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: Примаченко Александр Александрович

Группа: М8О-208Б-20

Вариант: 3

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2021

**Содержание**

1. Репозиторий
2. Постановка задачи
3. Общие сведения о программе
4. Исходный код
5. Выводы

**Репозиторий**

<https://github.com/SashaPaladin/OS/tree/main/2_lab>

**Постановка задачи**

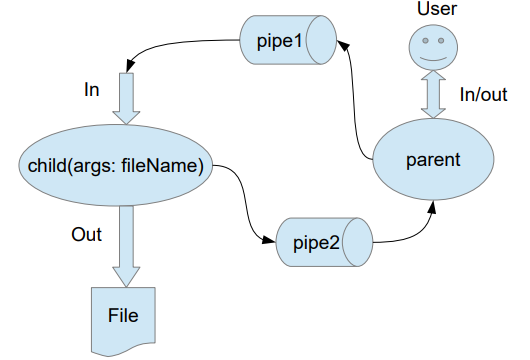
**Цель работы**

Приобретение практических навыков в:

1. Управление процессами в ОС
2. 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.



**Общие сведения о программе**

Программа компилируется из файла main.cpp. Также используется заголовочные файлы: unistd.h, stdio.h , stdlib.h, ctype.h. В программе используются следующие системные вызовы:

**fork** - создает копию текущего процесса, который является дочерним процессом для текущего процесса

**pipe** - создаёт однонаправленный канал данных, который можно использовать для взаимодействия между процессами.

**fflush** - если поток связан с файлом, открытым для записи, то вызов приводит к физической записи содержимого буфера в файл. Если же поток указывает на вводимый файл, то очищается входной буфер.

**close** - закрывает файл.

**read** - читает количество байт(третий аргумент) из файла с файловым дескриптором(первый аргумент) в область памяти(второй агрумент).

**write -**  записывает в файл с файловым дескриптором(первый аргумент) из области памяти(второй аргумент) количество байт(третий аргумент).

**perror –** вывод сообщения об ошибке.

**Исходный код**

#include "unistd.h"

#include <iostream>

#include <string>

#include <vector>

#include <fstream>

#include <sstream>

int main() {

std::cout << "Enter file name:" << std::endl;

std::string fileName;

std::cin >> fileName;

int fd[2];

pipe(fd);

int id = fork();

if (id == -1) {

perror("fork error");

return -1;

} else if (id == 0) {

int n;

std::ofstream out(fileName);

int res;

read(fd[0], &n, sizeof(int));

int \*p = new int[n];

read(fd[0], p, sizeof(int[n]));

res = p[0];

std::cout << res << std::endl;

for (int i = 0; i < n; i++) {

if (p[i] == 0) exit(-1);

else res = res / p[i];

}

std::cout << "res: " << res << std::endl;

out << res << std::endl;

write(fd[1], &res, sizeof(int));

delete[] p;

out.close();

close(fd[0]);

close(fd[1]);

} else {

int num;

std::string line;

std::vector<int> vec;

std::cin.ignore();

getline(std::cin, line));

std::cout << "Enter numbers:" << std::endl;

std::stringstream inp(line);

while (inp >> num) {

vec.push\_back(num);

}

int n = vec.size();

int \*p = new int[n];

for (int i = 0; i < vec.size(); i++) {

p[i] = vec[i];

}

write(fd[1], &n, sizeof(int));

write(fd[1], p, sizeof(int[n]));

delete[] p;

}

close(fd[0]);

close(fd[1]);

return 0;

}

**Выводы**

Существуют специальные системные вызовы(fork) для создания процессов, также существуют специальные каналы pipe, которые позволяют связать процессы и обмениваться данными при помощи этих pipe-ов. При использовании fork важно помнить, что фактически создается копию вашего текущего процесса и неправильная работа может привести к неожиданным результатам и последствиям, однако создание процессов очень удобно, когда вам нужно выполнять несколько действий параллельно. Также у каждого процесса есть свой id, по которому его можно определить. Также важно работать с чтением и записью из канала, помня что read, write возвращает количество успешно считанных/записанных байт и оно не обязательно равно тому значению, которое вы указали. Также важно не забывать закрывать pipe после завершения работы.