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: Кайдалова Александра Андреевна

Группа: М8О–212Б–22

Вариант: 10

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2023.

**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).

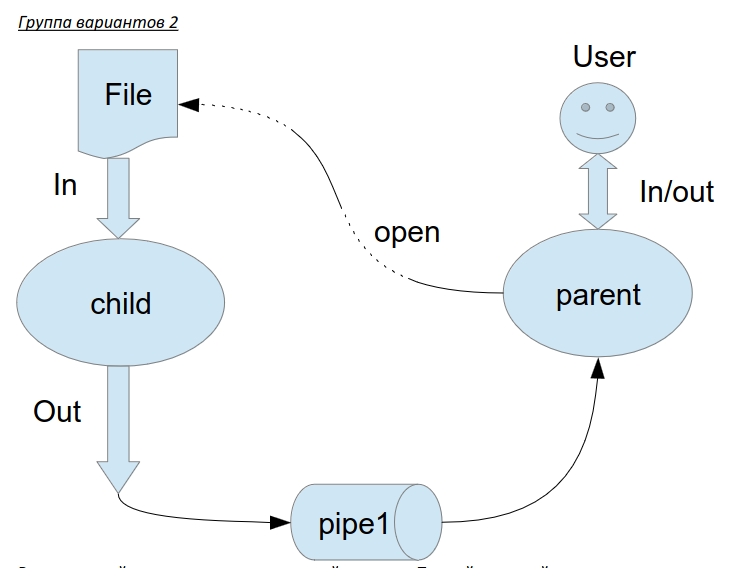
Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы.

**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 10**

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.c. Также используется заголовочные файлы: unistd.h, fcntl.h, simple.h . В программе используются следующие системные вызовы:

1. pid\_t fork() - создание дочернего процесса
2. int pipe(int pipefd[2]) - создание неименованного канала для передачи данных между процессами
3. int dup2(int oldfd, int newfd) - переназначение файлового дескриптора
4. int open(const char \*pathname, int flags, mode\_t mode) - открытие\создание файла
5. int close(int fd) - закрыть файл

**Общий метод и алгоритм решения**.

Для реализации поставленной задачи необходимо:

1. Изучить принципы работы pipe, fork, dub2, execlp.
2. Написать основную программу main, внутри которой создать дочерний поток и путь взаимодействия потоков.
3. Написать программу дочернего потока.
4. Написать функцию проверки числа на простоту.
5. Создать тесты, учитывающие все возможные случаи.
6. Написать Cmake.

**Основные файлы программы**

**main.cpp**

#include <unistd.h>

#include <fstream>

#include <fcntl.h>

#include <string>

using namespace std;

int main(){

// Чтение, запись имени тестового файла

write(STDOUT\_FILENO, "Enter the file name: \n", 22);

string name ="/home/alexandra/Рабочий стол/ос 3 сем/OS\_LW1/tests/";

char part[256];

int q = 0;

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

perror("Reading error. \n");

exit(-1);

}

else{

while (part[q] != '\n'){

name += part[q];

q++;

}

}

// Открытие тестового файла

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

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

if (file == -1){

perror("File error. \n");

exit(-1);

}

// Создание pipe

int fd[2];

int pipe1 = pipe(fd);

if (pipe1 == -1){

perror("Pipe1 error. \n");

exit(-1);

}

// Создание дочернего потока

pid\_t pid = fork();

if (pid == -1){

perror("Pid errror. \n");

exit(-1);

}

// Работа с дочерним потоком

if (pid == 0){

close(fd[0]);

if (dup2(file,STDIN\_FILENO) == -1){

perror("Dup2 error. \n");

exit(-1);

}

else{

dup2(file,STDIN\_FILENO);

}

const char\* child = "/home/alexandra/Рабочий стол/ос 3 сем/OS\_LW1/build/child";

execlp(child, child, NULL);

close(file);

close(fd[1]);

exit(0);

}

// Работа с родительским потоком

if (pid > 0){

close(fd[1]);

char result;

while ((read(fd[0], &result, sizeof(result)) > 0)){

write(STDOUT\_FILENO, &result, sizeof(result));

}

close(fd[0]);

exit(0);

}

exit(0);

}

**child.cpp**

#include <unistd.h>

#include <string>

#include <fstream>

#include <fcntl.h>

#include "simple.h"

using namespace std;

int main(){

char result[256]; int r = 0;

char num;

int number = 0;

int znak;

int size = 0;

// Проверка чтения данных

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

perror("Readind error. \n");

exit(-1);

}

else{

// Цикл работает, пока находяться числа или \n

while (num != ' '){

result[r] = num;

r++;

size++;

// Составление одного числа, происходит в рамках одной строки до \n

if (num != '\n'){

if (num == '-'){

znak = -1;

}

else{

znak = 1;

}

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

read(STDIN\_FILENO, &num, sizeof(num));

}

else{

// Обработка составленного числа

number \*= znak;

if (Simple(number)){

break;

}

number = 0;

size = 0;

read(STDIN\_FILENO, &num, sizeof(num));

}

}

}

// Запись составленных чисел в поток вывода (связан с fd[1] pipe1)

for (int q = 0; q < r - size; q++){

write(STDOUT\_FILENO, &result[q], sizeof(result[q]));

}

exit(0);

}

**simple.cpp**

#include <cmath>

bool Simple (int number) {

// Проверка числа на 0, 1 или любое отрицательноее число

if (number < 2) {

return true;

}

// Проверка числа на простоту

for( int w = 2; w <= sqrt(number)+1; w++) {

if( number % w == 0) {

return false;

}

}

// Если ничего не сработало - число простое

return true;

}

**simple.h**

bool Simple( int number);

**CmakeLists.txt**

cmake\_minimum\_required(VERSION 3.10)

project(osi\_1)

set(CMAKE\_CXX\_STANDARD 17)

add\_executable(main main.cpp)

add\_executable(child child.cpp simple.cpp)

**test1**

6

8

16

42

7

54

**test2**

6

3334

-1

0

12678

**test3**

4

1

67

89

5

**test4**

12456

87654

9876543218

10

0

7

-1

0

**test5**

0

2

4

6

8

10

**test6**

4

1234567890

-16

4

-22

**Пример работы**

alexandra@alexandra-HVY-WXX9:~/Рабочий стол/ос 3 сем/OS\_LW1/build$ make

Consolidate compiler generated dependencies of target main

[ 40%] Built target main

Consolidate compiler generated dependencies of target child

[100%] Built target child

alexandra@alexandra-HVY-WXX9:~/Рабочий стол/ос 3 сем/OS\_LW1/build$ ./main

Enter the file name:

test1

6

8

16

42

alexandra@alexandra-HVY-WXX9:~/Рабочий стол/ос 3 сем/OS\_LW1/build$ ./main

Enter the file name:

test2

6

3334

-alexandra@alexandra-HVY-WXX9:~/Рабочий стол/ос 3 сем/OS\_LW1/build$ ./main

Enter the file name:

test3

4

alexandra@alexandra-HVY-WXX9:~/Рабочий стол/ос 3 сем/OS\_LW1/build$ ./main

Enter the file name:

test4

12456

87654

9876543218

10

alexandra@alexandra-HVY-WXX9:~/Рабочий стол/ос 3 сем/OS\_LW1/build$ ./main

Enter the file name:

test5

alexandra@alexandra-HVY-WXX9:~/Рабочий стол/ос 3 сем/OS\_LW1/build$ ./main

Enter the file name:

test6

4

1234567890

**Вывод**

Благодаря данной лабораторной работе, я узнала, что такое процессы и потоки, научилась работать с ними на базе операционной системы Ubuntu, а именно создавать дочерние потоки с помощью системного вызова fork() , осуществлять взаимодействие потоков, используя pipe() , перенаправлять один поток на другой, с помощью dup2() . Разобралась с pid — идентификатором процесса операционной системы (его можно использовать для отслеживания процессов и их управления) и файловыми дескрипторами fd[0] – чтение и fd[1] – запись.

Работа с потоками улучшает производитьльность программы, поэтому полученные навыки будут полезны в дальнейших работах.