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-211БВ-24

Студент: Акшевский С. Д.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 29.09.25

Москва, 2025

Постановка задачи

Вариант 9.

Родительский процесс создает дочерний процесс. Первой строчкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия файла с таким именем на чтение. Стандартный поток ввода дочернего процесса переопределяется открытым файлом. Дочерний процесс читает команды из стандартного потока ввода. Стандартный поток вывода дочернего процесса перенаправляется в pipe1. Родительский процесс читает из pipe1 и прочитанное выводит в свой стандартный поток вывода. Родительский и дочерний процесс должны быть представлены разными программами.

В файле записаны команды вида:«число число число<endline>». Дочерний процесс производит деление первого числа команда, на последующие числа в команде, а результат выводит в стандартный поток вывода. 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-копию родителя
- int wait(NULL); - ожидает завершения любого дочернего процесса
- int execl(...); - заменяет текущий процесс новой программой
- int pipe(int *fd); - создает неименованный канал для обмена данными между процессами
- int open(const char *path, int flags); - открывает файл
- int close(int fd); - закрывает файловый дескриптор
- ssize_t read(int fd, void *buf, size_t count); - читает данные из файла
- ssize_t write(int fd, const void *buf, size_t count); - записывает данные в файл
- int dup2(int oldfd, int newfd); - дублирует файловый дескриптор

Родительский процесс начинает работу с чтения имени файла от пользователя, после чего создает неименованный канал (pipe) и порождает дочерний процесс с помощью fork(). Дочерний процесс перенаправляет свой стандартный вывод в канал и запускает программу-обработчик, в то время как родительский процесс перенаправляет свой ввод на чтение из этого канала, обеспечивая передачу данных между процессами.

Дочерний процесс открывает указанный файл и перенаправляет на него свой стандартный ввод, после чего читает числа из файла и выполняет арифметические

операции. Первое число в каждой строке делится на все последующие числа в той же строке, при этом осуществляется проверка деления на ноль. Результаты вычислений передаются через канал обратно в родительский процесс, который выводит их пользователю. При обнаружении деления на ноль оба процесса корректно завершают свою работу.

Код программы

main.c

```
#include <string>
#include "../include/io_helper.h"
#include <sys/wait.h>

int main(){
    std::string inputFileName;
    inputFileName = read_word();

    int pipe1[2]; //Child -> parent
        if(pipe(pipe1) == -1){
        perror("pipe");
        exit(EXIT_FAILURE);
    }

    int processId = fork();
    if(processId == -1){
        perror("fork");
        exit(EXIT_FAILURE);
    }
    else if(processId > 0){ //parent
        close(pipe1[1]);

        char buf[4096];
        int bytes;

        while(bytes = read(pipe1[0], buf, sizeof(buf))){
            if(bytes < 0){
                char er[] = "failed to read from pipe1!\n";
                write(STDERR_FILENO, er, sizeof(er));
                exit(EXIT_FAILURE);
            }
            write(STDOUT_FILENO, buf, bytes);
        }
```

```
        close(pipe1[0]);
        wait(NULL);
    }
    else { // child
        close(pipe1[0]);
        dup2(pipe1[1], STDOUT_FILENO);

        execl("child", "child", inputFileName.c_str(), NULL);

        char er[] = "execl call failed!\n";
        write(STDERR_FILENO, er, sizeof(er));
        exit(EXIT_FAILURE);
    }
}
```

child.cpp

```
#include "../include/child.h"
#include "../include/io_helper.h"


int main(int argc, char* argv[]){
    if (argc != 2) {
        char er[] = "Usage: child <filename>\n";
        write(STDERR_FILENO, er, sizeof(er));
        exit(EXIT_FAILURE);
    }

    std::string inputFileName = std::string(argv[1]);
    int inputFileFD = open(inputFileName.c_str(), O_RDONLY);
    if(inputFileFD == -1){
        char er[] = "Cant open the file\n";
        write(STDERR_FILENO, er, sizeof(er));
        exit(EXIT_FAILURE);
    }

    dup2(inputFileFD, STDIN_FILENO);
    close(inputFileFD);

    int is_last = 0;
    double firstNum = read_double(&is_last);
    if(is_last) return 0;

    double secondNum;
    do {
        secondNum = read_double(&is_last);
```

```
        if(secondNum == 0){
            write_line("division by zero");
            return 0;
        }
        write_line(double_to_string(firstNum / secondNum));
    } while(!is_last);

    return 0;
}
```

io_helper.h

```
#ifndef IO_HELPER_H
#define IO_HELPER_H
#include <string>

char read_char();
std::string read_word(int *is_last);
std::string read_word();
double read_double(int *is_last);

void write_char(const char c);
void write_string(const std::string &str);
void write_line(const std::string &line);

void reverse_string(std::string &str);
std::string double_to_string(const double num);
#endif
```

io_helper.c

```
#include "../include/io_helper.h"
#include <ctype.h>
#include <fcntl.h>
#include <stdlib.h>
#include <unistd.h>

char read_char(int &status) {
    char buffer[1];
    status = read(STDIN_FILENO, buffer, sizeof(char));
    return buffer[0];
}
std::string read_word(int *is_last) {
    std::string word = "";
    int status;
    char c;
    while((c = read_char(status)) != ' ' && c != '\n' && status == 1){
        word += c;
```

```
    }
    if(c == '\n' || status != 1) {
        *is_last = 1;
    }
    else{
        *is_last = 0;
    }
    return word;
}
std::string read_word(){
    int status;
    return read_word(&status);
}
double read_double(int *is_last) {
    std::string word;
    word = read_word(is_last);
    return atof(word.c_str());
}

void write_char(const char c) {
    char buffer[1] = {c};
    write(STDOUT_FILENO, buffer, sizeof(char));
}
void write_string(const std::string &str) {
    write(STDOUT_FILENO, str.c_str(), str.length() * sizeof(char));
}
void write_line(const std::string &line) {
    write_string(line);
    write_char('\n');
}
void reverse_string(std::string &str){
    for(int i = 0; i < str.length() / 2; i++){
        char buffer = str[i];
        str[i] = str[str.length() - i - 1];
        str[str.length() - i - 1] = buffer;
    }
}

std::string double_to_string(const double num){
    int intNum = (int)num;
    std::string intPart = "";



    if(num < 0){
        intNum = -intNum;
```

```
        intPart += '-';
    }
    while(intNum != 0){
        intPart += (intNum % 10) + '0';
        intNum /= 10;
    }

    reverse_string(intPart);
    int accuracy = 4;
    if(intPart == ""){
        intPart += '0';
    }

    double fractNum = abs(num) - (int)num;
    if(fractNum == 0) return intPart;
    intPart += '.';
    std::string fractPart = "";

    for(int i = 0; i < accuracy; i++){
        if(fractNum - (int)fractNum == 0){
            break;
        }
        fractNum *= 10;
        fractPart += (int)fractNum % 10 + '0';
    }
    return intPart + fractPart;
}
```

Протокол работы программы

test1.txt:

100 10 50 100 1000 10000 53 52 77 13

test2.txt

8.8 1 1.1 2.2 4.4 5.5 8.8 10 5

test3.txt

5 1 2 3 4 10.523 0 523.235

Тестирование:

```
./main
test1.txt
10
2
```

```
0.1
0.01
1.8867
1.9230
1.2987
7.6923

./main
test2.txt
8.8000
8
4
2
1.6000
1
0.88
1.7600

./main
test3.txt
5
2.5
1.6666
1.25
0.4751
division by zero
```

Вывод

В ходе лабораторной работы получил навыки работы с системными вызовами в языке си и успешно реализовал программу, использующую межпроцессное взаимодействие через механизм pipe. Родительский процесс организует ввод данных и вывод результатов, в то время как дочерний процесс выполняет арифметические операции с числами из файла и передачу результатов в родительский процесс.