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: Ивченко Анна Владимировна

Группа: М8О-208Б-20

Вариант: 4 Преподаватель: Миронов Евгений Сергеевич

Оценка: Дата: Подпись:

Москва, 2021

# 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

# Постановка задачи

**Цель работы**

Приобретение практических навыков в:

1. Управление процессами в ОС
2. Обеспечение обмена данных между процессами посредством каналов

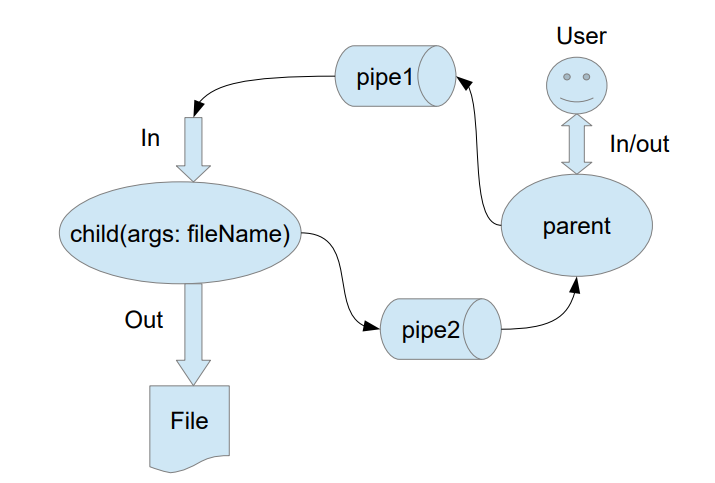
# Задание

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 работе.

Родительский процесс создает дочерний процесс.

4 вариант: Пользователь вводит команды вида: «число число число». Далее эти числа передаются от родительского процесса в дочерний. Дочерний процесс производит деление parent child(args: fileName) pipe1 pipe2 In/out User File In Out 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 float. Количество чисел может быть произвольным

# Общие сведения о программе

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

1. **fork** - создает копию текущего процесса, который является дочерним процессом для текущего процесса
2. **pipe** - создаёт однонаправленный канал данных, который можно использовать для взаимодействия между процессами.
3. **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.
4. **close** - закрывает файл.
5. **read** - читает количество байт(третий аргумент) из файла с файловым дескриптором(первый аргумент) в область памяти(второй агрумент).
6. **write -** записывает в файл с файловым дескриптором(первый аргумент) из области памяти(второй аргумент) количество байт(третий аргумент).
7. **perror –** вывод сообщения об ошибке.

# Общий метод и алгоритм решения

Для реализации поставленной задачи необходимо:

1. Изучить принципы работы fork, pipe, fflush, close, read, write.
2. Написать программу, которая будет работать с 2-мя процессами: родительский и дочерний, процессы связываются между собой при помощи pipe-ов.

Организовать работу с выделением памяти под строку неопределенной длины и запись длины в массив строки в качестве первого элемента для передачи между процессами через pipe.

# Исходный код

main.cpp

#include "unistd.h"

#include <iostream>

#include <string>

#include <vector>

#include <fstream>

#include <sstream>

using namespace std;

int main()

{

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

string fileName;

cin >> fileName;

int fd[2];

int fd1[2];

pipe(fd);

int id = fork();

if (id == -1)

{

perror("fork error");

return -1;

}

else if (id == 0)

{

printf("[%d] It's child\n", getpid());

float res;

int n = 1;

ofstream out(fileName);

while (read(fd[0], &n, sizeof(int)) && n != 0) {

float \*p = new float[n];

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

res = p[0];

for (int i = 1; i < n; i++){

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

res = res / p[i];

}

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

out << res << endl;

delete [] p;

}

out.close();

close(fd[0]);

close(fd[1]);

}

else

{

float num;

string line;

vector<float> vec;

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

cin.ignore();

while(cin.good() && getline(cin, line)){

stringstream inp(line);

while (inp >> num) {

vec.push\_back(num);

}

int n = vec.size();

float \*p = new float[n];

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

p[i] = vec[i];

}

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

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

vector<float>().swap(vec);

delete [] p;

}

close(fd[0]);

close(fd[1]);

}

return 0;

}

# Демонстрация работы программы

# 

# Выводы

В ходе проделанной работы, были получены знания по управлению процессами в ОС Unix и обеспечении обмена данных между процессами с помощью каналов