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: Муратов Артём Алексеевич

Группа: М8О–212Б–20

Вариант: 9

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2023.

**Постановка задачи**

## Цель работы

Целью является приобретение практических навыков в:

* Управление процессами в ОС
* Обеспечение обмена данными между процессами посредством труб

## 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.

9 вариант) В файле записаны команды вида:«число число число <endline>». Дочерний процесс производит деление первого числа команда, на последующие числа в команде, а результат выводит в стандартный поток вывода. 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.c. Для сборки используется CMake. Также используется заголовочные файлы:

#include <string.h>

#include <unistd.h>

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <fcntl.h>

#include <sys/wait.h> .

В программе используются следующие системные вызовы:

1. **read** — считывание с консоли.
2. **write** — запись в консоль.
3. **fork** — создание дочернего процесса, который повторяет программу родительского процесса.
4. **pipe** — создание одностороннюю трубу для передачи данных между процессами.
5. **exit** — завершение программы.
6. **open** — открытие файла.
7. **close** — закрытие файла.
8. **dup2** — переназначение файлового дескриптора.
9. **execvp** — запуск исполняемого файла изнутри другой программы.
10. **waitpid** — ожидание завершения дочернего процесса.

**Общий метод и алгоритм решения**.

Для реализации поставленной задачи необходимо:

1. Изучить принципы работы функции fork(), pipe().
2. Создать child.c и main.c.
3. Реализовать в main.c:
   1. Создание двух процессов и общение между программами child.c и main.c с помощью двух односторонних труб.
   2. Открытие заданного файла, считывание чисел из него и отправка считанных чисел в child.c.
   3. Прием и вывод ответа из child.c.
4. Реализовать в child.c:
   1. Считывание из main.c строки чисел с помощью read.
   2. Функцию,решающую задачу варианта.
   3. Передача ответа в main.c с помощью write.

**Основные файлы программы**

**Main.c**

#include <string.h>

#include <unistd.h>

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <fcntl.h>

#include <sys/wait.h>

int new\_proccess() {

pid\_t pid = fork();

return pid;

}

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

int fd1[2];

int fd2[2];

if( pipe(fd1) == -1 || pipe(fd2) == -1) {

write(STDERR\_FILENO, "Something wrong with pipe", 25);

exit(-1);

}

pid\_t pid = new\_proccess();

if( pid == -1 ){

write(STDERR\_FILENO, "Something wrong with fork",25);

exit(-1);

} else if ( 0 == pid ) {

close(fd1[1]);

dup2(fd1[0],STDIN\_FILENO);

close(fd2[0]);

dup2(fd2[1],STDOUT\_FILENO);

execvp("./child", argv);

}

else {

char file\_name[128];

read(STDIN\_FILENO,file\_name, 128);

int pos = strlen(file\_name) - 1;

if (file\_name[pos] == '\n') {

file\_name[pos] = '\0';

}

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

char numbers[128];

if (file != -1) {

read(file, numbers, 128);

}

else {

write(STDERR\_FILENO, "The file was not open\n", 23);

exit(2);

}

close(fd1[0]);

write(fd1[1], numbers, 128 );

close(fd1[1]);

waitpid(pid, NULL, -1);

char answer[128] = " ";

close(fd2[1]);

read(fd2[0], answer, 128);

write(STDOUT\_FILENO, answer, 128);

write(STDOUT\_FILENO, "\n", 1);

close(fd2[0]);

}

return 0;

}

**Сhild.c**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

float solving\_func(char \* nums) {

float answer = strtof(nums, NULL);

float divider = 0;

int flag = 0;

for (int i = 0; i < strlen(nums)+1; ++i) {

if ((flag == 0) && ((nums[i] == ' ') || (nums[i] == '\0'))) {

flag = 1;

}

else if ((flag == 1) && (nums[i] != ' ') && (nums[i] != '\0')) {

divider = divider\*10 + ((int)nums[i] - 48);

}

else if ((flag == 1) && ((nums[i] == ' ') || (nums[i] == '\0'))) {

if (divider == 0) {

write(STDERR\_FILENO, "Division by 0",14);

exit(1);

}

answer = answer / divider;

divider = 0;

}

}

return answer;

}

int main() {

char child\_numbers[128];

float answer;

read(STDIN\_FILENO,child\_numbers, 128);

int pos = strlen(child\_numbers) - 1;

if ( child\_numbers[pos] == '\n') {

child\_numbers[pos]='\0';

}

answer = solving\_func(child\_numbers);

char answer\_char[128] = " ";

gcvt(answer, 12, answer\_char);

write(STDOUT\_FILENO, answer\_char, 128);

return 0;

}

**CMakeLists.txt**

cmake\_minimum\_required(VERSION 3.10)

project(lab\_os\_1)

set(CMAKE\_C\_STANDARD 17)

set(CMAKE\_C\_STANDARD\_REQUIRED ON)

add\_executable(child child.c)

add\_executable(main main.c)

**Пример работы**

**1.** 10 2 5

virtbox@virtbox-VirtualBox:~/OS\_labs/lab\_os\_1/build$ ./main

/home/virtbox/OS\_labs/lab\_os\_1/tests/test\_1.txt

1

**2.** 120 2 5 4 3

virtbox@virtbox-VirtualBox:~/OS\_labs/lab\_os\_1/build$ ./main

/home/virtbox/OS\_labs/lab\_os\_1/tests/test\_2.txt

1

**3.** 2 0

virtbox@virtbox-VirtualBox:~/OS\_labs/lab\_os\_1/build$ ./main

/home/virtbox/OS\_labs/lab\_os\_1/tests/test\_3.txt

Division by 0

**4.** 127 34 23 3 5

virtbox@virtbox-VirtualBox:~/OS\_labs/lab\_os\_1/build$ ./main

/home/virtbox/OS\_labs/lab\_os\_1/tests/test\_4.txt

0.0108269397169

**5.** 0

virtbox@virtbox-VirtualBox:~/OS\_labs/lab\_os\_1/build$ ./main

/home/virtbox/OS\_labs/lab\_os\_1/tests/test\_5.txt

0

**6.** 0 1234 5678 90

virtbox@virtbox-VirtualBox:~/OS\_labs/lab\_os\_1/build$ ./main

/home/virtbox/OS\_labs/lab\_os\_1/tests/test\_6.txt

0

**Вывод**

В процессе выполнения данной лаборатоной я изучил основные системные вызовы для работы с процессами, обмена информацией между ними, ввода и вывода данных программы.