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: Лисин Роман Сергеевич

Группа: М80-206Б-20

Вариант: 13

Преподаватель: Соколов Андрей Алексеевич

Дата: 13.11.2021

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.

13 вариант: Child1 переводит строки в нижний регистр. Child2 превращает все пробельные символы в символ «\_».

**Листинг программы**

**child1.c**

#include <stdio.h>

#include <stdlib.h>

#include <sys/wait.h>

#include "ctype.h" //to\_lower

#include "string.h"

#include "unistd.h"

#include <sys/mman.h>

#include <sys/stat.h>

#include <fcntl.h>

#define MAP\_SIZE 4096

char empty = 1;

char\* empty\_string = &empty;

int main() {

char\* file\_name = "file1\_mapped";

int fd = open(file\_name, O\_RDWR | O\_CREAT, S\_IWRITE | S\_IREAD);

if (fd < 0) {

perror("Can't open file");

return -1;

}

char\* file = mmap(NULL, MAP\_SIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0);

if (file == MAP\_FAILED) {

perror("Can't map a file");

return -1;

}

while (strcmp(file, empty\_string) == 0) {}

char\* string = (char\*)malloc(strlen(file) \* sizeof(char));

strcpy(string, file);

for (int i = 0; i < strlen(string); ++i) {

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

// fprintf(stderr, "%c\n", str[i]);

}

// string[0] = 2;

strcpy(file, string);

free(string);

return 0;

}

**child2.c**

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#include <sys/wait.h>

#include "ctype.h" //to\_lower

#include "string.h"

#include "unistd.h"

#include <sys/mman.h>

#include <sys/stat.h>

#include <fcntl.h>

#define MAP\_SIZE 4096

char empty = 1;

char\* empty\_string = &empty;

bool check\_uppercase(char\* str) {

for (int i = 0; i < strlen(str); ++i) {

if (str[i] != tolower(str[i])) {

return false;

}

}

return true;

}

int main() {

char\* file\_name = "file1\_mapped";

// fprintf(stderr, "%d\n", n);

int fd = open(file\_name, O\_RDWR | O\_CREAT, S\_IWRITE | S\_IREAD);

if (fd < 0) {

perror("Can't open file");

return -1;

}

char\* file = mmap(NULL, MAP\_SIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0);

if (file == MAP\_FAILED) {

perror("Can't map a file");

return -1;

}

while (strcmp(file, empty\_string) == 0 || !check\_uppercase(file)) {}

char\* string = (char\*)malloc(strlen(file) \* sizeof(char));

strcpy(string, file);

for (int i = 0; i < strlen(string); ++i) {

if (string[i] == ' ') {

string[i] = '\_';

}

// fprintf(stderr, "%c\n", str[i]);

}

//string[0] = 3;

strcpy(file, string);

free(string);

return 0;

}

**parent.c**

#include <stdio.h>

#include <stdlib.h>

#include <sys/wait.h>

#include "ctype.h" //to\_lower

#include "string.h"

#include "unistd.h"

#include <sys/mman.h>

#include <sys/stat.h>

#include <fcntl.h>

#define MAP\_SIZE 4096

// files for mapping

char\* file1\_name = "file1\_mapped";

// empty string as a signal

char empty = 1;

char\* empty\_string = &empty;

char\* get\_string() {

int len = 0, capacity = 10;

char\* s = (char\*)malloc(10 \* sizeof(char));

if (s == NULL) {

perror("Can't read a string");

exit(6);

}

char c;

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

while (c != '\n') {

s[len++] = c;

if (c == EOF) {

break;

}

if (len == capacity) {

capacity \*= 2;

s = (char\*)realloc(s, capacity \* sizeof(char));

if (s == NULL) {

perror("Can't read a string");

exit(6);

}

}

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

};

s[len] = '\0';

return s;

}

int main() {

// create a file for mapping

int fd1 = open(file1\_name, O\_RDWR | O\_CREAT, S\_IWRITE | S\_IREAD);

if (fd1 < 0) {

perror("Can't open file");

return -1;

}

// we should write smth to the file

if (write(fd1, empty\_string, sizeof(empty\_string)) < 0) {

perror("Can't write to file");

return -1;

}

char\* file1 = mmap(NULL, MAP\_SIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd1, 0);

if (file1 == MAP\_FAILED) {

perror("Can't map a file");

return -1;

}

pid\_t pid1 = fork();

if (pid1 == -1) {

perror("Child 1 wasn't created");

return -1;

}

if (pid1 == 0) { // child 1

execl("child1", "child1", NULL);

perror("Can't execute child process");

}

else {

pid\_t pid2 = fork();

if (pid2 == -1) {

perror("Child 2 wasn't created");

return -1;

}

if (pid2 == 0) { //child 2

execl("child2", "child2", NULL);

}

else { // parent

int n;

//read(0, &n, sizeof(int));

char\* str = get\_string();

n = strlen(str);

strcpy(file1, str);

int status;

if (wait(&status) == -1){

perror("wait");

}

if (wait(&status) == -1){

perror("wait");

}

if (!WIFEXITED(status) || (WIFEXITED(status) && WEXITSTATUS(status) != 0)) {

perror("Error in children");

}

// while (strcmp(file1, empty\_string) != 0) {}

char\* string = (char\*)malloc(n \* sizeof(char));

strcpy(string, file1);

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

free(string);

if (munmap(file1, MAP\_SIZE) < 0) {

perror("Can't unmap files");

return -1;

}

if (close(fd1) < 0) {

perror("Can't close files");

return -1;

}

if (remove(file1\_name) < 0) {

perror("Can't delete files");

return -1;

}

}

}

return 0;

}

**Makefile**

all: child1 child2 var13

child1:

gcc child1.c -o child1

child2:

gcc child2.c -o child2

var13:

gcc var13.c -o var13

clean:

rm -rf child1 child2 var13

**Примеры работы**

roma@DESKTOP-JD58QU2:~/os\_lab2$ ./var13

N waS H

n\_was\_h

roma@DESKTOP-JD58QU2:~/os\_lab2$ ./var13

N KHSIOSJCBKJXLZ KL

n\_khsiosjcbkjxlz\_kl

roma@DESKTOP-JD58QU2:~/os\_lab2$ ./var13

N

n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

roma@DESKTOP-JD58QU2:~/os\_lab2$ ./var13

LJnenso o

\_\_ljnenso\_o

roma@DESKTOP-JD58QU2:~/os\_lab2$ ./var13

OppO

oppo

roma@DESKTOP-JD58QU2:~/os\_lab2$ ./var13

JJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJWOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO

jjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjwooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo

roma@DESKTOP-JD58QU2:~/os\_lab2$ ./var13

M kMla lLLL mLMLcql; npoK

\_\_m\_kmla\_llll\_mlmlcql;\_npok

**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.