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: Литовченко Анна

Группа: М8О-207Б-21

Вариант: 5

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2022

**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/Annalitov/OS

**Постановка задачи**

**Цель работы**

Приобретение практических навыков в:

* Управление 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 1

Родительский процесс создает дочерний процесс. Первой строчкой пользователь в консоль родительского процесса пишет имя файла, которое будет передано при создании дочернего процесса. Родительский и дочерний процесс должны быть представлены разными программами. Родительский процесс передает команды пользователя через pipe1, который связан с стандартным входным потоком дочернего процесса. Дочерний процесс при необходимости передает данные в родительский процесс через pipe2. Результаты своей работы дочерний процесс пишет в созданный им файл. Допускается просто открыть файл и писать туда, не перенаправляя стандартный поток вывода.

5 вариант) Пользователь вводит команды вида: «число<endline>». Далее это число передается от родительского процесса в дочерний. Дочерний процесс производит проверку на простоту. Если число составное, то в это число записывается в файл. Если число отрицательное или простое, то тогда дочерний и родительский процессы завершаются.

**Общие сведения о программе**

Программа компилируется из файла main.cpp. Дочерний процесс представлен программой child.cpp.

В программе используются следующие системные вызовы:

1. pipe() - создаёт однонаправленный канал данных, который можно использовать для взаимодействия между процессами.
2. fork() - создает копию текущего процесса, который является дочерним процессом для текущего процесса
3. execl() - загружает и запускает другие программы, известные как "дочерние" процессы.
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**

С помощью вызова fork() создаются родительский и дочерний процессы. Родительский процесс считывает название будущего файла и число, затем передает всё в дочерний процесс, который создает файл и выполняет проверку данного числа. Если число удовлетворяет условиям, то он записывает его в файл, в противном случае передает родительскому процессу информацию о том, что число не подошло.

**Исходный код**

**main.cpp**

#include <iostream>

#include <string>

#include <cstdlib>

#include <sys/types.h>

#include <unistd.h>

#include <fstream>

#include <errno.h>

#include <signal.h>

#include <sys/wait.h>

using namespace std;

int main(){

fstream f;

string filename;

cout<<"Enter a filename: "<<endl;

cin >> filename;

int fd\_1[2];

int fd\_2[2];

int pipe\_1[2];

int pipe\_2[2];

if (pipe(pipe\_1) == -1){

perror("pipe");

exit(EXIT\_FAILURE);

}

if (pipe(pipe\_2) == -1){

perror("pipe");

exit(EXIT\_FAILURE);

}

string num;

pid\_t id = fork();

if (id == -1){

perror("fork");

exit(EXIT\_FAILURE);

} else if (id == 0) {

fd\_1[0] = pipe\_1[0];

fd\_1[1] = pipe\_1[1];

fd\_2[0] = pipe\_2[0];

fd\_2[1] = pipe\_2[1];

execl("./child", to\_string(fd\_1[0]).c\_str(), to\_string(fd\_1[1]).c\_str(), to\_string(fd\_2[0]).c\_str(), to\_string(fd\_2[1]).c\_str(), filename.c\_str(), NULL);

} else {

cout << "Enter a number: " << endl;

cin >> num;

int s\_size = num.size();

char str\_array[s\_size];

for (int k = 0; k < s\_size; ++k) {

str\_array[k] = num[k];

}

write(pipe\_1[1], &s\_size, sizeof(int));

write(pipe\_1[1], str\_array, sizeof(char)\*s\_size);

int flag\_0;

read(pipe\_2[0], &flag\_0, sizeof(int));

if (flag\_0 == 1) {

cout << "The number is prime or negative" << endl;

}

}

close(pipe\_1[0]);

close(pipe\_1[1]);

close(pipe\_2[0]);

close(pipe\_2[1]);

return 0;

}

**child.cpp**

#include <iostream>

#include <string>

#include <cstdlib>

#include <sys/types.h>

#include <unistd.h>

#include <fstream>

#include <errno.h>

#include <signal.h>

#include <sys/wait.h>

using namespace std;

int isNotPrime(int n)

{

if (n < 2) {

return 0;

} else {

for (int i = 2; i \* i < n + 1; i++) {

if (n % i == 0) {

return 1;

}

}

}

return 0;

}

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

string filename = argv[4];

int fd\_1[2];

int fd\_2[2];

int flag\_1 = 1;

int flag\_2 = 2;

fd\_1[0] = stoi(argv[0]);

fd\_1[1] = stoi(argv[1]);

fd\_2[0] = stoi(argv[2]);

fd\_2[1] = stoi(argv[3]);

fstream f;

f.open(filename, fstream::in | fstream::out | fstream::app);

while(true) {

int num\_size;

read(fd\_1[0], &num\_size, sizeof(int));

char num\_str[num\_size];

read(fd\_1[0], &num\_str, sizeof(char)\*num\_size);

string result;

for (int i = 0; i < num\_size; i++) {

result.push\_back(num\_str[i]);

}

int number;

int number\_1;

number = stoi(result);

number\_1 = abs(number);

if ( number > 0 && isNotPrime(number\_1) > 0 ) {

f << result << endl;

cout << "A number " << result << " is added to file!" << endl;

write(fd\_2[1], &flag\_2, sizeof(int));

} else {

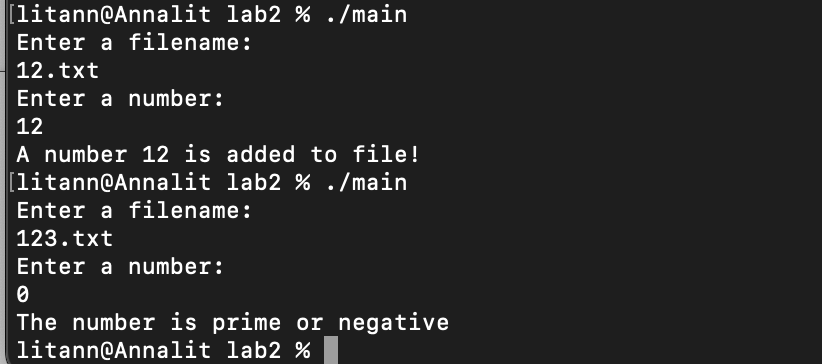
write(fd\_2[1], &flag\_1, sizeof(int));

}

}

return 0

}

**Демонстрация работы программы**

**Выводы**

Проделав данную лабораторную работу я приобрела практические навыки в управлении процессами в операционной системе и смогла произвести обмен данными между ними, использую каналы.