Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №1 по курсу**

**«Операционные системы»**

**ВЗАИМОДЕЙСТВИЕ МЕЖДУ ПРОЦЕССАМИ**

Студент: Кудрявов Леонид Вадимович

Группа: М8О–208Б–22

Вариант: 10

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2023.

**Постановка задачи**

## Цель работы

Целью является приобретение практических навыков в:

* Управлении процессами в ОС
* Обеспечении обмена данными 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.

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.

10 вариант) В файле записаны команды вида: «число<endline>». Дочерний процесс производит проверку этого числа на простоту. Если число составное, то дочерний процесс пишет это число в стандартный поток вывода. Если число отрицательное или простое, то тогда дочерний и родительский процессы завершаются. Количество чисел может быть произвольным.

**Общие сведения о программе**

Программа компилируется из файла main.cpp. Помимо этого, есть программа child\_process.cpp, которая запускается при помощи execlp. Обе программы собираются при помощи системы сборки CMake. В программе используются следующие системные вызовы:

1. **fork** - создает дочерний процесс, повторяющий программу родительского.
2. **execlp** - позволяет запустить исполняемый файл изнутри программы.
3. **pipe** - создает пайп для передачи данных между процессами
4. **dup2** - переназначение файлового дескриптора
5. **open** - открытие файла
6. **сlose** - закрытие файла

**Общий метод и алгоритм решения**.

Для реализации поставленной задачи необходимо:

1. Изучить принципы работы fork, exec, pipe.
2. Написать основную программу процесса main.cpp, которая будет осуществлять вызов системных функций.
3. Написать программу child\_process.cpp, которая будет запускаться из дочернего процесса при помощи exec

**Основные файлы программы**

**Main.cpp**

#include <iostream>

#include <unistd.h>

#include <string>

#include <fstream>

#include <fcntl.h>

int main() {

const char\* CHILD\_PATH = "/home/petrkozyrev/osi/os\_lab\_1/build/child";

std::string name ="/home/petrkozyrev/osi/os\_lab\_1/test/";

char smb;

int ind = 0;

if( read(STDIN\_FILENO, &smb, 1) == -1){

perror("Reading error\n");

exit(1);

}

while( smb != '\n'){

name += smb;

if ( read(STDIN\_FILENO, &smb, 1) == -1){

perror("Reading error\n");

exit(1);

}

}

const char\* file\_name = name.c\_str();

int pipe\_fd[2];

int err = pipe(pipe\_fd);

if( err == -1){

perror("Pipe error\n");

exit(1);

}

pid\_t pid = fork();

if( pid == -1){

perror("Pid errror\n");

exit(1);

}

if( pid == 0){

close(pipe\_fd[0]);

int input\_file = open(file\_name,O\_RDONLY);

if( input\_file == -1){

perror("Cant open file\n");

exit(1);

}

dup2(input\_file,STDIN\_FILENO);

if( dup2(pipe\_fd[1],STDOUT\_FILENO) == -1){

perror("Cant dup2\n");

exit(1);

}

execlp(CHILD\_PATH, CHILD\_PATH, NULL);

close(input\_file);

perror("execlp\n");

exit(EXIT\_SUCCESS);

}

else{

close(pipe\_fd[1]);

char buff;

while( (read(pipe\_fd[0], &buff, 1) > 0)){

write(STDOUT\_FILENO,&buff,1);

}

write(STDOUT\_FILENO,"\n",1);

close(pipe\_fd[0]);

exit(EXIT\_SUCCESS);

}

return 0;

}

**Child-process.cpp**

#include <iostream>

#include <unistd.h>

#include <string>

#include <fstream>

#include <fcntl.h>

#include "prime.cpp"

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

char buff[256];

char smb;

int number=0,sgn = 1,ind = 0,len\_lst\_number = 0;

if (read(STDIN\_FILENO, &smb, 1) == -1){

return -1;

}

while( smb != ' '){

buff[ind] = smb;

++ind;

if( smb != '\n'){

if( smb == '-'){

sgn = -1;

}

len\_lst\_number++;

number = (number\*10) + (smb-'0');

read(STDIN\_FILENO, &smb, 1);

}

else{

number \*= sgn;

if( isPrime(number) or number <= 0){

break;

}

number = 0;

sgn = 1;

len\_lst\_number = 0;

read(STDIN\_FILENO, &smb, 1);

}

}

for( int i=0; i < ind-len\_lst\_number - 1; i++){

write(STDOUT\_FILENO,&buff[i],1);

}

return 0;

}

**Prime.cpp**

#include <cmath>

#include "prime.h"

bool isPrime( int number) {

if( number < 2) {

return false;

}

for( int i = 2; i <= sqrt(number); i++) {

if( number % i == 0) {

return false;

}

}

return true;

}

**CMakeLists.txt**

cmake\_minimum\_required(VERSION 3.10)

project(osi\_1)

set(CMAKE\_CXX\_STANDARD 17)

add\_executable(main scr/main.cpp)

add\_executable(child scr/child\_process.cpp)

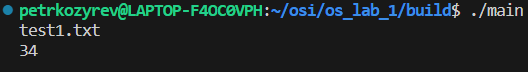
**Пример работы**

1. Test1.txt:

34

11

12



1. Test2.txt

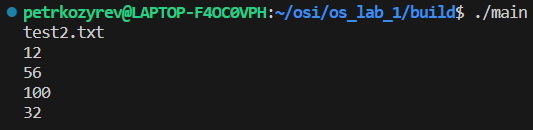
12

56

100

32

19



1. Test3.txt

3

324

53

8239

22

Изображение выглядит как текст, Шрифт, снимок экрана, Графика

Автоматически созданное описание

1. Test4.txt

10004

50005

123456

77777777

4387438

5

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

1. Test5.txt

444444

55555

12

42

887878734

32324890

843958

10

120

10015

87654

1010290

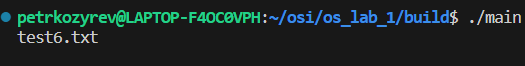
9

19

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

1. Test6.txt (пустой файл)



**Вывод**

Я изучил основные системные вызовы для работы с процессами и обмена данными между ними. Для решения задания разделяю на родительский и дочерний процесс с помощью системного вызова fork. Для обмена данными между процессами использую pipe. Также изучил как работают файловые дескрипторы, чтобы перенаправлять данные из файлов и пайпа на потоки ввода/вывода при помощи dup2.