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: Головенко Анатолий Валерьевич

Группа: М8О–212Б–22

Вариант: 6

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2020.

**Постановка задачи**

Цель работы

Целью является приобретение практических навыков в:

* Управлении процессами в ОС
* Обеспечение обмена данными между процессами

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.

6 вариант) В файле записаны команды вида: «число число число<endline>». Дочерний процесс считает их сумму и выводит результат в стандартный поток вывода. Числа имеют тип int. Количество чисел может быть произвольным.

**Общие сведения о программе**

Программа компилируется из файла parent.cpp. Также есть программа child.cpp, которая вызывается с помощью exec. Сборка происходит при помощи системы Cmake. В программе используются следующие системные вызовы:

1. **fork** - создаёт дочерний процесс
2. execlp - вызывает исполняемый файл из программы,
3. pipe - создаёт пайп, позволяющий производить обмен данными между процессами
4. dup2 - позволяет изменить стандартный поток ввода/вывода
5. open - открытие файла
6. close - закрытие файла

**Общий метод и алгоритм решения**.

Для реализации поставленной задачи необходимо:

1. Изучить принципы работы fork, exec, pipe, dup2.
2. Написать основную программу parent.cpp, в которой будут приходить вызовы системных функций
3. Написать программу child.cpp, которая будет запускать

**Основные файлы программы**

**parent.cpp:**

#include <iostream>

#include <unistd.h>

#include <string>

#include <fstream>

#include <fcntl.h>

using namespace std;

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

{

string file\_name;

char symbol;

if (read(STDIN\_FILENO, &symbol, sizeof(char)) == -1)

{

return 7;

}

while (symbol != '\n')

{

file\_name += symbol;

if (read(STDIN\_FILENO, &symbol, sizeof(char)) == -1)

{

return 7;

}

}

int file = open(file\_name.c\_str(), O\_RDONLY | O\_CREAT, 0777);

if (file == -1)

{

return -1;

}

int fd[2]; // fd[0] - read; fd[1] - write

// Creating pipe and checking that it was created

if (pipe(fd) == -1)

{

return 1;

}

// Forking the process

int id = fork();

if (id == -1) // fork error

{

return 2;

}

if (id == 0)

{ // child process

dup2(file, 0);

if (dup2(fd[1], 1) == -1)

{

return 5;

}

close(fd[0]);

execlp("./child", "./child", NULL);

return 3;

}

else

{ // parent process

close(fd[1]);

int answer = 0;

string final = "";

if (read(fd[0], &answer, sizeof(int)) == -1)

{

return 7;

}

final = to\_string(answer);

write(STDOUT\_FILENO, &final, final.length());

close(fd[0]);

close(file);

}

return 0;

}

**child.cpp:**

#include <iostream>

#include <unistd.h>

#include <string>

#include <fstream>

using namespace std;

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

{

int count = 0;

char symbol;

string num;

if (read(STDIN\_FILENO, &symbol, sizeof(char)) == -1)

{

return -1;

}

while (symbol != '\n')

{

if (symbol != ' ')

{

num += symbol;

}

else

{

count += stoi(num);

num = "";

}

read(STDIN\_FILENO, &symbol, sizeof(char));

}

try

{

count += stoi(num);

}

catch (invalid\_argument &ex)

{

count = 0;

}

write(STDOUT\_FILENO, &count, sizeof(int));

return 0;

}

**CMakeLists.cpp:**

cmake\_minimum\_required(VERSION 3.10)

project(Parallel\_processes)

set(CMAKE\_CXX\_STANDARD 20)

add\_executable(parent parent.cpp)

add\_executable(child child.cpp)

**Пример работы**

1. Пустой файл

test\_1.txt:

anatolii@MacBook-Pro-Anatolii ~/D/O/h/build (main)> /Users/anatolii/ Desktop/Oper\_Syst/hometask\_1/build/parent

test\_1.txt

0

anatolii@MacBook-Pro-Anatolii ~/D/O/h/build (main)>

2) Файл с одним числом

test\_2.txt:

45

anatolii@MacBook-Pro-Anatolii ~/D/O/h/build (main)> /Users/anatolii/ Desktop/Oper\_Syst/hometask\_1/build/parent

test\_2.txt

45

anatolii@MacBook-Pro-Anatolii ~/D/O/h/build (main)>

3) Файл с несколькими числами:

test\_2.txt:

1 234 6 89 0 76 12 7 9 13 23 9

anatolii@MacBook-Pro-Anatolii ~/D/O/h/build (main)> /Users/anatolii/ Desktop/Oper\_Syst/hometask\_1/build/parent

test\_3.txt

479

anatolii@MacBook-Pro-Anatolii ~/D/O/h/build (main)>

**Вывод**

Я изучил основные системные вызовы для работы с процессами и обмена данными между ними. В моём варианте происходит разделение на родительский и дочерний процессы при помощи fork. В родительском процессе я считываю название файла с помощью read, открываю его и делаю чтение из файла - стандартным потоком ввода для дочернего процесса. Далее создаю pipe который послужит для дочернего потока стандартным потоком вывода. После этого командой execlp вызываю исполняемый файл. В нём происходит считывание и сложение чисел. Итоговая сумма передаётся в родительский процесс, считывается там из папа и выводит в консоль ответ, сумму всех чисел.