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: Салихов Тимур Русланович

Группа: М8О–212Б–22

Вариант: 1

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 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 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 принеобходимости 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.

1 вариант) Пользователь вводит команды вида: «число число число<endline>». Далее эти числа передаются от родительского процесса в дочерниий. Дочерний̆ процесс считает их сумму и выводит её в файл. Числа имеют тип int. Количество чисел может быть произвольным.

**Общие сведения о программе**

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

1. **fork** - создает дочерний процесс, повторяющий программу родительского.
2. **exec** - 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 calculator.cpp, которая будет запускаться из дочернего процесса при помощи exec

**Основные файлы программы**

**main.cpp:**

#include <unistd.h>

#include <string>

auto main() -> int {

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;

} else if (id == 0) { // child process

close(fd[1]);

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

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

return 3;

} else { // parent process

close(fd[0]);

std::string number = "";

char c;

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

return 4;

}

while (c != '\n') {

if (c == ' ') {

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

return 5;

}

number = "";

} else {

number += c;

}

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

return 4;

}

}

if (number != "") {

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

return 5;

}

}

close(fd[1]);

}

return 0;

}

**calculator.cpp:**

#include <fcntl.h>

#include <unistd.h>

#include <string>

auto main() -> int {

int outputd;

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

std::string received;

int sum = 0;

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

sum += std::stoi(received);

}

std::string sum\_str = std::to\_string(sum);

char output[100];

sprintf(output, "The sum is: %d", sum);

if (write(outputd, &output, 12 + sum\_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(calculator calculator.cpp)

**Пример работы**

1. **Пустой ввод:**

**timursalihov@MBP-Timur-2 ~/O/L/build (main)>** /Users/timursalihov/OperatingSystems/LW1/build/main

**timursalihov@MBP-Timur-2 ~/O/L/build (main)>**

**output.txt:**

The sum is: 0

1. **Ввод одного числа:**

**timursalihov@MBP-Timur-2 ~/O/L/build (main)>** /Users/timursalihov/OperatingSystems/LW1/build/main

42

**timursalihov@MBP-Timur-2 ~/O/L/build (main)>**

**output.txt:**

The sum is: 42

1. **Ввод нескольких чисел**

**timursalihov@MBP-Timur-2 ~/O/L/build (main)>** /Users/timursalihov/OperatingSystems/LW1/build/main

1 43 213 12 7

**timursalihov@MBP-Timur-2 ~/O/L/build (main)>**

**output.txt:**

The sum is: 276

1. **Ввод большого числа**

**timursalihov@MBP-Timur-2 ~/O/L/build (main)>** /Users/timursalihov/OperatingSystems/LW1/build/main

167437345

**timursalihov@MBP-Timur-2 ~/O/L/build (main)>**

**output.txt:**

The sum is: 167437345

**Вывод**

Мной изучены основные системные вызовы для работы с процессами и обмена данными между ними. В моем варианте происходит разделение на родительский и дочерний процессы при помощи fork. Родительский процесс читает данные из стандартного ввода и записывает их в пайп. Дочерний процесс перенаправляет эти данные на свой стандартный поток ввода, после чего запускает исполняемый файл при помощи exec, в котором происходит чтение чисел полученных через пайп и подсчет суммы. Далее сумма записывается в выходной файл при помощи open. В процессе выполнения работы я столкнулся с определенными трудностями, в основном связанными с использованием функций read и write. Моей ошибкой было то, что я пытался записывать бинарные данные, а читал текстовые из-за этого мой пайп работал не так как ожидалось, однако мне удалось решить эту проблему. Также пришлось изучить как работают файловые дескрипторы, чтобы научиться перенаправлять данные из файлов на потоки ввода/вывода при помощи dup2.