоли клиент считывает строки, содержащие числа, и передаёт их серверу через первый канал. Сервер принимает строку чисел из своего стандартного ввода (который подключён к первому каналу), разбивает её на отдельные числовые значения и выполняет операцию деления первого числа на каждое из последующих. Все результаты операций, а также служебные сообщения сервер записывает в указанный выходной файл. В случае обнаружения деления на ноль сервер записывает соответствующую ошибку в файл, отправляет клиенту специальный сигнал через второй канал и аварийно завершает свою работу. Клиент, получив сигнал о делении на ноль, также прекращает выполнение, выводит сообщение об ошибке и корректно завершает работу. При нормальном выполнении после каждой успешной обработки сервер отправляет клиенту подтверждение через второй канал, а клиент продолжает ожидание новых вводов пользователя до команды выхода или разрыва соединения.

Код программы

client.c

```c
#include <stdio.h>
#include <stdlib.h>
```

```c
#include <unistd.h>
#include <string.h>
#include <sys/wait.h>
#include <signal.h>

int main() {
    int pipe1[2], pipe2[2];
    pid_t pid;
    char filename[100];
    char user_input[1024];
    char response_buffer[256];

    if (pipe(pipe1) == -1 || pipe(pipe2) == -1) {
        perror("Ошибка создания pipe");
        exit(EXIT_FAILURE);
    }

    printf("Введите имя файла для результатов: ");
    if (fgets(filename, sizeof(filename), stdin) == NULL) {
        perror("Ошибка чтения имени файла");
        exit(EXIT_FAILURE);
    }
    filename[strcspn(filename, "\n")] = 0;

    pid = fork();
    if (pid == -1) {
        perror("Ошибка fork");
        exit(EXIT_FAILURE);
    }

    if (pid == 0) {
        // Дочерний процесс
        close(pipe1[1]);
        close(pipe2[0]);
        dup2(pipe1[0], STDIN_FILENO);
        dup2(pipe2[1], STDOUT_FILENO);
        close(pipe1[0]);
        close(pipe2[1]);

        execl("./server", "server", filename, NULL);
        perror("Ошибка execl");
        exit(EXIT_FAILURE);

    } else {
        // Родительский процесс
        close(pipe1[0]);
        close(pipe2[1]);

        printf("Дочерний процесс создан.\n\n");

        while (1) {
            printf("Введите числа: ");
            if (fgets(user_input, sizeof(user_input), stdin) == NULL) {
                break;
```

```
            }

            if (strncmp(user_input, "exit", 4) == 0) {
                break;
            }

            // Отправляем данные
            write(pipe1[1], user_input, strlen(user_input));

            // Ждем ответ от сервера
            ssize_t bytes_read = read(pipe2[0], response_buffer, sizeof(response_buffer) -
1);
            if (bytes_read > 0) {
                response_buffer[bytes_read] = '\0';

                if (strstr(response_buffer, "DIVISION_BY_ZERO") != NULL) {
                    printf("Обнаружено деление на ноль! Завершаем работу.\n");
                    break;
                } else {
                    printf("Сервер: %s", response_buffer);
                }
            } else if (bytes_read == 0) {
                printf("Сервер закрыл соединение\n");
                break;
            }
        }

        close(pipe1[1]);
        close(pipe2[0]);
        wait(NULL);
        printf("Родительский процесс завершен.\n");
    }

    return 0;
}
```

server.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <math.h>
#include <fcntl.h>

void write_to_file(int file, const char* message) {
    size_t len = strlen(message);
    if (write(file, message, len) == -1) {
        perror("Ошибка записи в файл");
        exit(EXIT_FAILURE);
    }
}
```

```c
void process_numbers(int file, const char* input) {
    char buffer[4096];
    strncpy(buffer, input, sizeof(buffer) - 1);
    buffer[sizeof(buffer) - 1] = '\0';

    float numbers[100];
    int count = 0;

    // Разбираем строку на числа
    char* token = strtok(buffer, " ");
    while (token != NULL && count < 100) {
        numbers[count++] = atof(token);
        token = strtok(NULL, " ");
    }

    if (count < 2) {
        write_to_file(file, "Ошибка: нужно минимум 2 числа\n");
        return;
    }

    char result_str[256];
    float first = numbers[0];

    // Формируем строку для файла
    snprintf(result_str, sizeof(result_str), "Делим %.2f на:", first);
    write_to_file(file, result_str);
    write_to_file(file, "\n");

    // Выполняем деление
    for (int i = 1; i < count; i++) {
        if (fabs(numbers[i]) < 1e-6) {
            snprintf(result_str, sizeof(result_str), "  ОШИБКА: деление на ноль! %.2f /
%.2f\n", first, numbers[i]);
            write_to_file(file, result_str);

            const char error_msg[] = "DIVISION_BY_ZERO\n";
            write(STDOUT_FILENO, error_msg, strlen(error_msg));
            exit(EXIT_FAILURE);
        }

        float result = first / numbers[i];
        snprintf(result_str, sizeof(result_str), "  %.2f / %.2f = %.2f\n", first,
numbers[i], result);
        write_to_file(file, result_str);
    }
    write_to_file(file, "---\n");
}

int main(int argc, char* argv[]) {
    if (argc != 2) {
        const char error_msg[] = "Использование: server <имя_файла>\n";
        write(STDERR_FILENO, error_msg, strlen(error_msg));
        exit(EXIT_FAILURE);
    }
```

```
    const char* filename = argv[1];

    int file = open(filename, O_WRONLY | O_CREAT | O_TRUNC, 0644);
    if (file == -1) {
        perror("Ошибка открытия файла");
        exit(EXIT_FAILURE);
    }

    write_to_file(file, "Сервер запущен. Ожидание чисел...\n");

    char input_buffer[4096];
    ssize_t bytes_read;

    // Читаем из STDIN (который перенаправлен из pipe1)
    while ((bytes_read = read(STDIN_FILENO, input_buffer, sizeof(input_buffer) - 1)) > 0)
{
        input_buffer[bytes_read] = '\0';


        if (input_buffer[bytes_read - 1] == '\n') {
            input_buffer[bytes_read - 1] = '\0';
        }

        if (strlen(input_buffer) == 0) continue;

        process_numbers(file, input_buffer);

        // Отправляем подтверждение обратно (эхо)
        const char response[] = "OK\n";
        write(STDOUT_FILENO, response, strlen(response));
    }

    write_to_file(file, "Сервер завершил работу\n");
    close(file);
    return 0;
}
```

Протокол работы программы

Входные данные:

```
Введите имя файла для результатов: result.txt
Дочерний процесс создан.

Введите числа: 1 2 3 4
Сервер: OK
Введите числа: 1 2 3
Сервер: OK
Введите числа: 1 0
Обнаружено деление на ноль! Завершаем работу.
Родительский процесс завершен.
```

Выходные данные:

result.txt

```
Сервер запущен. Ожидание чисел...
Делим 1.00 на:
  1.00 / 2.00 = 0.50
  1.00 / 3.00 = 0.33
  1.00 / 4.00 = 0.25
---
Делим 1.00 на:
  1.00 / 2.00 = 0.50
  1.00 / 3.00 = 0.33
---
Делим 1.00 на:
  ОШИБКА: деление на ноль! 1.00 / 0.00

```

Вывод

При выполнении данной лабораторной работы я научился работать с процессами в ОС. На практике увидел работу fork, pipe и других системных вызовов. Научился обрабатывать ввод, вывод как поток байтов, преобразовывать его в строки.