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: Епифанов Евгений Валерьевич

Группа: М8О–212Б–22

Вариант: 4

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2020.

**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 одной̆ из двух операционных систем. В результате работы программа (основной̆ 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 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 со 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.

4 вариант) Пользователь вводит команды вида: «число число число». Далее эти числа передаются от родительского процесса в дочерний. Дочерний процесс производит деление 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. Также используется файл child.cpp, который запускается через exec. Обе программы собираются при помощи системы сборки CMake. В программе используются следующие системные вызовы:

1. **fork** - создает дочерний процесс.
2. **exec** - позволяет запустить исполняемый файл изнутри программы.
3. **pipe** - создает канал для передачи данных между процессами.
4. **dup2** - переназначение файлового дескриптора.
5. **open** - открытие файла.
6. **сlose** - закрытие файла.
7. **wait** - ожидание дочерних процессов.

**Общий метод и алгоритм решения**.

Для выполнения лабораторной работы необходимо:

1. Изучить принципы работы fork, exec, pipe.
2. Написать основную программу main.cpp, в которой будет осуществляться вызов системных функций.
3. Написать программу child.cpp, которая будет запускаться из родительского процесса при помощи exec.

**Основные файлы программы**

**main.cpp:**

#include <unistd.h> // for fork() and pipe

#include <string>

int main()

{

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

if (pipe(fd) == -1) return -1;

int id = fork(); // create a process

if (id == -1) // error

{

return -1;

}

else if (id == 0) // child

{

close(fd[1]);

dup2(fd[0], STDIN\_FILENO);

execlp("./child", NULL);

return 0;

}

else // parent (id > 0)

{

close(fd[0]);

std::string number = "";

char c;

if (read(STDIN\_FILENO, &c, sizeof(char)) == -1) return -1;

while (c != '\n')

{

if (c == ' ')

{

if (write(fd[1], &number, sizeof(number)) == -1) return -1;

number = "";

}

else

{

number += c;

}

if (read(STDIN\_FILENO, &c, sizeof(char)) == -1) return -1;

}

if (number != "")

{

if (write(fd[1], &number, sizeof(number)) == -1) return -1;

}

close(fd[1]);

int status;

waitpid(0, &status, 0);

if (status != 0) return -1;

}

return 0;

}

**child.cpp:**

#include <fcntl.h> // for open

#include <unistd.h> // for read() and write()

#include <string>

int main()

{

int output;

output = open("output.txt", O\_WRONLY, 0);

std::string parentdata;

float divisible = 0;

bool notempty = 0;

while (read(STDIN\_FILENO, &parentdata, sizeof(parentdata)) != 0) {

if (divisible == 0)

{

notempty = 1;

divisible = std::stof(parentdata);

}

else

{

if (std::stof(parentdata) == 0) exit(-1);

divisible /= std::stof(parentdata);

}

}

if (!notempty) exit(-1);

std::string div\_str = std::to\_string(divisible);

div\_str = div\_str.substr(0, 8);

char answer[100];

sprintf(answer, "The answer is: %f", divisible);

if (write(output, &answer, 15 + div\_str.length()) == -1) {

return -1;

}

}

**CMakeLists.txt:**

cmake\_minimum\_required(VERSION 3.10)

project(Child\_processes\_with\_pipes)

set (CMAKE\_CXX\_STANDARD 20)

add\_executable(main main.cpp)

add\_executable(child child.cpp)

**Пример работы**

1. **Пустой ввод:**

**evgeniy@EP-iMac build % /Users/evgeniy/Desktop/OS\_Labs/Lab1/src/build/main**

**evgeniy@EP-iMac build %**

**output.txt:**

1. **Ввод одного числа:**

**evgeniy@EP-iMac build % /Users/evgeniy/Desktop/OS\_Labs/Lab1/src/build/main**

6.784

**evgeniy@EP-iMac build %**

**output.txt:**

The answer is: 6.784000

1. **Ввод нескольких чисел**

**evgeniy@EP-iMac build % /Users/evgeniy/Desktop/OS\_Labs/Lab1/src/build/main**

123.123 14.88 2.7

**evgeniy@EP-iMac build %**

**output.txt:**

The answer is: 3.064591

1. **Ввод большого числа**

**evgeniy@EP-iMac build % /Users/evgeniy/Desktop/OS\_Labs/Lab1/src/build/main**

78956.45 9.123456

**evgeniy@EP-iMac build %**

**output.txt:**

The answer is: 8654.226562

**Вывод**

Мной были изучены и использованы основные системные вызовы для работы с процессами и обмена данными между ними. Пришлось изучить принцип работы файловых дескрипторов, чтобы научиться перенаправлять данные из файлов на потоки ввода и вывода при помощи dup2. Для выполнения моего варианта лабораторной работы потребовалось разделить процессы на родительский и дочерний, это было сделано с помощью fork. Родительский процесс читает данные из стандартного ввода и записывает их в канал, а дочерний процесс перенаправляет эти данные с канала на свой стандартный поток ввода. Дочерний процесс запускается через exec, в нем полученные данные используются для вычисления частного. Далее ответ записывается в выходной файл при помощи open. При выполнении работы я столкнулся с некоторыми проблемами, но все они были вызваны особенностями работы программы VSCode. После внесения изменений в код либо ввод исходных данных, либо ввод данных в файл отрабатывали некорректно. Проблема была решена дополнительной перезагрузкой окна приложения.