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: Шведов А.И.

Преподаватель: Бахарев В.Д.

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата: 03.10.24

Москва, 2024

**Постановка задачи**

**Вариант 3:**

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 int. Количество чисел может быть произвольным.

**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 эквиваентными 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 получнные 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;

        }

    }

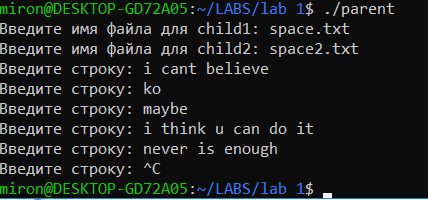
    close(file);

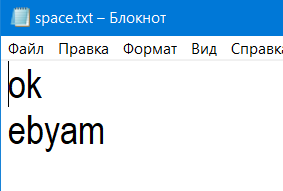
    return 0;

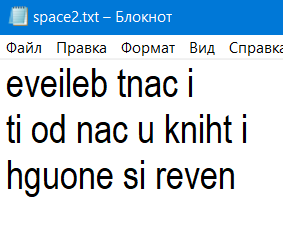
}

**Протокол работы программы**

Тестирование:





**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.