Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работ №2 по курсу**

**«Операционные системы»**

**Управление процессами в ОС**

Студент: Петрухин Дмитрий Олегович

Группа: М80 – 201Б-18

Вариант: 13

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2019

**Содержание**

1. Постановка задачи
2. Общие сведения о программе
3. Общий метод и алгоритм решения
4. Основные файлы программы
5. Тестирование
6. Демонстрация работы программы
7. Вывод

**Постановка задачи**.

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.

**Задание согласно 13 варианту:**

Родительский процесс считывает матрицы из стандартного ввода, дочерний процесс вычисляет среднеарифметическое матриц и передает их родительского процессу. Родительский процесс печатает результат в стандартный выход.

**Общие сведения о программе**

Программа компилируется из одного файла lab.c. В данном файле используются заголовочные файлы sys/types.h, unistd.h, stdlib.h, string.h. В программе используются следующие системные вызовы:

1. **read/write** – предназначены для осуществления потоковых операций ввода (чтения) и вывода (записи) информации над каналами связи, описываемыми файловыми дескрипторами, т.е. для pipe, файлов и для потокового ввода.
2. **pipe** – для создания однонаправленного канала, через который могут общаться два процесса. При нормальном завершении вызова в первый элемент массива(аргумент pipe) – fd[0] – будет занесен файловый дескриптор, соответствующий выходному потоку данных pip’а и позволяющий выполнять только операцию чтения, а во второй элемент массива – fd[1] – будет занесен файловый дескриптор, соответствующий входному потоку данных и позволяющий выполнять только операцию записи. Системный вызов возвращает значение 0 при нормальном завершении и значение -1 при возникновении ошибок.
3. **fork** – системный вызов для порождения нового процесса. Процесс, который инициировал системный вызов fork, принято называть родительским процессом (parent process). Вновь порожденный процесс принято называть процессом-ребенком (child process). Процесс-ребенок является почти полной копией родительского процесса. У порожденного процесса по сравнению с родительским изменяются значения следующих параметров: PID, PPID. При однократном системном вызове возврат из него может произойти дважды: один раз в родительском процессе, а второй раз в порожденном процессе. Если создание нового процесса произошло успешно, то в порожденном процессе системный вызов вернет значение 0, а в родительском процессе – положительное значение, равное идентификатору процесса-ребенка. Если создать новый процесс не удалось, то системный вызов вернет в инициировавший его процесс отрицательное значение.

**Общий метод и алгоритм решения**.

Для реализации поставленной задачи необходимо:

1. Используя системный вызов pipe создать канал, по которому будут обмениваться данными два процесса.
2. Используя системный вызов fork создать дочерний процесс.
3. В родительском процессе считывать данные cо стандартного потока и поэлементно их отправлять в первый pipe в цикле for с помощью системного вызова write.
4. Пока родительский процесс на записал данные в канал. Дочерний процесс ждет. И как только родительский процесс записал данные в канал дочерний процесс считывает их, производит вычисления и возвращает результат родительскому процессу с помощью второго pipe. Родительский процесс ожидает, пока придут данные во второй pipe, а затем этот процесс их считает с помощью системного вызова read.
5. Родительский процесс выводит результат используя write.

**Основные файлы программы**.

**Файл main.c**

#include <sys/types.h>

#include <unistd.h> // for fileno, read/write

#include <stdlib.h> //for atoi/exit

#include <string.h>

void ftoa(float n, char s[]) //float\_to\_char

{

int i, sign, k, d;

if ((sign = n) < 0) /\* записываем знак \*/

n = -n; /\* делаем n положительным числом \*/

i = 0;

k = n;

n = n - k;

while (n - (int)n)

n\*=10;

d = (int)n;

do { /\* генерируем цифры после точки в обратном порядке \*/

s[i++] = d % 10 + '0'; /\* берем следующую цифру \*/

} while ((d /= 10) > 0); /\* удаляем \*/

s[i++] = '.';

do {

s[i++] = k % 10 + '0';

} while ((k /= 10) > 0);

if (sign < 0)

s[i++] = '-';

s[i] = '\0';

reverse(s);

}

void reverse(char s[])

{

int i, j;

char c;

for (i = 0, j = strlen(s)-1; i<j; i++, j--) {

c = s[i];

s[i] = s[j];

s[j] = c;

}

}

int main(){

char num1[10], num2[10], s;

int trub1[2];

int trub2[2];

double x=0, summ;

int i, m, n;

pid\_t p;

if(pipe(trub1)<0) return 1;

if(pipe(trub2)<0) return 1;

p = fork(); // create two process

if(p == -1){ // everything is bad, the fork is not working

return -1;

} else if (p == 0){

read(trub1[0], num2, sizeof(num2));

m = atoi(num2);

read(trub1[0], num2, sizeof(num2));

n = atof(num2);

// for first matrix

for(i=0;i<m\*n;i++){

read(trub1[0], num2, sizeof(num2));

x +=atof(num2);

}

summ = x / (m\*n); // means value first matrix

x = 0;

ftoa(summ,num2);

write(trub2[1], &num2, sizeof(num2));

//for second matrix

for(i=0;i<m\*n;i++){

read(trub1[0], num2, sizeof(num2));

x +=atof(num2);

}

summ = x / (m\*n);

ftoa(summ,num2);

write(trub2[1], &num2, sizeof(num2));

//child process

} else {

read(STDIN\_FILENO,num1,10);

m = atoi(num1);

if (!atoi(num1)){

write(STDOUT\_FILENO, "m should not be zero \n", sizeof "m should not be zero\n" - 1);

write(STDOUT\_FILENO, "enter number m\n", sizeof "enter number m\n" - 1);

read(STDIN\_FILENO,num1,10);

m = atoi(num1);

}

write(trub1[1], &num1, sizeof(num1));

write(STDOUT\_FILENO, "enter n not equal to zero\n", sizeof "enter n not equal to zero\n" - 1);

read(STDIN\_FILENO,num1,10);

n = atoi(num1);

write(trub1[1], &num1, sizeof(num1));

for(i=0;i<m\*n;i++) {

read(STDIN\_FILENO,num1,10);

write(trub1[1], &num1, sizeof(num1));

}

for(i=0;i<m\*n;i++) {

read(STDIN\_FILENO,num1,15);

write(trub1[1], &num1, sizeof(num1));

}

read(trub2[0], &num1, sizeof(num1));

write(STDOUT\_FILENO, "Means value first matrix\n", sizeof "Means value first matrix\n" - 1);

write(STDOUT\_FILENO,&num1,strlen(num1));

s = '\n';

write(STDOUT\_FILENO,&s, sizeof(s));

read(trub2[0], &num1, sizeof(num1));

write(STDOUT\_FILENO, "Means value second matrix\n", sizeof "Means value second matrix\n" - 1);

write(STDOUT\_FILENO,&num1,strlen(num1));

write(STDOUT\_FILENO,&s, sizeof(s));

//parent process

}

return 0;

}

**Тестирование.**

Числа вводятя по одному через enter. Вводятся числа m и n, затем вводится m\*n чисел первой матрицы и второй матрицы соответственно. В случае если считанное m не является числом или равно нулю, то выведется информация о неккоректном вводе, а затем m будет считанно заново.

test1:

2

2

2

5

9

9

9

3

9

3

3

**Демонстрация работы программы.**

main:Desktop petruhin$ gcc fork.c -o tesy

**fork.c:** В функции «**ftoa**»:

**fork.c:29:6:** **предупреждение:** implicit declaration of function «**reverse**»; did you mean «**revoke**»? [**-Wimplicit-function-declaration**]

**reverse**(s);

**^~~~~~~**

revoke

**fork.c:** На верхнем уровне:

**fork.c:32:7:** **предупреждение:** несовместимые типы для «**reverse**»

void **reverse**(char s[])

**^~~~~~~**

**fork.c:29:6:** **замечание:** здесь была предыдущая неявная декларация «**reverse**»

**reverse**(s);

**^~~~~~~**

main:Desktop petruhin$ ./tesy

2

enter n not equal to zero

2

5

9

9

9

3

9

3

3

Means value first matrix

8.0

Means value second matrix

4.5

main:Desktop petruhin$ ./tesy

0

m should not be zero enter number m

4

enter n not equal to zero

1

10

10

10

-110

1

1

1

3

Means value first matrix

-20.0

Means value second matrix

1.5

main:Desktop petruhin$ ./tesy

afsg

m should not be zero enter number m

3

enter n not equal to zero

2

-10

-25

100

-96

37

99.56

32.44

55.99

32.11

-350.33

21.9

311

Means value first matrix

17.5933342

Means value second matrix

17.1849994

**Вывод.**

Pipe() не очень удобен в более усложненных задачах. Механизм pipe пригоден лишь для родственных процессов, имеющих общего прародителя, инициировавшего системный вызов pipe() и для общения с другими процессами использоваться pipe() не может.

Так же pipe() имеет однонаправленную связь, т.е процесс-родитель пишет информацию в pipe, а затем процесс-ребенок ее оттуда считывает, но уже процесс-ребенок не сможет записать данные в этот pipe.

При громоздких программах очень сложно уследить все особенности и нюансы работы системных вызовов read() и write() при работе c pip’ом, а следовательно возможны ошибки и проблемы с работоспособностью программы.

Системный вызов pipe() явно не лучшое решение для обмена данными между процессами, он требует определенных доработок и упрощенного использования для студентов.