

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-210Б-23

Студент: Нугаев М. Э.

Преподаватель: Бахарев В.Д.

Оценка: \_\_\_\_\_

Дата: 30.11.24

Москва, 2024

# Постановка задачи

## Вариант 20:

Пользователь вводит строки». Далее эти строки передаются от родительского процесса в дочерний. Дочерний процесс конвертирует эти строки и затем записывает полученный результат в файл.

## Общий метод и алгоритм решения

Использованные системные вызовы:

- `pid_t fork(void)`; – создает дочерний процесс.
- `int pipe(int *fd)`; – создаёт пайп и помещает дескрипторы в `fd[0]`, `fd[1]`, для чтения и записи.
- `int write(int fd, const void* buff, int count)`; – записывает по дескриптору `fd` `count` байт из `buff`.
- `void exit(int number)`; – вызывает нормальное завершение программы с кодом `number`.
- `int dup2(int fd1, int fd2)`; – делает эквивалентными дескрипторы `fd1` и `fd2`.
- `int exec(char* path, const char* argc)`; – заменяет текущий процесс на процесс `path`, с аргументами `argc`;
- `int close(int fd)`; – закрывает дескриптор `fd`.
- `pid_t wait(int status)` — функция, которая приостанавливает выполнение текущего процесса до тех пор, пока дочерний процесс не завершится,

Я создал два файла `parent` и `child1` (`child2`).

Программа создает неименованный канал (`pipe`) для передачи данных между родительским и дочерним процессами. Она запрашивает у пользователя ввод имени файла, после чего создаёт дочерний процесс с помощью `fork()`. Дочерний процесс перенаправляет стандартный ввод на чтение из канала и запускает другую программу (дочернюю программу), передавая ей имя файла. Родительский процесс читает данные из стандартного ввода до тех пор, пока пользователь не остановит ввод и записывает эти данные в канал. После завершения записи родительский процесс закрывает канал и ожидает завершения дочернего процесса.

В файле `child` я обрабатываю полученные из родительского процесса данные и записываю их в файл.

## Код программы

### Parent.c

```
#include <unistd.h>
#include <string.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <fcntl.h>

#define BUFFER_SIZE 256

int main() {
    int pipe1[2], pipe2[2];
    char filename1[BUFFER_SIZE];
```

```

char filename2[BUFFER_SIZE];
ssize_t bytes_read;

const char *prompt1 = "Введите имя файла для child1: ";
write(STDOUT_FILENO, prompt1, strlen(prompt1));
bytes_read = read(STDIN_FILENO, filename1, BUFFER_SIZE - 1);
filename1[bytes_read - 1] = '\0';

const char *prompt2 = "Введите имя файла для child2: ";
write(STDOUT_FILENO, prompt2, strlen(prompt2));
bytes_read = read(STDIN_FILENO, filename2, BUFFER_SIZE - 1);
filename2[bytes_read - 1] = '\0';

int file_check1 = open(filename1, O_WRONLY | O_CREAT | O_TRUNC, 0666);
if (file_check1 == -1) {
    const char *error_msg = "Ошибка: не удалось открыть файл для child1\n";
    write(STDERR_FILENO, error_msg, strlen(error_msg));
    perror("open");
    return 1;
}
close(file_check1);

int file_check2 = open(filename2, O_WRONLY | O_CREAT | O_TRUNC, 0666);
if (file_check2 == -1) {
    const char *error_msg = "Ошибка: не удалось открыть файл для child2\n";
    write(STDERR_FILENO, error_msg, strlen(error_msg));
    perror("open");
    return 1;
}
close(file_check2);

if (pipe(pipe1) == -1 || pipe(pipe2) == -1) {
    const char *error_msg = "Ошибка при создании pipe\n";
    write(STDERR_FILENO, error_msg, strlen(error_msg));
    return 1;
}

pid_t pid1 = fork();
if (pid1 == -1) {
    const char *error_msg = "Ошибка при создании дочернего процесса 1\n";
    write(STDERR_FILENO, error_msg, strlen(error_msg));
    return 1;
} else if (pid1 == 0) {
    close(pipe1[1]);
    dup2(pipe1[0], STDIN_FILENO);
    execl("./child1", "./child1", filename1, NULL);
    const char *error_msg = "Ошибка при запуске child1\n";
    write(STDERR_FILENO, error_msg, strlen(error_msg));
    exit(1);
}

pid_t pid2 = fork();
if (pid2 == -1) {
    const char *error_msg = "Ошибка при создании дочернего процесса 2\n";

```

```

        write(STDERR_FILENO, error_msg, strlen(error_msg));
        return 1;
    } else if (pid2 == 0) {
        close(pipe2[1]);
        dup2(pipe2[0], STDIN_FILENO);
        execl("./child2", "./child2", filename2, NULL);
        const char *error_msg = "Ошибка при запуске child2\n";
        write(STDERR_FILENO, error_msg, strlen(error_msg));
        exit(1);
    }

    close(pipe1[0]);
    close(pipe2[0]);

    char input[BUFFER_SIZE];
    const char *prompt = "Введите строку: ";

    while (1) {
        write(STDOUT_FILENO, prompt, strlen(prompt));
        bytes_read = read(STDIN_FILENO, input, sizeof(input) - 1);
        if (bytes_read <= 0) {
            const char *error_msg = "Ошибка при чтении строки\n";
            write(STDERR_FILENO, error_msg, strlen(error_msg));
            break;
        }

        if (bytes_read > 0 && input[bytes_read - 1] == '\n') {
            input[bytes_read - 1] = '\0';
        }
        if (strlen(input) > 10) {
            write(pipe2[1], input, strlen(input) + 1);
        } else {
            write(pipe1[1], input, strlen(input) + 1);
        }
    }

    close(pipe1[1]);
    close(pipe2[1]);

    waitpid(pid1, NULL, 0);
    waitpid(pid2, NULL, 0);

    return 0;
}

```

### **Child1.c (Child2.c)**

```

#include <unistd.h>
#include <fcntl.h>
#include <string.h>
#include <stdlib.h>

```

```
#define BUFFER_SIZE 256

// Функция для инвертирования строки
void invert_string(char *str) {
    int len = strlen(str);
    for (int i = 0; i < len / 2; i++) {
        char temp = str[i];
        str[i] = str[len - i - 1];
        str[len - i - 1] = temp;
    }
}

int main(int argc, char *argv[]) {
    if (argc < 2) {
        const char *error_msg = "Ошибка: имя файла не передано\n";
        write(STDERR_FILENO, error_msg, strlen(error_msg));
        return 1;
    }

    int file = open(argv[1], O_WRONLY | O_CREAT | O_TRUNC, 0666);
    if (file == -1) {
        const char *error_msg = "Ошибка: не удалось открыть файл для записи\n";
        write(STDERR_FILENO, error_msg, strlen(error_msg));
        perror("open");
        return 1;
    }

    char buffer[BUFFER_SIZE];
    ssize_t bytes_read;

    while ((bytes_read = read(STDIN_FILENO, buffer, sizeof(buffer) - 1)) > 0) {
        buffer[bytes_read - 1] = '\0';

        invert_string(buffer);

        if (write(file, buffer, strlen(buffer)) == -1 || write(file, "\n", 1) == -1) {
            const char *error_msg = "Ошибка: не удалось записать в файл\n";
            write(STDERR_FILENO, error_msg, strlen(error_msg));
            close(file);
            return 1;
        }
    }

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



# Протокол работы программы

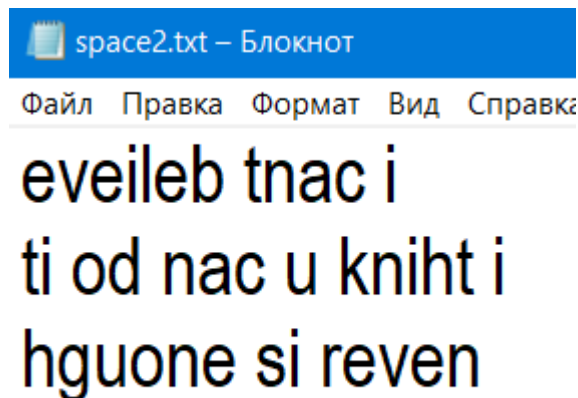
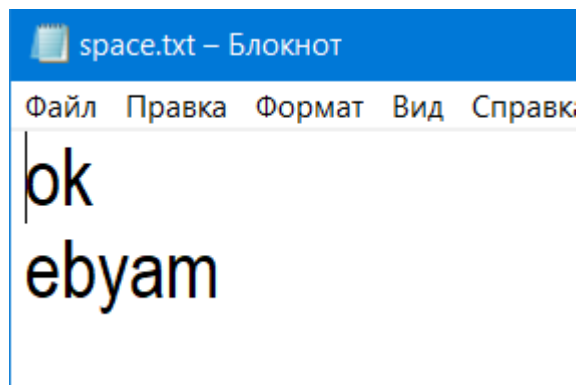
Тестирование:

```

miron@DESKTOP-GD72A05:~/LABS/lab 1$ ./parent
Введите имя файла для child1: space.txt
Введите имя файла для child2: space2.txt
Введите строку: i cant believe
Введите строку: ko
Введите строку: maybe
Введите строку: i think u can do it
Введите строку: never is enough
Введите строку: ^C
miron@DESKTOP-GD72A05:~/LABS/lab 1$ _

```

 space.txt	20.11.2024 22:10	Текстовый докум...	1 КБ
 space2.txt	20.11.2024 22:10	Текстовый докум...	1 КБ



## Вывод

Было интересно решать лабораторную работу. Я научился использовать некоторые системные вызовы, а также обмениваться данными между процессами с помощью каналов. Было интересно узнать как можно писать программы используя их. Возникли трудности с обработкой всех ошибок системных вызовов в программе.