Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №4 по курсу**

**«Операционные системы»**

**Освоение принципов работы с файловыми системами. Обеспечение обмена данных между процессами посредством технологии «File Mapping»**

Студент: Гаптулхаков Руслан Рамилевич

Группа: М80 – 208Б-19

Вариант: 9

Преподаватель: Миронов Евгений Сергеевич

Дата:20.04.21

Оценка:

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2021

1. **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 через отображаемые файлы (memory-mapped files).

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

Родительский процесс создаёт дочерний процесс. Первой строкой пользователь в консоль родительского процесса вводит имя файла, из которого будут использованы данные дочерним процессом.

Процесс пишет результаты своей работы в стандартный вывод. Родительский и дочерний процесс должны быть представлены разными программами.

В файле записаны команды вида:«число число число<endline>». Дочерний процесс производит деление первого числа команда, на последующие числа в команде, а результат выводит в стандартный поток вывода. Если происходит деление на 0, то тогда дочерний и родительский процесс завершают свою работу. Проверка деления на 0 должна осуществляться на стороне дочернего процесса. Числа имеют тип float.

1. **Общие сведения о программе**

Программа написана на языке Си в UNIX-подобной операционной системе (Ubuntu). В программе создается дочерний процесс child.

Имена отображаемых файлов известны в заранее. Отображение файлов выполняется при запуске как родительского, так и дочерних процессов.

Программа принимает на вход неограниченное количество чисел.

Программа завершает свою работу при нажатии Ctrl+D.

Программа обрабатывает все возможные системные ошибки и выводит соответствующие сообщения в случае их возникновения.

1. **Общий метод и алгоритм решения**

При запуске программы пользователю предлагается ввести имя файла, откуда будут считываться данные.Если пользователь ввёл имя несуществующего файла, он будет создан.

После запуска программы выполняется отображение двух файлов, имена которых известны заранее. Так как операционная система не позволяет выполнить отображение пустого файла, то перед отображением в файлы записываются «пустые» строки. В качестве «пустой» строки используется строка, состоящая из одного системного символа.

В начале, происходи отоброжение нужных файлов и переменных, имена которых, опредедены, как константы. Отоброжение происходит с ключом shared, чтобы другой процесс работать с общей памятью. Получаем указатели. Функция ftrance() помогает выделить нужное количество памяти. Затем происходит копирование процесса при помощи функции fork(). После в дочернем процессе происходит выполнение файла ./child. Там происходит такое же отображение файлов на память, считывание данных, расчёт и запись в один из файлов. Родительский процесс дожидается конца выполнения дочернего и выводи результат на экран.

1. **Основные файлы программы**

**parent.c**

#include <unistd.h>

#include <stdio.h>

#include <fcntl.h>

#include<stdlib.h>

#include <sys/types.h>

#include <string.h>

#include <sys/mman.h>

#include <sys/wait.h>

#include <sys/stat.h>

#include <pthread.h>

#define SIZEBR 2000

#define SH\_NAME "my\_shared\_mem"

#define SH\_SIZE\_NAME "my\_shared\_mem\_size"

#define MUTEX\_NAME "my\_mutex"

int main()

{

    // Read file name

    char\* name = NULL;

    size\_t len;

    printf("Enter a filename with tests: ");

    if(getline(&name, &len, stdin) == -1){

        perror("getline");

    }

    name[strlen(name) - 1] = '\0';

    // Create file descriptors

    int fd\_shared\_data = shm\_open(SH\_NAME, O\_RDWR | O\_CREAT, S\_IRWXU);

    int fd\_shared\_data\_size = shm\_open(SH\_SIZE\_NAME, O\_RDWR | O\_CREAT, S\_IRWXU);

    int fd\_mutex = shm\_open(MUTEX\_NAME,O\_RDWR | O\_CREAT, S\_IRWXU);

    //int fd\_file = shm\_open(name, O\_RDWR | O\_CREAT,S\_IRWXU);

    if((fd\_shared\_data == -1) || (fd\_shared\_data\_size == -1)|| (fd\_mutex == -1)){

        perror("Error of shm\_open file!");

        return -1;

    }

    // Define  size files

/\*  struct stat statbuf;

    if (fstat(fd\_file, &statbuf) < 0 ){

        perror("fstat");

        return -2;

    }\*/

    ftruncate(fd\_mutex,sizeof(pthread\_mutex\_t\*));

    ftruncate(fd\_shared\_data,getpagesize());

    ftruncate(fd\_shared\_data\_size,sizeof(int));

    // mapping file

    //len = statbuf.st\_size;

    //char\* src = (char\*) mmap(NULL, len, PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd\_file, 0);

    pthread\_mutex\_t \*Lock = (pthread\_mutex\_t\*) mmap(NULL,sizeof(pthread\_mutex\_t\*),PROT\_READ | PROT\_WRITE, MAP\_SHARED,fd\_mutex,0);

    float\* data = (float\*) mmap(NULL,getpagesize(),PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd\_shared\_data, 0);

    int\* sizes\_ = (int\*) mmap(NULL,sizeof(int),PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd\_shared\_data\_size, 0);

    pthread\_mutexattr\_t MutexAttribute;

    pthread\_mutexattr\_setpshared(&MutexAttribute, PTHREAD\_PROCESS\_SHARED);

    pthread\_mutex\_init(Lock, &MutexAttribute);

    // Creating child process

    int id = fork();

    if (id == -1)

    {

        perror("Error: Can't creating fork.");

        return -3;

    }

    if (id == 0)

    { // Child process

        int err = execl("./child", "child",name,SH\_NAME,SH\_SIZE\_NAME,MUTEX\_NAME,NULL);

        if(err == -1){

            perror("Eror of executing child programm");

            return -1;

        }

    }

    else

    { //Parent process

        wait(NULL);

        for(int i = 0; i < \*sizes\_; ++i){

            printf("%f ", data[i]);

            //printf("\nclear");

        }

    }

        munmap(Lock,sizeof(pthread\_mutex\_t\*));

        //munmap(src,len);

        munmap(data, getpagesize());

        munmap(sizes\_, sizeof(int));

        //Close fd shared memory

        close(fd\_shared\_data);

        close(fd\_mutex);

        //close(fd\_file);

        close(fd\_shared\_data\_size);

        //Unlick shared memory

        shm\_unlink(SH\_NAME);

        shm\_unlink(MUTEX\_NAME);

        shm\_unlink(name);

        shm\_unlink(SH\_SIZE\_NAME);

        return 0;

}

**child.c**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <fcntl.h>

#include <stdbool.h>

#include <string.h>

#include <sys/types.h>

#include <pthread.h>

#include <sys/mman.h>

#include <sys/wait.h>

#include <sys/stat.h>

#define SIZEBR 2000

int main(int argc, char\* argv[]){

    char\* filename = argv[1];

    char\* sh\_data\_name = argv[2];

    char\* sh\_size\_data\_name = argv[3];

    char\* mutex\_name = argv[4];

    //Create fd shared file

    int fd\_sh\_data\_name = shm\_open(sh\_data\_name, O\_RDWR | O\_CREAT, S\_IRWXU);

    int fd\_mutex\_name = shm\_open(mutex\_name,O\_RDWR | O\_CREAT, S\_IRWXU);

    int fd\_filename = open(filename,O\_RDWR | O\_CREAT, 0777);

    int fd\_sh\_size\_data\_name = shm\_open(sh\_size\_data\_name,O\_RDWR | O\_CREAT, S\_IRWXU);

    if((fd\_sh\_data\_name == -1) ||(fd\_filename == -1) || (fd\_mutex\_name == -1) ||(fd\_sh\_size\_data\_name == -1)){

        perror("Error of shm\_open file!");

        return -6;

    }

    struct stat statbuf;

    if (fstat(fd\_filename, &statbuf) < 0 ){

        perror("fstat");

        return -9;

    }

    size\_t len = statbuf.st\_size;

    ftruncate(fd\_filename,len);

    //printf("%lu /", len);

    ftruncate(fd\_mutex\_name,sizeof(pthread\_mutex\_t\*));

    ftruncate(fd\_sh\_data\_name,getpagesize());

    ftruncate(fd\_sh\_size\_data\_name,sizeof(int));

    //Map shared files in memory

    char\* str = (char\*) mmap(NULL,getpagesize(),PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd\_filename, 0);

    //printf("%s /", str);

    float\* result\_mass = (float\*) mmap(NULL,getpagesize(),PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd\_sh\_data\_name, 0);

    pthread\_mutex\_t \*Lock = (pthread\_mutex\_t\*) mmap(NULL,sizeof(pthread\_mutex\_t\*),PROT\_READ | PROT\_WRITE, MAP\_SHARED,fd\_mutex\_name,0);

    int\* sizes\_ = (int\*) mmap(NULL,sizeof(int),PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd\_sh\_size\_data\_name, 0);

    float data[2];

    int k = 0, j = 0;

    size\_t index = 0;

    \*sizes\_ = 0;

    char\* substr = (char\*) malloc(sizeof(char)\*256);

    for(int i = 0; i < len; i = i+k+1){

        for(j = i, k = 0; j < len; ++j, ++k){

            if((str[j] == ' ') ||(str[j] == '\n')){

                substr[k] = '\0';

                break;

            }

            substr[k] = str[j];

        }

        if(i == 0){

            data[0] = atof(substr);

            continue;

        }

        if(i != 0){

            data[1] = atof(substr);

            if(data[1] == 0){

                printf("\nDivision 0\n");

                //printf("%d ", \*sizes\_);

                return -10;

            }

            result\_mass[index++] = data[0] / data[1];

            \*sizes\_ += 1;

            data[0] = data[1];

            //printf("%f ", result\_mass[index - 1]);

        }

    }

    free(substr);

    munmap(str,len);

    munmap(Lock,sizeof(pthread\_mutex\_t\*));

    munmap(result\_mass,getpagesize());

    munmap(sizes\_, sizeof(int));

    close(fd\_sh\_data\_name);

    close(fd\_mutex\_name);

    close(fd\_sh\_size\_data\_name);

    close(fd\_filename);

    shm\_unlink(sh\_data\_name);

    shm\_unlink(mutex\_name);

    shm\_unlink(filename);

    shm\_unlink(sh\_size\_data\_name);

    return 0;

}

1. **Демонстрация работы программы**

**rusya@DESKTOP-93JGKCU:/mnt/c/Users/rusya/Desktop/lab4\_OS**$ ./parent

Enter a filename with tests: input.txt

Division 0

1. 1.166667 1.200000 1.250000

**rusya@DESKTOP-93JGKCU:/mnt/c/Users/rusya/Desktop/lab4\_OS$** ./parent

Enter a filename with tests: input2.txt

1.125000 1.142857 1.166667 1.200000 1.250000 1.333333 1.500000 2.000000

**rusya@DESKTOP-93JGKCU:/mnt/c/Users/rusya/Desktop/lab4\_OS$** strace -f -e trace="%process,read,write,dup2,mmap" -o strace\_log.txt ./parent

Enter a filename with tests: input2.txt

1.125000 1.142857 1.166667 1.200000 1.250000 1.333333 1.500000 2.000000

**rusya@DESKTOP-93JGKCU:/mnt/c/Users/rusya/Desktop/lab4\_OS$** cat strace\_log.txt

248 execve("./parent", ["./parent"], 0x7fff2a091b88 /\* 27 vars \*/) = 0

248 arch\_prctl(0x3001 /\* ARCH\_??? \*/, 0x7ffca85df9f0) = -1 EINVAL (Invalid argument)

248 mmap(NULL, 33170, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7f18d1f6a000

248 read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0 7\0\0\0\0\0\0"..., 832) = 832

248 mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f18d1f68000

248 mmap(NULL, 44000, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7f18d1f5d000

248 mmap(0x7f18d1f60000, 16384, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x3000) = 0x7f18d1f60000

248 mmap(0x7f18d1f64000, 4096, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x7000) = 0x7f18d1f64000

248 mmap(0x7f18d1f66000, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x8000) = 0x7f18d1f66000

248 read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\220\201\0\0\0\0\0\0"..., 832) = 832

248 mmap(NULL, 140408, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7f18d1f3a000

248 mmap(0x7f18d1f41000, 69632, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x7000) = 0x7f18d1f41000

248 mmap(0x7f18d1f52000, 20480, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x18000) = 0x7f18d1f52000

248 mmap(0x7f18d1f57000, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1c000) = 0x7f18d1f57000

248 mmap(0x7f18d1f59000, 13432, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7f18d1f59000

248 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

248 mmap(NULL, 2036952, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7f18d1d48000

248 mmap(0x7f18d1d6d000, 1540096, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x25000) = 0x7f18d1d6d000

248 mmap(0x7f18d1ee5000, 303104, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x19d000) = 0x7f18d1ee5000

248 mmap(0x7f18d1f30000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1e7000) = 0x7f18d1f30000

248 mmap(0x7f18d1f36000, 13528, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7f18d1f36000

248 mmap(NULL, 12288, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f18d1d45000

248 arch\_prctl(ARCH\_SET\_FS, 0x7f18d1d45740) = 0

248 write(1, "Enter a filename with tests: ", 29) = 29

248 read(0, "input2.txt\n", 1024) = 11

248 mmap(NULL, 8, PROT\_READ|PROT\_WRITE, MAP\_SHARED, 5, 0) = 0x7f18d1f9f000

248 mmap(NULL, 4096, PROT\_READ|PROT\_WRITE, MAP\_SHARED, 3, 0) = 0x7f18d1f72000

248 mmap(NULL, 4, PROT\_READ|PROT\_WRITE, MAP\_SHARED, 4, 0) = 0x7f18d1f71000

248 clone(child\_stack=NULL, flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLD, child\_tidptr=0x7f18d1d45a10) = 249

248 wait4(-1, <unfinished ...>

249 execve("./child", ["child", "input2.txt", "my\_shared\_mem", "my\_shared\_mem\_size", "my\_mutex"], 0x7ffca85dfad8 /\* 27 vars \*/) = 0

249 arch\_prctl(0x3001 /\* ARCH\_??? \*/, 0x7ffc98c37fc0) = -1 EINVAL (Invalid argument)

249 mmap(NULL, 33170, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7ff420af2000

249 read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0 7\0\0\0\0\0\0"..., 832) = 832

249 mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7ff420af0000

249 mmap(NULL, 44000, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7ff420ae5000

249 mmap(0x7ff420ae8000, 16384, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x3000) = 0x7ff420ae8000

249 mmap(0x7ff420aec000, 4096, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x7000) = 0x7ff420aec000

249 mmap(0x7ff420aee000, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x8000) = 0x7ff420aee000

249 read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\220\201\0\0\0\0\0\0"..., 832) = 832

249 mmap(NULL, 140408, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7ff420ac2000

249 mmap(0x7ff420ac9000, 69632, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x7000) = 0x7ff420ac9000

249 mmap(0x7ff420ada000, 20480, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x18000) = 0x7ff420ada000

249 mmap(0x7ff420adf000, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1c000) = 0x7ff420adf000

249 mmap(0x7ff420ae1000, 13432, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7ff420ae1000

249 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

249 mmap(NULL, 2036952, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7ff4208d0000

249 mmap(0x7ff4208f5000, 1540096, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x25000) = 0x7ff4208f5000

249 mmap(0x7ff420a6d000, 303104, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x19d000) = 0x7ff420a6d000

249 mmap(0x7ff420ab8000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1e7000) = 0x7ff420ab8000

249 mmap(0x7ff420abe000, 13528, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7ff420abe000

249 mmap(NULL, 12288, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7ff4208cd000

249 arch\_prctl(ARCH\_SET\_FS, 0x7ff4208cd740) = 0

249 mmap(NULL, 4096, PROT\_READ|PROT\_WRITE, MAP\_SHARED, 5, 0) = 0x7ff420b27000

249 mmap(NULL, 4096, PROT\_READ|PROT\_WRITE, MAP\_SHARED, 3, 0) = 0x7ff420afa000

249 mmap(NULL, 8, PROT\_READ|PROT\_WRITE, MAP\_SHARED, 4, 0) = 0x7ff420af9000

249 mmap(NULL, 4, PROT\_READ|PROT\_WRITE, MAP\_SHARED, 6, 0) = 0x7ff420af8000

249 exit\_group(0) = ?

249 +++ exited with 0 +++

248 <... wait4 resumed>NULL, 0, NULL) = 249

248 --- SIGCHLD {si\_signo=SIGCHLD, si\_code=CLD\_EXITED, si\_pid=249, si\_uid=1000, si\_status=0, si\_utime=0, si\_stime=0} ---

248 write(1, "1.125000 1.142857 1.166667 1.200"..., 72) = 72

248 exit\_group(0) = ?

248 +++ exited with 0 +++

1. **Выводы**

В данной лабораторной работе был рассмотрен и применён механизм межпроцессорного взаимодействия при помощи отображаемых файлов. 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.

Недостатком можнет послужить тот факт, что дочерние процессы тоже должны отображать файлы перед началом работы.