****Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №2 по курсу**

**«Операционные системы»**

Группа: М80 – 201Б-19

Студент: Цыкин И.А.

Преподаватель: Миронов Е.С.

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Москва, 2020.

**Содержание**

1. Постановка задачи
2. Общие сведения о программе
3. Общий метод и алгоритм решения
4. Листинг программы
5. Результаты работы программы
6. 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).

Вариант 6: В файле записаны команды вида: «число число число<endline>». Дочерний процесс считает их сумму и выводит результат в стандартный поток вывода. Числа имеют тип int.

**Общие сведения о программе**

Программа компилируется из двух файлов parent.c и child.c.

В программе используются следующие системные вызовы:

1. **pipe** – для создания однонаправленного канала, через который могут общаться два процесса. Системный вызов возвращает два дескриптора файлов. Один для чтения из канала, другой для записи в канал.
2. **fork** – для создания дочернего процесса.
3. **int execve(const char \*filename, char \*const argv[], char \*const envp[])** (и другие вариации exec) - замена образа памяти процесса
4. **int dup2(int oldfd, int newfd)** - переназначение файлового дескриптора

**Общий метод и алгоритм решения**.

Для реализации, поставленной задачи необходимо:

1. Создать программу, которую будет принимать данные, по этим данным открывать файл и подсчитывать числа и записывать результат в стандартный вывод, а затем компилируем
2. Прочитать название файла
3. Создать массив аргументов, который позже будет передаваться дочернему процессу
4. Используя системный вызов pipe создать канал, по которому будут обмениваться данными два процесса.
5. Используя системный вызов fork создать дочерний процесс.
6. В дочернем процессе происходит замена параметров ввода и вывода
7. В дочернем процессе происходит замена образа памяти процесса, куда передаются аргументы.
8. Дочерний процесс выводит результат используя write в стандартный поток вывода.
9. Родительский процесс прочитывает исходящие данные и выводит результат

**Листинг программы**

**parent.c:**

#include <stdio.h>//подключение библиотек

#include <stdlib.h>

#include <unistd.h>

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

int fd[2];

char name[100];

printf("Enter name of file: ");

scanf("%s", name);//получаем имя файла для прочтения

if(pipe(fd) == -1){//создаем канал

printf("Error with opening pipe\n");

return 2;

}

char fd\_ = fd[1] + '0';//перевод в тип char

char\* arg[] = {name, &fd\_, NULL};//массив данных для ввода в дочерний процесс

if (fork() > 0) {//родительский процесс

int res;

read(fd[0], &res, sizeof(int));

printf("Result from child:%d\n", res);

}else {{//дочерний процесс

if (dup2(fd[1],1) == -1){//передача параметров ввода и вывода

printf("Error with opening dup2\n");

return 1;

}

if (dup2(fd[0],0) == -1){

printf("Error with opening dup2\n");

return 1;

}

execve("./child", arg, NULL);//открытие созданного процесса

}

return 0;

}

**сhild.c:**

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

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

int i = 0;

int fd = atoi(argv[1]); //Для перевода дескрипторов из char\* обратно в int

FILE\* file;

file = fopen(argv[0], "r");

if(file == NULL){

printf("Error with opening file\n");

return 3;

}

int a1, a2, a3;

fscanf(file, "%d %d %d\n", &a1, &a2, &a3);

int res = a1 + a2 + a3;

write(1, &res, sizeof(int));// вывод результата в стандартный поток вывода

fclose(file);

return 0;

}

**text.txt:**

**333 -133 688**

**Результаты работы программы**

vaney@V-box:~/Examples$ gcc -o child child.c  
vaney@V-box:~/Examples$ gcc -o a.out parent.c  
vaney@V-box:~/Examples$ ./a.out  
Enter name of file: test.txt  
Result from child:888  
vaney@V-box:~/Examples$

**Вывод**

Выполнив лабораторную работу, я познакомился с такими вещами, как процессы и каналы, а также с системные вызовы. Связывание процессов может помочь повысить эффективность программы, решающей задачу, в которой есть множество независимых действий. Каналы позволяют передавать информацию между процессами, что позволяет независимым процессам синхронизировать работу, решая общую задачу, с их помощью можно управлять работой процессов, передавая им различные сигналы и сообщения. Эти знания необходимы для эффективного написания программ, использующих идею разделенния процессов.