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: Басова Татьяна Валентиновна

Группа: М8О–212Б–22

Вариант: 1

Преподаватель: Соколов Андрей Алексеевич

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 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.

1 вариант) Пользователь вводит команды вида: «число число число<endline>». Далее эти числа передаются от родительского процесса в дочерний. Дочерний 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. Также используется файл calc.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 calc.cpp, которая будет запускаться из дочернего процесса при помощи exec.

**Основные файлы программы**

**main.cpp:**

#include <sys/wait.h>

#include <unistd.h>

#include <string>

auto main() -> int {

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

// create pipe and check if it was created

if (pipe(fd) == -1) {

return 1;

}

// create the process

int id = fork();

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

return 2;

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

close(fd[1]);

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

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

return 3;

} else { // parent

close(fd[0]);

char c;

while (read(STDIN\_FILENO, &c, sizeof(char)) > 0) {

if (c == '\n') {

break; // stop reading if \n

}

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

return 5;

}

}

close(fd[1]);

int status;

waitpid(0, &status, 0); // wait child process finish

if (status != 0)

return 6;

}

return 0;

}

**child.cpp:**

#include <fcntl.h>

#include <unistd.h>

#include <string>

auto main() -> int {

int outputd;

outputd = open("output.txt", O\_RDWR | O\_CREAT, 0);

std::string s = "0";

char ch;

int sum = 0;

while (read(STDIN\_FILENO, &ch, ch!='\n')) {

if ( ch == ' '){

sum += std::stoi(s);

s = "";

}

else

s += ch;

}

sum+=std::stoi(s);

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

char output[100];

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

if (write(outputd, &output, 13 + sum\_str.length()) == -1) {

return 1;

}

}

**CMakeLists.txt:**

cmake\_minimum\_required(VERSION 3.10)

project(OS\_1)

set (CMAKE\_CXX\_STANDARD 17)

add\_executable(main main.cpp)

add\_executable(calc calc.cpp)

**Пример работы**

1. **Пустой ввод:**

**tanya@tanya-S431:~/Рабочий стол/os\_1/build$ ./main**

**output.txt:**

**The sum is: 0**

1. **Ввод одного числа:**

**tanya@tanya-S431:~/Рабочий стол/os\_1/build$ ./main**

**4**

**output.txt:**

**The sum is: 4**

1. **Ввод нескольких чисел**

**tanya@tanya-S431:~/Рабочий стол/os\_1/build$ ./main**

**1 2 3 4**

**output.txt:**

**The sum is: 10**

**4) Ввод большого числа**

**tanya@tanya-S431:~/Рабочий стол/os\_1/build$ ./main**

**11234567**

**output.txt:**

**The sum is: 11234567**

**Вывод**

Мной были изучены и использованы основные системные вызовы для работы с процессами и обмена данными между ними. Для выполнения задания необходимо было изучить принцип работы файловых дескрипторов и dup2. 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 столкнулась с некоторыми проблемами, связанными с вводом данных, но были решены с помощью посимвольного считывания данных из консоли.