Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №4 по курсу**

**«Операционные системы»**

**File mapping**

Студент: Е. А. Айрапетова

Группа: М8О–206Б–20

Вариант: 16

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2020.

**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.

**Общие сведения о программе**

Программа компилируется из файла lab4.c. Также используется заголовочные файлы: stdio.h, stdlib.h, string.h. unistd.h, sys/wait.h, stdbool.h, fcntl.h, sys/mman.h, errno.h. В программе используются следующие системные вызовы:

* pipe - создает канал и возвращает два файловых дескриптора, для взаимодействия по каналу;
* close – закрывает файловый дескриптор;
* fork – создает дочерний процесс и продолжает выполнение текущей программы в нем;

**Общий метод и алгоритм решения**.

Родительский процесс принимает на вход строки. Затем открывает файл, который 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.

В дочерний алгоритм передаем имя файла, отображенного в память. В дочернем процессе мы так же отображаем данный файл в память. В конце работы дочерний алгоритм снимает у себя отображение с помощью munmap.

В итоге, всё что дочерний процесс сделал с файлом, теперь лежит в памяти родительского процесса. Считываем это, и выводим на экран.

**Основные файлы программы**

**lab4.c**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <sys/wait.h>

#include <stdbool.h>

#include <fcntl.h>

#include <sys/mman.h>

#include <errno.h>

int readString(char\*\* str) { // считывает строку (до '\n') и возвращает ее размер

int size = 0;

int capacity = 10;

char a = '\0';

if (!((\*str) = malloc(sizeof(char) \* capacity))) {

write(STDERR\_FILENO, "MEM ERR\n", sizeof(char) \* 8);

exit(-1);

}

while (read(0, &a, sizeof(char)) == 1) {

if (a == '\n') {

break;

}

if (size == capacity) {

capacity = capacity \* 3 / 2;

if (!((\*str) = realloc((\*str), sizeof(char) \* capacity))) {

write(STDERR\_FILENO, "MEM ERR\n", sizeof(char) \* 8);

exit(-1);

}

}

(\*str)[size] = a;

++size;

}

if (size == 0) {

free((\*str));

(\*str) = NULL;

return -1;

}

if (!((\*str) = realloc((\*str), sizeof(char) \* (size + 1)))) {

write(STDERR\_FILENO, "MEM ERR\n", sizeof(char) \* 8);

exit(-1);

}

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

return size;

}

void parent(char\* data, int fd) {

while (1) {

char\* str;

int size = readString(&str);

data[0] = size;

data[1] = -1;

if (size == 0) {

data[2] = '\n';

break;

}

memcpy(data + 2, str, size + 1);

dup2(fd, 1);

while (data[1] != 1 && data[1] != 0) {}

if (data[1] == 1) {

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

}

else {

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

}

}

wait(NULL);

close(fd);

}

void child(char\* data) {

while (1) {

while (data[2] == '\0') {}

char\* str = data + 2;

int size = data[0];

if (size == 0) {

break;

}

if(str[0] < 'A' || str[0] > 'Z'){

data[1] = 0;

}

else {

data[1] = 1;

}

data[2] = '\0';

}

}

int main () {

char\* mfile = malloc(sizeof(char) \* 100);

char\* file = malloc(sizeof(char) \* 100);

scanf("%s", mfile);

scanf("%s", file);

int outFD, mFD;

if ((outFD = open(file, O\_CREAT | O\_WRONLY, S\_IWUSR | S\_IRUSR)) == -1) {

fprintf(stderr, "Open file error\n");

return -1;

}

if ((mFD = open(mfile, O\_CREAT | O\_RDWR, S\_IWUSR | S\_IRUSR)) == -1) {

fprintf(stderr, "Open memory file error\n");

return -1;

}

int pid;

if (ftruncate(mFD, 100)) {

perror("ftruncate");

return -1;

}

char\* ptr;

if ((ptr = (char\*) mmap(NULL, 100, PROT\_READ | PROT\_WRITE, MAP\_SHARED, mFD, 0)) == MAP\_FAILED) {

perror("Map");

return -1;

}

ptr[2] = '\0';

close(mFD);

pid = fork();

if (pid == -1) {

perror("FORK");

}

else if (pid == 0) {

child(ptr);

}

else {

parent(ptr, outFD);

}

munmap(ptr, 100);

close(outFD);

return 0;

}

**Пример работы**

jane@Evgenia:/mnt/c/Files/ОСи$ ./a.out

f.txt

String

Also string

wrong string

2 wrong string

jane@Evgenia:/mnt/c/Files/ОСи$ cat f.txt

String

Also string

ERROR

ERROR

jane@Evgenia:/mnt/c/Files/ОСи$ ./a.out

f.txt

jane@Evgenia:/mnt/c/Files/ОСи$ cat f.txt

jane@Evgenia:/mnt/c/Files/ОСи$

**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.