**Отчет**

**по лабораторной работе №2.**

**3 семестр**

|  |  |
| --- | --- |
| **Студент:** | **Фирфаров А.С.** |
| **Группа:** | **8O-208Б** |
| **Преподаватель** | **Миронов Е.С.** |

## 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 решения задачи один или несколько дочерних процессов. Взаимодействие между процессами осуществляется через системные сигналы/события и/или каналы (pipe).

**Вариант задания**

26. Дочерний процесс при создании принимает имя файла. При работе дочерний процесс получает числа от родительского процесса и пишет их в файл. Родительский процесс создает n дочерних процессов и передает им поочередно числа из последовательности от 1...m.

**Описание алгоритма**

Создается n дочерних процессов. Между родительским и дочерним процессом устанавливается соединение посредством pipe. Дочерним процессам последовательно передается имя файла и последовательность чисел 1…m с помощью write. Дочерний процесс принимает данные с помощью read и записывает последовательность чисел в указанный файл. Далее дочерний процесс завершается.

**Текст программы**

#include <unistd.h>

#include <stdio.h>

#include <stdlib.h>

int main() {

int n;

int m;

int pipefd[2];

int resultFork;

printf("Кол-во дочерних процессов: \n");

scanf("%d", &n);

printf("Введите предел последовательности: \n");

scanf("%d", &m);

for (int i = 0; i < n; ++i) {

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

printf("Не удалось создать pipe\n");

exit(-1);

}

resultFork = fork();

if (resultFork < 0) {

printf("Не удалось создать дочерний процесс\n");

exit(-1);

}

else if (resultFork > 0) {

char fileName[256];

close(pipefd[0]);

printf("Введите имя файла: ");

scanf("%s", fileName);

write(pipefd[1], fileName, 256);

for (int j = 1; j < m + 1; ++j) {

write(pipefd[1], &j, sizeof(int));

}

close(pipefd[1]);

}

else if (resultFork == 0) {

char name[256];

int num = 0;

close(pipefd[1]);

read(pipefd[0], name, 256);

FILE\* fp = fopen(name,"a");

while (read(pipefd[0], &num, sizeof(int))) {

fprintf(fp, "%d ", num);

}

fclose(fp);

close(pipefd[0]);

exit(0);

}

}

return 0;

}

**Тестирование**

user@lubuntu:~/OC/lab2$ ./main.exe

Кол-во дочерних процессов:

4

Введите предел последовательности:

20

Введите имя файла: 1

Введите имя файла: 2

Введите имя файла: 3

Введите имя файла: 4

user@lubuntu:~/OC/lab2$ cat 1

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 user@lubuntu:~/OC/lab2$

user@lubuntu:~/OC/lab2$ cat 2

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 user@lubuntu:~/OC/lab2$

user@lubuntu:~/OC/lab2$ cat 3

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 user@lubuntu:~/OC/lab2$

user@lubuntu:~/OC/lab2$ cat 4

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 user@lubuntu:~/OC/lab2$

user@lubuntu:~/OC/lab2$ ./main.exe

Кол-во дочерних процессов:

1

Введите предел последовательности:

5

Введите имя файла: wwwwweeeeerrrrrtttttffffffggggggg

user@lubuntu:~/OC/lab2$ cat wwwwweeeeerrrrrtttttffffffggggggg

1 2 3 4 5 user@lubuntu:~/OC/lab2$

user@lubuntu:~/OC/lab2$ ./main.exe

Кол-во дочерних процессов:

2

Введите предел последовательности:

1

Введите имя файла: t.txt

Введите имя файла: k.txt

user@lubuntu:~/OC/lab2$ cat t.txt

1 user@lubuntu:~/OC/lab2$

user@lubuntu:~/OC/lab2$ cat k.txt

1 user@lubuntu:~/OC/lab2$

user@lubuntu:~/OC/lab2$

**Выводы**

В ходе данной лабораторной работы я понял, что использование нескольких процессов позволяет разделить задачу на подзадачи. Каждый процесс при этом выполняет свою собственную задачу. В моем варианте лабораторной работы каждый дочерний процесс записывает последовательность чисел в файл. Обмен данными между дочерними и родительскими процессами производится посредством pipe. Важно уметь корректно передавать данные между процессами с помощью файловых дескрипторов и системных вызовов для нормального функционирования программы. Также следует убедиться, что процессы не мешают работе друг друга. Я думаю, что навыки по использованию нескольких процессов в программе важны, так как это широко используется в программировании и позволяет распределить задачу между несколькими процессами.