Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №2 по курсу**

**«Операционные системы»**

**Управление процессами в ОС. Обеспечение обмена данных между процессами посредством технологии “File mapping”.**

Студент: Речинская Ангелина Юрьевна

Группа: М80-206Б-20

Вариант: 1

Преподаватель: Соколов Андрей Алексеевич

Дата: 18.12.2021

Оценка: 5

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2021

**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 через отображаемые файлы (memory-mapped files).

Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы.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 при необходимости 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.

1 вариант) Пользователь вводит команды вида: «число число число». Далее эти числа передаются от родительского процесса в дочерний. Дочерний процесс считает их сумму и выводит её в файл. Числа имеют тип int. Количество чисел может быть произвольным.

**Листинг программы**

**child.c**

#include <stdio.h>

#include <stdbool.h>

#include <stdlib.h>

#include <sys/mman.h>

#include <sys/types.h>

#include <fcntl.h>

#include <string.h>

#include <unistd.h>

#include <sys/stat.h>

#include <ctype.h>

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

if (argc != 3){

perror("incorrect number of arguments\n");

exit(EXIT\_FAILURE);

}

int out=open(argv[2],O\_CREAT | O\_WRONLY | O\_TRUNC);

if (out==-1){

close(out);

perror("I can not open file\n");

exit(EXIT\_FAILURE);

}

int k;

int desc = open(argv[1], O\_RDWR);

if(desc < 0){

perror("Tmp file not created\n");

return -6;

}

struct stat buff;

stat(argv[1], &buff);

k=buff.st\_size;

char\* fd = mmap(NULL, sizeof(char)\*k,PROT\_WRITE|PROT\_READ,MAP\_SHARED,desc, 0);

if (fd == MAP\_FAILED){

perror("mmap error\n");

return -5;

}

int cnt=0;

char M[k];

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

int sum=0;

while(fd[i]!='\n'){

int n=0;

int z=1;

while(fd[i]==' ' || fd[i]=='\t'){

i++;

}

if (fd[i]=='-'){

z=-1;

i++;

}

while(fd[i]!=' ' && fd[i]!='\n' && fd[i]!='\t' && i<k){

n=n\*10+fd[i]-'0';

i++;

}

sum=sum+z\*n;

}

char t[11];

int u;

int j=0;

int sum1=sum;

if (sum1<0){

t[j]='-';

j++;

sum1\*=-1;

}

if (sum==0){

t[j]='0';

j++;

}

while(sum1!=0){

u=sum1%10;

sum1=sum1/10;

t[j]=u+'0';

j++;

}

char g[j];

if (t[0]=='-'){

g[0]='-';

for(int i=j-1;i>0;i--){

g[j-i]=t[i];

}

}

else{

for(int i=j-1;i>-1;i--){

g[j-i-1]=t[i];

}

}

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

M[cnt]=g[i];

cnt++;

}

M[cnt]='\n';

cnt++;

}

cnt++;

fd[0]='\n';

write(out,&M,sizeof(char)\*(cnt-1));

for(int i=1;i<cnt;i++){

fd[i]=M[i-1];

}

msync(fd, sizeof(char)\*cnt, MS\_SYNC);

munmap(fd, sizeof(char)\*cnt);

ftruncate(desc, sizeof(char)\*cnt);

close(desc);

close(out);

return 0;

}

**main.c**

#include <unistd.h>

#include <stdio.h>

#include <stdbool.h>

#include <stdlib.h>

#include <string.h>

#include <sys/wait.h>

#include <sys/mman.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <ctype.h>

#include "unistd.h"

int main(){

char string[100];

char p;

read(0,&p,sizeof(char));

int z=0;

while(p!='\n'){

string[z]=p;

z++;

read(0,&p,sizeof(char));

}

char file\_name[z];

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

file\_name[i]=string[i];

}

char template[] = "/tmp/tmpXXXXXX";

int desc = mkstemp(template);

if(desc < 0){

perror("Tmp file not created\n");

return -6;

}

printf("%s\n", template);

int id1 = fork();

if (id1 == -1) {

perror("fork error");

exit(EXIT\_FAILURE);

} else if (id1 == 0){

sleep(5);

execl("child.out", "child.out", template, file\_name, NULL);

} else{

char M[3000];

int k=0;

while(read(0,&p,sizeof(char))!=0){

if((p<'0' || p>'9') && p!=' ' && p!='\n' && p!='\t' && p!='-'){

while(read(0,&p,sizeof(char))){

}

exit(EXIT\_FAILURE);

}

M[k]=p;

k++;

}

ftruncate(desc, sizeof(char)\*k);

char\* fd = mmap(NULL, sizeof(char)\*k, PROT\_READ | PROT\_WRITE, MAP\_SHARED, desc, 0);

if (fd == MAP\_FAILED){

perror("mmap error\n");

exit(EXIT\_FAILURE);

}

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

fd[i]=M[i];

}

msync(fd, sizeof(char)\*k, MS\_SYNC);

munmap(fd, sizeof(char)\*k);

int status;

waitpid(id1, &status,WUNTRACED);

if(WEXITSTATUS(status)){

exit(EXIT\_FAILURE);

}

struct stat buff;

fstat (desc, &buff);

k=buff.st\_size;

fd = mmap(NULL, sizeof(char)\*k, PROT\_READ | PROT\_WRITE, MAP\_SHARED, desc, 0);

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

printf("%c",fd[i]);

}

}

return 0;

}

**Примеры работы**

salutik@salutik:~/un/OS/lab4$ ./main

qwerty

/tmp/tmpbyMwNi

1 2 3 4 5

-1 0 1

0 0 0

-10 -3 -1

-10 3

1

0

-1

15

0

0

-14

-7

1

0

-1

salutik@salutik:~/un/OS/lab4$ cat qwerty

15

0

0

-14

-7

1

0

-1

**Вывод**

В данной лабораторной работе мной был изучен и применён на практике механизм межпроцессорного взаимодействия при помощи отображаемых файлов (технология «File Mapping»). Файл отображается на оперативную память, так что мы получаем доступ к его содержимому и можем обращаться с ним как с массивом.

Таким образом, вместо многократного выполнения небыстрых запросов на чтение и запись мы выполняем отображение файла на ОЗУ и получаем произвольный доступ за О(1). По этой причине при использовании технологии «File Mapping» можно добиться ускорения работы программы в несколько раз по сравнению с использованием, например, механизма межпроцессорного взаимодействия при помощи каналов.

В качестве недостатка можно выделить тот факт, что дочерние процессы обязательно должны знать имя отображаемого файла и выполнить их отображение перед началом работы. Также файлы занимают память, несравнимую с каналами.