Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №4 по курсу**

**«Операционные системы»**

Студент: Ивенкова Л.В.

Группа: М8О–208Б–19

Вариант: 13

Преподаватель: Миронов Е.С.

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2021

**Постановка задачи**

**Цель работы**

Приобретение практических навыков в:

· Освоение принципов работы с файловыми системами

· Обеспечение обмена данных между процессами посредством технологии «File mapping»

**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 через отображаемые файлы (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.

Вариант 13: Child1 переводит строки в нижний регистр. Child2 заменяет пробелы на «\_».

**Общие сведения о программе**

Программа состоит из файлов parent.c, child1.c и child2.c.

**Общий метод и алгоритм решения**.

Родительский процесс принимает на вход строки. Затем открывает файл “File.txt” – его мы будем отображать в памать. Сначала записываем в него считанную нами строку, а затем, собственно, и отображаем в память процесса с помощью mmap (размер файла узнаем с помощью структуры stat).

Далее создаём два дочерних процесса. С помощью вложенных if (по pid процессов) и waitpid контролируем переходы между родительским и дочерними процессами – родитель не начнёт выводить результат, пока оба дочерних алгоритма не закончат работу.

Дочерние алгоритмы вызываем с помощь системного вызова execl, передавая в него имя файла, отображенного в память.

В дочерних процессах мы так же отображаем данный файл в память (при этом, так как мы указывали 4-м параметром «MAP\_SHARED», то всё то, что дочерние процессыбудут делать с файлом, так же произойдёт и в отображённой памяти у родительского процесса). В конце работы дочерние алгоритмы снимают у себя отображение с помощью munmap.

В итоге, всё что они сделали с файлом, теперь лежит в памяти рабъдительского процесса. Считываем это, и выводим на экран.

**Основные файлы программы**

***Child1.c***

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <fcntl.h>

#include <errno.h>

#include <sys/stat.h>

#include <sys/mman.h>

#include <ctype.h>

void low(char\* string, int l){

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

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

}

}

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

void\* adr;

int fd = open(argv[1], O\_RDWR, S\_IRWXU);

if (fd == -1){

perror("Error creating shared child file!\n");

exit(errno);

}

struct stat statBuf;

if(fstat(fd, &statBuf) < 0){

perror("fstat error");

exit(errno);

}

char\* string = (char\*) malloc(statBuf.st\_size);

if(string == NULL){

printf("malloc error");

return 1;

}

adr = mmap(NULL, statBuf.st\_size, PROT\_WRITE | PROT\_READ , MAP\_SHARED, fd, 0);

f (adr == MAP\_FAILED){

perror("mmap error");

exit(errno);

}

close(fd);

strcpy(string, (char\*)adr);

int len = strlen(string);

low(string, len);

sprintf((char\*) adr, "%s", string);

if (munmap(adr, statBuf.st\_size) < 0) {

perror("Can't msync files");

exit(1);

}

}

**Child2.c**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <fcntl.h>

#include <errno.h>

#include <sys/stat.h>

#include <sys/mman.h>

#include <ctype.h>

void SpaceToLine(char\* string, int l){

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

if(string[i] == ' ' )

string[i] = '\_';

}

}

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

void\* adr;

int fd = open(argv[1], O\_RDWR, S\_IRWXU);

if (fd == -1){

perror("Error creating shared child file!\n");

exit(errno);

}

struct stat statBuf;

if(fstat(fd, &statBuf) < 0){

perror("fstat error");

exit(errno);

}

char\* string = (char\*) malloc(statBuf.st\_size);

if(string == NULL){

printf("malloc error");

return 1;

}

adr = mmap(NULL, statBuf.st\_size, PROT\_WRITE | PROT\_READ , MAP\_SHARED, fd, 0);

if (adr == MAP\_FAILED){

perror("mmap error");

exit(errno);

}

close(fd);

strcpy(string, (char\*)adr);

int len = strlen(string);

SpaceToLine(string, len);

sprintf((char\*) adr, "%s", string);

if (munmap(adr, statBuf.st\_size) < 0) {

perror("Can't munmap files");

exit(1);

}

}

**Parent.c**

#include <unistd.h>

#include <stdio.h>

#include <stdlib.h>

#include <errno.h>

#include <fcntl.h>

#include <sys/mman.h>

#include <sys/stat.h>

#include <string.h>

#include <stdbool.h>

#include <ctype.h>

#include <sys/wait.h>

const char\* FILE\_NAME = "file.txt";

void getString(char\* str, int\* cap, int\* len){

char c = getchar();

while (c != EOF) {

str[(\*len)++] = c;

if ((\*len) >= (\*cap)) {

(\*cap) \*= 2;

str = (char\*) realloc(str, (\*cap) \* sizeof(char));

}

c = getchar();

}

str[(\*len)] = '\0';

}

int main(){

struct stat statBuf;

int len = 0;

int cap = 1;

char \*str = (char\*) malloc(cap \* sizeof(char));

printf("Введите строки:\n");

char c = getchar();

while (c != EOF) {

str[(len)++] = c;

if (len >= cap) {

cap \*= 2;

str = (char\*) realloc(str, cap \* sizeof(char));

}

c = getchar();

}

str[len] = '\0';

int tmpfd = open(FILE\_NAME, O\_CREAT | O\_RDWR, S\_IREAD | S\_IWRITE);

if (tmpfd == -1){

perror("Ошибка создания файла!\n");

exit(errno);

}

write(tmpfd, str, strlen(str));

free(str);

if(fstat(tmpfd, &statBuf) < 0){

perror("fstat error");

exit(errno);

}

// MAP\_SHARED - изменения в памяти немедленно будут отражаться в основном файле

// Возвращает указатель на начало памяти

void\* mappedFile = mmap(NULL, statBuf.st\_size, PROT\_READ | PROT\_WRITE, MAP\_SHARED, tmpfd, 0);

if (mappedFile == MAP\_FAILED){

perror("mmap error");

exit(errno);

}

close(tmpfd);

pid\_t id = fork();

if(id == -1){

printf("Fork error\n");

exit(0);

}

else if(id > 0){ //PARENT

int status1;

waitpid(id, &status1, 0);

pid\_t id2 = fork();

if(id2 == -1){

printf("Fork error!\n");

exit(-1);

}

else if(id2 > 0){ //PARENT

int status2;

waitpid(id2, &status2, 0);

char\* result = (char \*) malloc(len \* sizeof(char));

strcpy(result, (char\*)mappedFile);

printf("\nРезультат работы:\n");

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

free(result);

remove(FILE\_NAME);

if (munmap(mappedFile, statBuf.st\_size) < 0) {

perror("Can't munmap file");

exit(1);

}

}

else {

execl("child2", "child2", FILE\_NAME, NULL);

}

}

else if (id == 0)

execl("child1", "child1", FILE\_NAME, NULL);

else{

printf("Fork error 2\n");

exit(-1);

}

return 0;

}

**Пример работы**

parsifal@DESKTOP-3G70RV4:~/OS/Lab4$ strace -o log.txt ./parent

Введите строки:

MrYaaaaUUuuuu?

Nya Crya cvA

beWARe of tHe

WeEpiNg aNgeLs

Or... ..

Puffff!!

Результат работы:

mryaaaauuuuuu?

nya\_crya\_cva

beware\_\_\_of\_the\_

\_\_\_\_weeping\_angels

or...\_\_..

puffff!!

parsifal@DESKTOP-3G70RV4:~/OS/Lab4$ cat log.txt

execve("./parent", ["./parent"], 0x7fffefbb3ef0 /\* 27 vars \*/) = 0

brk(NULL) = 0x7fffd6c56000

arch\_prctl(0x3001 /\* ARCH\_??? \*/, 0x7fffdf4cbac0) = -1 EINVAL (Invalid argument)

access("/etc/ld.so.preload", R\_OK) = -1 ENOENT (No such file or directory)

openat(AT\_FDCWD, "/etc/ld.so.cache", O\_RDONLY|O\_CLOEXEC) = 3

fstat(3, {st\_mode=S\_IFREG|0644, st\_size=30788, ...}) = 0

mmap(NULL, 30788, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7fad40b25000

close(3) = 0

openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libc.so.6", O\_RDONLY|O\_CLOEXEC) = 3

read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\360q\2\0\0\0\0\0"..., 832) = 832

pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784

pread64(3, "\4\0\0\0\20\0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0", 32, 848) = 32

pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\363\377?\332\200\270\27\304d\245n\355Y\377\t\334"..., 68, 880) = 68

fstat(3, {st\_mode=S\_IFREG|0755, st\_size=2029224, ...}) = 0

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7fad40b20000

pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784

pread64(3, "\4\0\0\0\20\0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0", 32, 848) = 32

pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\363\377?\332\200\270\27\304d\245n\355Y\377\t\334"..., 68, 880) = 68

mmap(NULL, 2036952, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7fad408f0000

mprotect(0x7fad40915000, 1847296, PROT\_NONE) = 0

mmap(0x7fad40915000, 1540096, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x25000) = 0x7fad40915000

mmap(0x7fad40a8d000, 303104, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x19d000) = 0x7fad40a8d000

mmap(0x7fad40ad8000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1e7000) = 0x7fad40ad8000

mmap(0x7fad40ade000, 13528, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7fad40ade000

close(3) = 0

arch\_prctl(ARCH\_SET\_FS, 0x7fad40b21380) = 0

mprotect(0x7fad40ad8000, 12288, PROT\_READ) = 0

mprotect(0x7fad40b30000, 4096, PROT\_READ) = 0

mprotect(0x7fad40b1d000, 4096, PROT\_READ) = 0

munmap(0x7fad40b25000, 30788) = 0

brk(NULL) = 0x7fffd6c56000

brk(0x7fffd6c77000) = 0x7fffd6c77000

fstat(1, {st\_mode=S\_IFCHR|0620, st\_rdev=makedev(0x88, 0), ...}) = 0

write(1, "\320\222\320\262\320\265\320\264\320\270\321\202\320\265 \321\201\321\202\321\200\320\276\320\272\320\270:\n", 29) = 29

fstat(0, {st\_mode=S\_IFCHR|0620, st\_rdev=makedev(0x88, 0), ...}) = 0

read(0, "MrYaaaaUUuuuu?\n", 1024) = 15

read(0, "Nya Crya cvA\n", 1024) = 13

read(0, "beWARe of tHe \n", 1024) = 17

read(0, " WeEpiNg aNgeLs\n", 1024) = 19

read(0, "Or... ..\n", 1024) = 10

read(0, "Puffff!!\n", 1024) = 9

read(0, "", 1024) = 0

openat(AT\_FDCWD, "file.txt", O\_RDWR|O\_CREAT, 0600) = 3

write(3, "MrYaaaaUUuuuu?\nNya Crya cvA\nbeWA"..., 83) = 83

fstat(3, {st\_mode=S\_IFREG|0600, st\_size=83, ...}) = 0

mmap(NULL, 83, PROT\_READ|PROT\_WRITE, MAP\_SHARED, 3, 0) = 0x7fad40b2c000

close(3) = 0

clone(child\_stack=NULL, flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLD, child\_tidptr=0x7fad40b21650) = 439

wait4(439, [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 439

--- SIGCHLD {si\_signo=SIGCHLD, si\_code=CLD\_EXITED, si\_pid=439, si\_uid=1000, si\_status=0, si\_utime=0, si\_stime=0} ---

clone(child\_stack=NULL, flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLD, child\_tidptr=0x7fad40b21650) = 440

wait4(440, [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 440

--- SIGCHLD {si\_signo=SIGCHLD, si\_code=CLD\_EXITED, si\_pid=440, si\_uid=1000, si\_status=0, si\_utime=0, si\_stime=0} ---

write(1, "\n", 1) = 1

write(1, "\320\240\320\265\320\267\321\203\320\273\321\214\321\202\320\260\321\202 \321\200\320\260\320\261\320\276\321\202\321\213:"..., 33) = 33

write(1, "mryaaaauuuuuu?\nnya\_crya\_cva\nbewa"..., 83) = 83

write(1, "\n", 1) = 1

unlink("file.txt") = 0

munmap(0x7fad40b2c000, 83) = 0

exit\_group(0) = ?

+++ exited with 0 +++

**Вывод**

В СИ помимо механизма общения между процессами через pipe, также существуют и другие способы взаимодействия, например отображение файла в память, такой подход работает быстрее, засчет отстутствия постоянных вызово read, write и тратит меньше памяти под кэш. После отображения возвращается void\*, который можно привести к своему указателю на тип и обрабатывать данные как массив, где возвращенный указатель – указатель на первый элемент.