Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №4 по курсу**

**«Операционные системы»**

Студент: Шубин Г. С.

Группа: М8О–308Б–19

Вариант: 11

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.

Child1 переводит строки в верхний регистр. Child2 убирает все задвоенные пробелы.

**Общие сведения о программе**

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

1. **mmap** – создает отображение файла в память.
2. **munmap** – снимает отображение.
3. **open** – открывает файл.
4. **close –** закрывает файл.
5. **sem\_init –** инициализация семафора.
6. **sem\_wait** – ожидание доступа, если значение семафора отрицательное, то вызывающий поток блокируется до тех пор, пока один из потоков не вызовет sem\_post.
7. **sem\_post** – увеличивает значение семафора и разблокирует ожидающие потоки.
8. **sem\_destroy** – уничтожает семафор.
9. **read** – чтение из файла в буфер.
10. **write** – запись из буфера в файл.
11. **sleep** – переход в режим ожидания на указанное количество секунд.
12. **exit** – завершение работы программы с некоторым статусом.

**Общий метод и алгоритм решения**.

Для реализации поставленной задачи необходимо:

1. Изучить работу с отображением файла в память (mmap и munmap).
2. Изучить работу с процессами(fork).
3. Создать 2 дочерних и 1 родительский процесс.
4. В каждом процессе отобразить файл в память, преобразовать в соответствии с вариантом и снять отображение (mmap, munmap).

**Листинг программы**

**lab4.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>

#include <semaphore.h>

void remove\_spaces(char\* src, int size)

{

    char\* res = malloc(size\*sizeof(char));

    int j = 0;

    bool flag = true;

    for(int i = 0; i < size; ++i)

    {

        if(src[i] == ' ' && flag)

        {

            flag = false;

            src[i] = '\t';

        }

        else if(src[i] != ' ') { flag = true; }

    }

    for(int i = 0; i < size; ++i)

    {

        if(src[i] != ' ' && src[i] != '\t')

        {

            res[j] = src[i];

            ++j;

        }

        else if(src[i] == '\t')

        {

            res[j] = ' ';

            ++j;

        }

    }

    res = realloc(res, j\*sizeof(char));

    for(int i = 0; i < size; ++i)

    {

        if(i < j) { src[i] = res[i]; }

        else { src[i] = ' '; }

    }

    src = realloc(src, j\*sizeof(char));

    free(res);

}

int main(int argc, char\* argv[])

{

    if(argc != 3)

    {

        printf("INVALID COUNT OF ARGS\nUSAGE: %s <file>\n", argv[0]);

        exit(-1);

    }

    int fd\_0 = -1;

    int fd\_1 = -1;

    char\* src;

    struct stat statbuf;

    if((fd\_0 = open(argv[1], O\_RDWR)) < 0)

    {

        printf("OPEN ERROR\n");

        exit(-1);

    }

    if((fd\_1 = open(argv[2], O\_CREAT | O\_RDWR, S\_IRUSR | S\_IWUSR)) < 0)

    {

        printf("OPEN ERROR\n");

        exit(-1);

    }

    if(fstat(fd\_0, &statbuf) < 0)

    {

        printf("FSTAT ERROR\n");

        exit(-1);

    }

    if(ftruncate(fd\_1, statbuf.st\_size) < 0)

    {

        printf("FTRUNCATE ERROR\n");

        exit(-1);

    }

    char buff[statbuf.st\_size];

    if(read(fd\_0, buff, statbuf.st\_size) != statbuf.st\_size)

    {

        printf("READ ERROR\n");

        exit(-1);

    }

    if(write(fd\_1, buff, statbuf.st\_size) != statbuf.st\_size)

    {

        printf("READ ERROR\n");

        exit(-1);

    }

    int pid\_0 = 0;

    int pid\_1 = 0;

    int status\_0 = 0;

    int status\_1 = 0;

    sem\_t semaphore;

    sem\_init(&semaphore, 0, 1);

    if((pid\_0 = fork()) > 0)

    {

        if((pid\_1 = fork()) > 0)

        {

            sem\_wait(&semaphore);

            sleep(2);

            waitpid(pid\_1, &status\_1, WNOHANG);

            waitpid(pid\_0, &status\_0, WNOHANG);

            src = (char\*)mmap(0, statbuf.st\_size, PROT\_READ, MAP\_SHARED, fd\_1, 0);

            if(src == MAP\_FAILED)

            {

                printf("MMAP ERROR\n");

                exit(-1);

            }

            for(int i = 0; i < statbuf.st\_size; ++i) { printf("%c", src[i]); }

            printf("\n");

            if(munmap(src, statbuf.st\_size) != 0)

            {

                printf("MUNMAP ERROR\n");

                exit(-1);

            }

            sleep(2);

            sem\_post(&semaphore);

        }

        else if(pid\_1 == 0)

        {

            sem\_wait(&semaphore);

            sleep(1);

            src = (char\*)mmap(0, statbuf.st\_size, PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd\_1, 0);

            if(src == MAP\_FAILED)

            {

                printf("MMAP ERROR\n");

                exit(-1);

            }

            remove\_spaces(src, statbuf.st\_size);

            if(munmap(src, statbuf.st\_size) != 0)

            {

                printf("MUNMAP ERROR\n");

                exit(-1);

            }

            sleep(1);

            sem\_post(&semaphore);

        }

        else

        {

            printf("FORK ERROR 1\n");

            exit(-1);

        }

    }

    else if (pid\_0 == 0)

    {

        sem\_wait(&semaphore);

        sleep(1);

        src = (char\*)mmap(0, statbuf.st\_size, PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd\_1, 0);

        if(src == MAP\_FAILED)

        {

            printf("MMAP ERROR\n");

            exit(-1);

        }

        for(int i = 0; i < statbuf.st\_size; ++i) { src[i] = toupper(src[i]); }

        if(munmap(src, statbuf.st\_size) != 0)

        {

            printf("MUNMAP ERROR\n");

            exit(-1);

        }

        sleep(1);

        sem\_post(&semaphore);

    }

    else

    {

        printf("FORK ERROR 2\n");

        exit(-1);

    }

    sem\_destroy(&semaphore);

    close(fd\_0);

    close(fd\_1);

    return 0;

}

**Пример работы**

gregory@DESKTOP-7L8SUG4:~/labsOS/lab4$ ./a.out test.txt res.txt

**test.txt:**

text TEXT

TmP tMp

tt chECK

str caps

**res.txt:**

TEXT TEXT!

TMP TMP

TT CHECK

STR CAPS

**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, за счет отсутствия постоянных вызовов 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.