Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №2 по курсу**

**«Операционные системы»**

Студент: Марченко Алексей Эдуардович

Группа: М8О-207Б-21

Вариант: 7

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2023

**Содержание**

1. Репозиторий
2. Постановка задачи
3. Общие сведения о программе
4. Общий метод и алгоритм решения
5. Исходный код
6. Демонстрация работы программы
7. Выводы

**Репозиторий**

https://github.com/MarchAleksey/OS\_Labs

**Постановка задачи**

**Цель работы**

Приобретение практических навыков в управлении процессами в ОС, обеспечение обмена данных между процессами посредством каналов.

**Задание**

Составить и отладить программу на языке Си, осуществляющую работу с процессами и

взаимодействие между ними в одной из двух операционных систем. В результате работы

программа (основной процесс) должен создать для решение задачи один или несколько

дочерних процессов. Взаимодействие между процессами осуществляется через системные

сигналы/события и/или каналы (pipe).

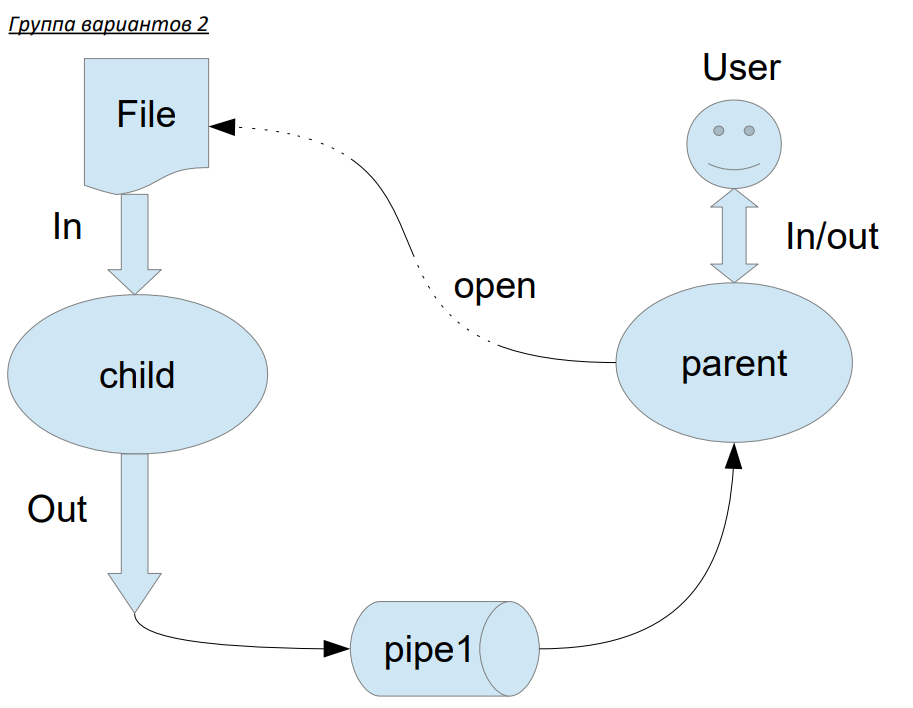
Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы.

**Группа вариантов 2**

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.

**Вариант 7**

В файле записаны команды вида: «число число число». Дочерний процесс считает их сумму и выводит результат в стандартный поток вывода. Числа имеют тип float. Количество чисел может быть произвольным.



**Общие сведения о программе**

Программа компилируется из файла main.cpp, child.cpp. В программе используются следующие системные вызовы:

1. open() - открывает файл.
2. pipe() - существует для передачи информации между различными процессами.
3. fork() - создает новый процесс.
4. execpl() - передает процесс на исполнение другой программе.
5. read() - читает данные из файла.
6. dup2() – дублирует файловые дескрипторы в Linux.
7. close() - закрывает файл.

**Общий метод и алгоритм решения**

Пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия файла с таким именем на запись для дочернего процесса. Дочерний процесс вычислит сумму чисел из файла и передаст ее обратно через канал в родительский процесс, который выведет сумму на экран.

**Исходный код**

**main.cpp**

#include <unistd.h>

#include <fcntl.h>

#include <iostream>

int main() {

    std::string file\_name;

    if (!getline(std::cin, file\_name)) {

        std::cerr << "input error\n";

        return -1;

    }

    int file = open(file\_name.c\_str(), O\_RDONLY);

    if (file == -1) {

        std::cerr << "file error\n";

        return -1;

    }

    int fd[2];

    if (pipe(fd) == -1) {

        std::cerr << "pipe error\n";

        return -1;

    }

    pid\_t fork\_id = fork();

    if (fork\_id == -1) {

        std::cerr << "fork error\n";

        return -1;

    } else if (fork\_id == 0) {

        execlp("./child",

               std::to\_string(fd[0]).c\_str(),

               std::to\_string(fd[1]).c\_str(),

               std::to\_string(file).c\_str(),

               NULL

        );

    } else {

        float sum;

        char buf;

        while (read(fd[0], &buf, sizeof(buf)) > 0 && buf != EOF) {

            read(fd[0], &sum, sizeof(sum));

            std::cout << sum << std::endl;

        }

    }

    close(fd[1]);

    close(fd[0]);

    close(file);

    return 0;

}

**child.cpp**

#include <unistd.h>

#include <iostream>

int main(int argc, char const \*argv[])

{

    int fd[2];

    fd[0] = std::stoi(argv[0]);

    fd[1] = std::stoi(argv[1]);

    int file = std::stoi(argv[2]);

    if (dup2(file, 0) == -1) {

        std::cerr << "dub error\n";

        return -1;

    }

    float x, sum = 0;

    char c;

    do {

        std::cin >> x;

        sum += x;

        c = getchar();

        if (c == '\n') {

            write(fd[1], &c, sizeof(c));

            write(fd[1], &sum, sizeof(sum));

            sum = 0;

        }

    } while(c != EOF);

    write(fd[1], &c, sizeof(c));

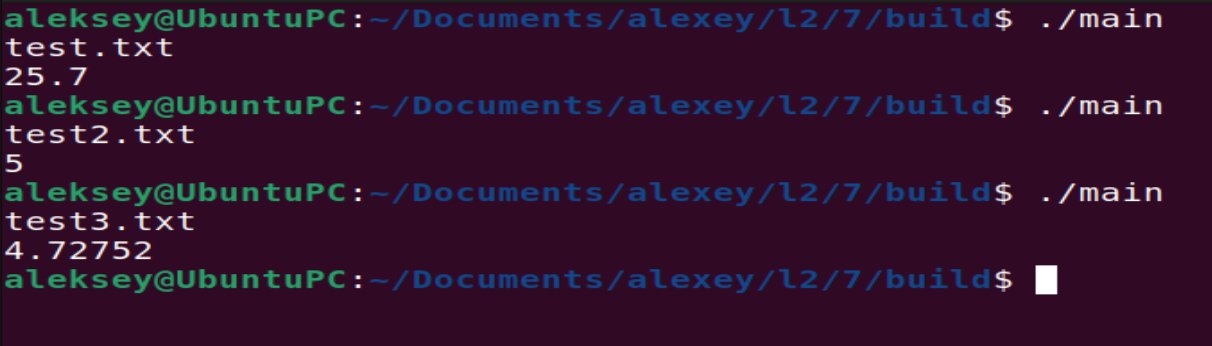
    close(fd[1]);

    close(fd[0]);

    return 0;

}

**Демонстрация работы программы**



**Выводы**

В ходе выполнения лабораторной работы №2 я приобрел практические навыки в управлении процессами в ОС, и в обеспечении обмена данных между процессами посредством каналов.