****Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №2**

**По курсу:  
«Операционные системы»**

|  |  |
| --- | --- |
| Студент: | Захаров И.С. |
| Группа: | М8О-208Б-17 |
| Преподаватель: | Миронов Е. С. |
|  |  |
| Оценка: |  |
| Дата: |  |

**Москва, 2018 г.**

## 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.

7. На вход программе подается название 2-ух файлов. Необходимо отсортировать оба файла (каждый в отдельном процессе) произвольной сортировкой (на усмотрение студента). Содержимое обоих файлов вывести в стандартный поток вывода родительским процессом.

## Системные вызовы

Для ОС Linux

* pid\_t fork() - создание дочернего процесса
* int execve(const char \*filename, char \*const argv[], char \*const envp[]) (и другие вариации exec) - замена образа памяти процесса
* pid\_t waitpid(pid\_t pid, int \*status, int options) - Ожидание завершения дочернего процесса
* void exit(int status) - завершения выполнения процесса и возвращение статуса
* int pipe(int pipefd[2]) - создание неименованного канала для передачи данных между процессами
* int dup2(int oldfd, int newfd) - переназначение файлового дескриптора
* int open(const char \*pathname, int flags, mode\_t mode) - открытие\создание файла
* int close(int fd) - закрыть файл
* int mkfifo(const char \*pathname, mode\_t mode) - создание именованного канала

**Исходный код**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/wait.h>

#include <errno.h>

#include <math.h>

// size of file

int filesize( FILE \*fp ){

int k = 0;

int s;

while ((fscanf(fp, "%d ", &s)!=EOF)) {

if(!fp) break; //чтобы не делал лишнего

k+=1;}

return k;

}

int cmpfunc (const void \* a, const void \* b) {

return ( \*(int\*)a - \*(int\*)b );

}

// sorting for file

int sort(char \*fname) {

int err = 0;

int sizef;

if (fname) {

FILE \*f = fopen(fname, "r");

if (f) {

sizef = filesize(f);

fclose(f);

int \*A;

A = (int\*)malloc(sizef \* sizeof(int));

f = fopen(fname, "r");

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

fscanf(f, "%d ", &A[i]);

}

fclose(f);

qsort(A, sizef, sizeof(int), cmpfunc);

f = fopen(fname, "w");

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

fprintf(f, "%d ", A[i]);

}

free(A);

fclose(f);

}

else

err = 1;

}

return err;

}

// function printing for parent process

int parent(char \*fname1, char \*fname2) {

int err = 0;

printf("Printing files : %s %s\n", fname1, fname2);

if (fname1 && fname2) {

FILE \*f1 = fopen(fname1, "r");

int size1 = filesize(f1);

if (f1) {

FILE \*f2 = fopen(fname2, "r");

int size2 = filesize(f2);

if (f2) {

f1 = fopen(fname1, "r");

f2 = fopen(fname2, "r");

printf("FIRST FILE : \n ");

int \*A1 = (int\*)malloc(size1 \* sizeof(int));

for(int i = 0; i < size1; i++){

fscanf(f1, "%d", &A1[i]);

printf("%d ", A1[i]);

}

printf("\nSECOND FILE : \n ");

int\* A2 = (int\*)malloc(size2 \* sizeof(int));

for(int i = 0; i < size2; i++){

fscanf(f2, "%d", &A2[i]);

printf("%d ", A2[i]);

}

free(A1);

free(A2);

fclose(f1);

fclose(f2);

}

else

err = -2;

}

else

err = -1;

}

printf("\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

printf("Finished printing\n");

return err;

}

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

int err = 0;

if (argc > 2) {

int pid1 = fork();

if (pid1 < 0) {

perror("Fork error: ");

err = -1;

}

else if (!pid1) {

int pid2 = fork();

if (pid2 < 0) {

perror("Fork error: ");

err = -2;

}

else if (!pid2)

err = sort(argv[2]);

else {

err = sort(argv[1]);

waitpid(pid2, 0, 0);

}

}

else if (pid1) {

waitpid(pid1, 0, 0);

err = parent(argv[1], argv[2]);

}

}

return err;

}

## Отчет Strace

munmap(0x7f24fda89000, 16121856) = 0

munmap(0x7f24fcb29000, 16121856) = 0

close(5) = 0

close(6) = 0

write(1, " 90 90 90 90 90 90 90 90 90 90 9"..., 605 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90

) = 605

write(1, "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n", 21\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

) = 21

write(1, "Finished printing\n", 18Finished printing

) = 18

exit\_group(0) = ?

+++ exited with 0 +++

## Вывод

Из данной лабораторной работы могу сделать вывод, что использование дополнительных процессов в проектах бывает полезно. К примеру, в своей лабораторной работе я использовал два дочерних процесса, в каждом из которых я отсортировал по одному файлу, которые пользователь подает при вызове программы как аргумент. Также в каждом процессе я возвращаю свой код, по которому можно понять род ошибки.