Московский Авиационный Институт
(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Лабораторная работа №4 по курсу
«Операционные системы»

ОТОБРАЖАЕМЫЕ ФАЙЛЫ

Студент: Лукманова Аэлита
Группа: М8О–201Б–19
Вариант: 21
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.

Вариант 21:

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 принимает от пользователя строки произвольной длины и пересылает их в pipe1 или в pipe2 в зависимости от фильтрации. Дочерние процессы производят работу над строками и пишут результат в стандартный вывод.
Правило фильтрации: нечетные строки в pipe1, четные в pipe2. Дочерние процессы инвертируют строки

Общие сведения о программе

Программа компилируется из файла main.c вместе с файлов io.c. Также используется заголовочные файлы: stdio.h, unistd.h, stdlib.h, stdbool.h, sys/types.h, sys/stat.h, fcntl.h, sys/mman.h. В программе используются следующие системные вызовы:

1. **mmap** — отражает *length* байтов, начиная со смещения *offset* файла (или другого объекта), определенного файловым дескриптором *fd*, в память, начиная с адреса *start*. При удачном выполнении mmap возвращает указатель на область с отраженными данными. При ошибке возвращается значение MAP_FAILED (-1), а переменная *errno* приобретает соответствующее значение.

2. **munmap** — удаляет все отражения из заданной области памяти, после чего все ссылки на данную область будут вызывать ошибку

"неправильное обращение к памяти". При удачном выполнении munmap возвращаемое значение равно нулю. При ошибке возвращается -1, а переменная *errno* приобретает соответствующее значение.

3. **fork —** создает новый процесс, который является копией родительского процесса, за исключением разных process ID и parent process ID. В случае успеха fork() возвращает 0 для ребенка, число больше 0 для родителя – child ID, в случае ошибки возвращает -1.
4. **read —** предназначена для чтения какого-то числа байт из файла, принимает в качестве аргументов файловый дескриптор, буфер, в который будут записаны данные и число байт. В случае успеха вернет число прочитанных байт, иначе -1.
5. **write —** предназначена для записи какого-то числа байт в файл, принимает в качестве аргументов файловый дескриптор, буфер, из которого будут считаны данные для записи и число байт. В случае успеха вернет число записанных байт, иначе -1.

Общий метод и алгоритм решения.

Для реализации поставленной задачи необходимо:

1. Изучить принципы работы mmap.
2. Написать парсер для типа float, который бы превращал строковое представление числа в естественное (учитывая не валидные данные).
3. Организовать считывание названия файла и строки произвольной длины.
4. Разработать архитектуру проекта так, чтобы избежать *race condition* при обращении к общей разделяемой памяти.
5. Реализовать функции для процесса-родителя и процесса-ребенка.
6. Реализовать сообщение между процессами при помощи отраженных данных.
7. Реализовать обработку системных ошибок согласно заданию.

Основные файлы программы

child.h:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <errno.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/mman.h>


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

    char * file_name = argv[1];

    int fd = open(file_name, O_RDWR | O_CREAT);
    if (fd < 0) {
        printf("can't open %s", file_name);
        return 0;
    }

    struct stat st;
    fstat(fd, &st);
    char *contents = mmap(NULL, st.st_size, PROT_READ | PROT_WRITE, MAP_SHARED, fd,
0);

    printf("\nHello, I'm %s\nI've recieved this contents of size %lld :\n%s", argv[0],
st.st_size, contents);

    int currentIndex = 0;
    char *enter = "\n";

    while (currentIndex < (int)st.st_size) {
        char tmpString[100] = {""};
```

```
        int ind = 0;
        while (contents[currentIndex] != *enter) {
            //printf("contents[%d]: %c\n", currentIndex, contents[currentIndex]);
            tmpString[ind] = contents[currentIndex];
            currentIndex++;
            ind++;
        }

        for (size_t i = currentIndex-ind; i < currentIndex; i++) {
            contents[i] = tmpString[currentIndex - i - 1];
        }

        currentIndex++;
    }

    printf("Then I reversed each string and rewrite them. This is result: \n%s\n",
contents);
    munmap(contents, st.st_size);
    close(fd);


    printf("%s say goodbye\n", argv[0]);
    return 0;
}
```

main.c:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <errno.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/mman.h>
#include <signal.h>



#define NUMBER_OF_STRING 25
#define MAX_STRING_SIZE 200


//Вариант 21

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

    //MARK: -названия файлов на запись
    char firstChildFile[200];
    char secChildFile[200];
    printf("Введите имя файла на запись для первого дочернего процесса: ");
    fgets(firstChildFile, 200, stdin);
    firstChildFile[strlen(firstChildFile) - 1] = '\0';

    printf("Спасибо, введите имя файла на запись для второго дочернего процесса: ");
    fgets(secChildFile, 200, stdin);
    secChildFile[strlen(secChildFile) - 1] = '\0';


    //MARK: -инициализация
    int fd[2];
    int fd2[2];
    if (pipe(fd) == -1) {
        return 1;
```

```
}
if (pipe(fd2) == -1) {
    return 1;
}

pid_t pid_ch1 = fork();
if (pid_ch1 == -1) {
    return 2;
}


if (pid_ch1 == 0) { //child1
    printf("child1");

    char *argv[10];

    argv[0] = "./child1";
    argv[1] = firstChildFile;
    argv[2] = NULL;
    wait(NULL);
    execv("./child", argv);
}

if (pid_ch1 > 0) { //parent

    pid_t pid_ch2 = fork();

    if (pid_ch2 == 0) { //child2
        printf("child2");

        char *argv[10];

        argv[0] = "./child2";
        argv[1] = secChildFile;
        argv[2] = NULL;
        wait(NULL);
        execv("./child", argv);

    } else {

        //MARK: -работа с pid
        printf("Hello, I'm a parent\n");

        //отсылаем детям сигналы стоп
        kill(pid_ch1, SIGSTOP);
        kill(pid_ch2, SIGSTOP);

        //заполняем два массива четными и нечентными строками
        char arrayOfString[50][200];
        printf("Input strings: \n");
        int arrayCount = 0;
        char line[200];
        while(fgets(line, 200, stdin)){
            strcpy(arrayOfString[arrayCount], line);
            arrayCount++;
        }

        char arrayOfString1[NUMBER_OF_STRING][MAX_STRING_SIZE];
        char arrayOfString2[NUMBER_OF_STRING][MAX_STRING_SIZE];
        char currentStr[200];

        int arrayCount1 = 0;
        int arrayCount2 = 0;
        for (int i = 0; i < arrayCount; i++) {
            strcpy(currentStr, arrayOfString[i]);
            if (i%2 == 0) {
                strcpy(arrayOfString1[arrayCount1], currentStr);
                arrayCount1++;
            } else {
                strcpy(arrayOfString2[arrayCount2], currentStr);
                arrayCount2++;
            }
```

```
            }


            //создадим отображение файлов детей в виртуальной памяти
            char * file_name_ch1 = firstChildFile;
            char * file_name_ch2 = secChildFile;

//            O_RDWR нужно писать даже если просто записываем, потому что
            int fd_ch1 = open(file_name_ch1, O_RDWR | O_CREAT | O_TRUNC);
            int fd_ch2 = open(file_name_ch2, O_RDWR | O_CREAT | O_TRUNC);
            if (fd_ch1 < 0) {
                printf("can't open %s for reading", file_name_ch1);
                return 0;
            }
            if (fd_ch2 < 0) {
                printf("can't open %s for reading", file_name_ch2);
                return 0;
            }

            //зададим размеры каждому отображению:
            size_t page_size_ch1 = sizeof(arrayOfString1);
            size_t page_size_ch2 = sizeof(arrayOfString2);

//            MAP_PRIVATE - изменения видны только моему процессу, то есть даже не
попадут в файл, не то что в дети. хотя главное в файл, это как раз то, что нам нужно
            char *contents_ch1 = mmap(NULL, page_size_ch1, PROT_READ | PROT_WRITE,
MAP_SHARED, fd_ch1, 0);
            char *contents_ch2 = mmap(NULL, page_size_ch2, PROT_READ | PROT_WRITE,
MAP_SHARED, fd_ch2, 0);

            if (contents_ch1 == MAP_FAILED){
                    close(fd_ch1);
                    perror("Error mmapping the file");
                    exit(EXIT_FAILURE);
            }
            if (contents_ch2 == MAP_FAILED){
                    close(fd_ch2);
                    perror("Error mmapping the file");
                    exit(EXIT_FAILURE);
            }


            //заполним отображения одного файла нечетными строками, другого файла --
четными
            for (int i = 0; i < arrayCount1; i++) {
                write(fd_ch1, arrayOfString1[i], strlen(arrayOfString1[i]));
            }
            for (int i = 0; i < arrayCount2; i++) {
                write(fd_ch2, arrayOfString2[i], strlen(arrayOfString2[i]));
            }

            //покажем результат
            printf("File contents_ch1:\n%s\n", contents_ch1);
            printf("File contents_ch2:\n%s\n", contents_ch2);

            // Write it now to disk
            if (msync(contents_ch1, page_size_ch1, MS_SYNC) == -1) {
                   perror("Could not sync the file to disk");
            }
            if (msync(contents_ch2, page_size_ch2, MS_SYNC) == -1) {
                   perror("Could not sync the file to disk");
            }

            //удалим отображения
            munmap(contents_ch1, page_size_ch1);
            munmap(contents_ch2, page_size_ch2);

            //закроем файлы
            close(fd_ch1);
            close(fd_ch2);

            //отсылаем детям сигнал продолжать выполнение их процессов
```

```
            kill(pid_ch1, SIGCONT);
            kill(pid_ch2, SIGCONT);

            waitpid(pid_ch2, NULL, 0);
            waitpid(pid_ch1, NULL, 0);

            printf("\nparent say goodbye\n");
        }
    }
}
```

Пример работы

```
aelitalukmanova@MacBook-Pro-Aelita os-4 % ./main
Введите имя файла на запись для первого дочернего процесса: 1.txt
Спасибо, введите имя файла на запись для второго дочернего процесса: 2.txt
Hello, I'm a parent
Input strings:
This is 1st string
And this is 2nd one
3d
4th
fifth
This string will be 6th
File contents_ch1:
This is 1st string
3d
fifth

File contents_ch2:
And this is 2nd one
4th
This string will be 6th

Hello, I'm ./child1
I've recieved this contents of size 36 :
This is 1st string
3d
fifth
Then I reversed each string and rewrite them. This is result:
gnirts ts1 si sihT
d3
htfif
./child1 say goodbye
parent say goodbye
aelitalukmanova@MacBook-Pro-Aelita os-4 % ls
1.txt  2.txt  child  child.c      main   main.c
aelitalukmanova@MacBook-Pro-Aelita os-4 % cat 1.txt
gnirts ts1 si sihT
```

```
d3
htfif
aelitalukmanova@MacBook-Pro-Aelita os-4 % cat 2.txt
And this is 2nd one
4th
This string will be 6th
aelitalukmanova@MacBook-Pro-Aelita os-4 %
```

Вывод

File mapping позволяет отобразить данные из файла в какой-то участок памяти, так, что при работе с данным участком памяти производится работа с файлом. Данная технология может прилично увеличить время работы с большими файлами, так как уменьшается число системных вызовов для работы с файлом, не будет происходить лишнего копирования данных в буфер.

Также при помощи «File mapping» можно разделить использование некого участка памяти между процессами, что позволит обмениваться данными между ними. Однако общий участок памяти может быть опасен возможными *race condition'*ами, так как обращение к этой области памяти не является блокирующим, что заставляет программиста пользоваться семафорами и т.п.

Суммируя полученный опыт из ЛР2 и ЛР4, я думаю, что *каналы* лучше подходят для общения "один на один", они более просты в использовании, чем разделяемая память, они лучше подходят для синхронизации процессов, так как они блокирующие; когда как *разделяемая память* больше подходит для асинхронной работы с данными, она лучше подходит для общения между множеством процессов.