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 ПРОЦЕССОВ**

Студент: Куценко Максим Дмитриевич

Группа: М8О–212Б-22

Вариант: 3

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2023.

**Постановка задачи**

**Цель работы**

Приобретение практических навыков в: 

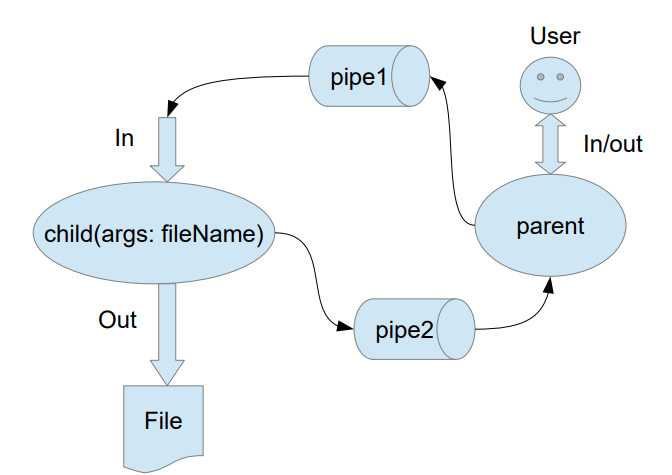
* Управление процессами в ОС 
* Обеспечение обмена данных между процессами посредством каналов

**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.

*Вариант задания 3:*

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.

Пользователь вводит команды вида: «число число число<endline>» 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 int. Количество чисел может быть произвольным.

**Общие сведения о программе**

При помощи Makefile собираются программы — main и divider. В main происходит создание pipe и процесса при помощи fork, работа с дескрипторами, а также чтение из пользовательского ввода и вывод результата. В divider происходит разделение ввода, вычисления и запись в файл.

Программа компилируется из файла main.c. Также используется заголовочные файлы:"stdlib.h", "stdio.h", "unistd.h", "fcntl.h", "string.h", "sys/wait.h".

В программе используются следующие системные вызовы:

1. **fork** – создаёт дочерний процесс.
2. **exec –** запуск исполняемого файла.
3. **pipe** — создание канала для передачи данных между процессами.
4. **dup2** — работа с дескрипторами.
5. **open** — открытие файла.
6. **wait** — ожидание окончания работы дочернего процесса родительским. Получает статус выхода.

**Общий метод и алгоритм решения**.

Для реализации поставленной задачи необходимо:

1. Изучить принципы взаимодействия процессов, работы с дескрипторами и использование read/write.
2. Написать main.c. Там в char\* считываются две строки и передаются по pipe в исполняемый файл divider. Потом, если программа завершилась успешно, выводим полученный от дочернего прооцесса по другому pipe результат.
3. Написать divider.c. Там при помощи strtok делим строки, получая имя файла и числа для деления. Создаём файл с данным именем и заносим в него ответ. Затем делим, пока не встречаем 0.

**Основные файлы программы**

**CmakeLists.txt**

cmake\_minimum\_required(VERSION 3.0)

project(Forks-pipes)

add\_executable(main main.c)

add\_executable(divider divider.c)

**main.c:**

#include "stdlib.h"

#include "stdio.h"

#include "unistd.h"

#include "sys/wait.h"

int main() {

int pipe\_fd[2]; // creating pipe to child process; pipe\_fd[0] - for reading, pipe\_fd[1] - for writing

int pipe\_bw[2]; // creating pipe from executed program to parent process

if ((pipe(pipe\_fd) == -1) || (pipe(pipe\_bw)==-1)) { // creating pipes

perror("Pipe creation error");

return 1;

}

pid\_t pid = fork(); // creating child process

if (pid == -1) {

perror("Child process creation error");

return 2;

} else if (pid == 0) { // going in child process

close(pipe\_fd[1]); // closing the descriptor of forward pipe to write

close(pipe\_bw[0]); // closing the descriptor of backward pipe to read

dup2(pipe\_fd[0], STDIN\_FILENO); // duplicating descriptors

dup2(pipe\_bw[1], STDOUT\_FILENO);

execl("divider", "divider", NULL); // executing program to divide the numbers

perror("Exec error");

return 3;

} else { // going in parent process

close(pipe\_fd[0]); // closing the descriptor of forward pipe to read

close(pipe\_bw[1]); // closing the descriptor of backward pipe to write

char\* input = 0; // creating char\* pointer for reading input

int len = 0;

int counter = 0; // creating variable for counting lines of input

while(counter != 2) {

char c;

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

perror("Input reading error");

return 4;

}

input = realloc(input, sizeof(char)\*(++len));

input[len-1]=c;

if (c == '\n') {

counter += 1;

}

}

input[len-1]=' '; // doing this for strtok later

input = realloc(input, sizeof(char)\*(++len));

input[len-1]='\0';

if ((write(pipe\_fd[1], &len, sizeof(int)) == -1) || (write(pipe\_fd[1], input, sizeof(char)\*len) == -1)) { // writing to forward pipe

perror("Writing to forward pipe error");

return 5;

}

close(pipe\_fd[1]); // closing the descriptor to write because it is no longer needed

int status;

wait(&status); // waiting for child process to finish

if (WIFEXITED(status) && WEXITSTATUS(status)!=0) { // checking for error in child

return(6);

}

int answ\_len;

if (read(pipe\_bw[0], &answ\_len, sizeof(int)) == -1) { // reading answer length from pipe

perror("Backward pipe reading error");

return 4;

}

char\* answer = malloc(sizeof(char)\*answ\_len);

if (read(pipe\_bw[0], &answer, answ\_len\*sizeof(char)) == -1) { // reading answer from pipe

perror("Backward pipe reading error");

return 4;

}

if ((write(STDOUT\_FILENO, "Here is the answer: ", 20\*sizeof(char)) == -1) || (write(STDOUT\_FILENO, &answer, answ\_len\*sizeof(char)) == -1) || (write(STDOUT\_FILENO, "\n", sizeof(char)) == -1)) { // writing answer to output

perror("Writing to stdout error");

return 5;

}

}

return 0;

}

**divider.c**

#include "stdlib.h"

#include "stdio.h"

#include "unistd.h"

#include "fcntl.h"

#include "string.h"

int main() {

int len;

if (read(STDIN\_FILENO, &len, sizeof(int)) == -1) { // reading length of data

perror("Forward pipe reading error");

\_exit(-1);

}

char\* data = malloc(sizeof(char)\*len);

if (read(STDIN\_FILENO, data, len) == -1) { // reading whole data

perror("Forward pipe reading error");

\_exit(-1);

}

char\* var = strtok(data, "\n"); // getting filename

var = strtok(NULL, "\n"); // getting array with all the numbers

var = strtok(var, " "); // getting first number as char\*

if ((var == NULL) || ((var[0]!='0') && (atoi(var)==0))) { // checking whether data is correct

perror("data format error");

\_exit(-2);

}

int file = open(data, O\_WRONLY | O\_CREAT, 0777);

if (file == -1) {

perror("File reading error");

\_exit(-3);

}

int result=atoi(var); // getting first number as int

var = strtok(NULL, " ");

while (var != NULL) { // dividing

printf("%d ", result);

int del = atoi(var);

if (del == 0) { // checking for 0 div

perror("Zero divider error");

\_exit(-4);

}

result/=del;

var = strtok(NULL, " ");

}

int num\_len=1, num = result/10; // creating string version of result

while (num>0) {

num/=10;

num\_len++;

}

char res\_str[num\_len];

sprintf(res\_str, "%d", result);

if ((write(file, "Here is the answer: ", 20\*sizeof(char)) == -1) || (write(file, &res\_str, num\_len\*sizeof(char)) == -1)) { // writing answer to the file

perror("Writing to file error");

\_exit(-5);

}

if ((write(STDOUT\_FILENO, &num\_len, sizeof(int)) == -1) || (write(STDOUT\_FILENO, &res\_str, num\_len\*sizeof(char)) == -1)) { // writing to backward pipe

perror("Writing to backward pipe error");

\_exit(-5);

}

return 0;

}

**Пример работы**

**[Maxim@HONOR-MB15 build]$** "/home/Maxim/Документы/Лабы\_3\_сем/OS/OS\_LR\_1/src/build/main"

output.txt

1000 5 2 4

Here is the answer: 25

**[Maxim@HONOR-MB15 build]$** "/home/Maxim/Документы/Лабы\_3\_сем/OS/OS\_LR\_1/src/build/main"

output.txt

100000 2 3 4 5 6

Here is the answer: 138

**[Maxim@HONOR-MB15 build]$** "/home/Maxim/Документы/Лабы\_3\_сем/OS/OS\_LR\_1/src/build/main"

output.txt

30 5 3 2

Here is the answer: 1

**[Maxim@HONOR-MB15 build]$** "/home/Maxim/Документы/Лабы\_3\_сем/OS/OS\_LR\_1/src/build/main"

text.txt

123456789 9 8 7 6 5 4 3 2 1

Here is the answer: 340

**[Maxim@HONOR-MB15 build]$** "/home/Maxim/Документы/Лабы\_3\_сем/OS/OS\_LR\_1/src/build/main"

output.txt

2

Here is the answer: 2

**[Maxim@HONOR-MB15 build]$** "/home/Maxim/Документы/Лабы\_3\_сем/OS/OS\_LR\_1/src/build/main"

output.txt

0 1 2 3

Here is the answer: 0

* **[Maxim@HONOR-MB15 build]$** "/home/Maxim/Документы/Лабы\_3\_сем/OS/OS\_LR\_1/src/build/main"

output.txt

100 2 3 0

zero divider error: Success

* **[Maxim@HONOR-MB15 build]$** "/home/Maxim/Документы/Лабы\_3\_сем/OS/OS\_LR\_1/src/build/main"

1 2 3 4 5

output.txt

data format error: Success

* **[Maxim@HONOR-MB15 build]$** "/home/Maxim/Документы/Лабы\_3\_сем/OS/OS\_LR\_1/src/build/main"

1 2 3 4 5

\n

data format error: Success

* **[Maxim@HONOR-MB15 build]$** "/home/Maxim/Документы/Лабы\_3\_сем/OS/OS\_LR\_1/src/build/main"

output.txt

\n

data format error: Success

* **[Maxim@HONOR-MB15 build]$** "/home/Maxim/Документы/Лабы\_3\_сем/OS/OS\_LR\_1/src/build/main"

\n

\n

data format error: Success

**Вывод**

Изучил, как устроены pipe и как они могут быть полезны при взаимодействии между процессами, условия для которого были созданы при помощи команд fork и exec. Выяснил, что из себя представляют дескрипторы и как на практике работает вывод в языке Си.

Также в процессе работы над программой определил, что лабораторные работы желательно писать, используя исключительно инструментарий языка Си, потому что системные вызовы ориентированы именно на него.

При записи в файл и вывод понял, что числа печатаются для пользователя в текстовом виде, а не в бинарном. Бинарные же символы в файлах формата .txt и в терминале отображаются некорректно.

Встретил ошибку, связанную со считыванием символов в char\* из пользовательского ввода и выяснил важность использования „\0“.