Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №4 по курсу**

**«Операционные системы»**

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

Студент: Почечура Артемий Андреевич

Группа: М80-206Б-20

Вариант: 14

Преподаватель: Соколов Андрей Алексеевич

Дата: 20.11.2021

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 memory-mapped files.

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

Родительский процесс создает два дочерних процесса. Перенаправление стандартных потоков ввода-вывода показано на картинке выше. Child1 и Child2 можно «соединить» между собой дополнительным каналом. Родительский и дочерний процесс должны быть представлены разными программами.

Родительский процесс принимает от пользователя строки произвольной длины и пересылает их в pipe1. Процесс child1 и child2 производят работу над строками. Child2 пересылает результат своей работы родительскому процессу. Родительский процесс полученный результат выводит в стандартный поток вывода.

14 вариант) Child1 переводит строки в нижний регистр. Child2 убирает все задвоенные пробелы.

**Листинг программы**

**child1.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[]){

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);

int n=buff.st\_size;

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

if (fd == MAP\_FAILED){

perror("mmap error\n");

return -5;

}

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

fd[i]=tolower(fd[i]);

}

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

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

close(desc);

return 0;

}

**child2.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[]){

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);

int n=buff.st\_size;

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

if (fd == MAP\_FAILED){

perror("mmap error\n");

return -5;

}

char S[n];

int r=0;

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

if(i>0 && fd[i]==' ' && fd[i-1]==' '){

continue;

}

S[r]=fd[i];

r++;

}

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

fd[i]=S[i];

}

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

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

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

close(desc);

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 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");

return -1;

} else if (id1 == 0){

sleep(10);

execl("child1.out", "child1.out", template, NULL);

} else{

int id2 = fork();

if (id2 == -1) {

perror("fork error");

return -1;

} else if(id2==0){

sleep(10);

execl("child2.out", "child2.out", template, NULL);

} else {

int n=0;

char p;

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

while(p!='\n'){

if(p<'0' || p>'9'){

printf("Wrong values\n");

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

}

return -1;

}

n=n\*10+p-'0';

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

}

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

char S[n];

read(0,&S,sizeof(char)\*n);

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

if (fd == MAP\_FAILED){

perror("mmap error\n");

return -5;

}

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

fd[i]=S[i];

}

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

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

int status;

waitpid(id2, &status,WUNTRACED);

if(WEXITSTATUS(status)){

return -1;

}

struct stat buff;

fstat (desc, &buff);

n=buff.st\_size;

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

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

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

}

}

}

return 0;

}

**Примеры работы**

root@DESKTOP-5HM2HTK:~#./a.out

10

AAAA AAA

aaaa aaa

root@DESKTOP-5HM2HTK:~#./a.out

5

KKKK

kkkk

root@DESKTOP-5HM2HTK:~#./a.out

8

R y

r y

**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.