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: Шандрюк Пётр Николаевич
Группа: М8О-208Б-20
Вариант: 11
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2021

**Содержание**

1. Репозиторий
1. Постановка задачи
1. Общие сведения о программе
1. Общий метод и алгоритм решения
1. Исходный код
1. Демонстрация работы программы
1. Выводы

## Репозиторий

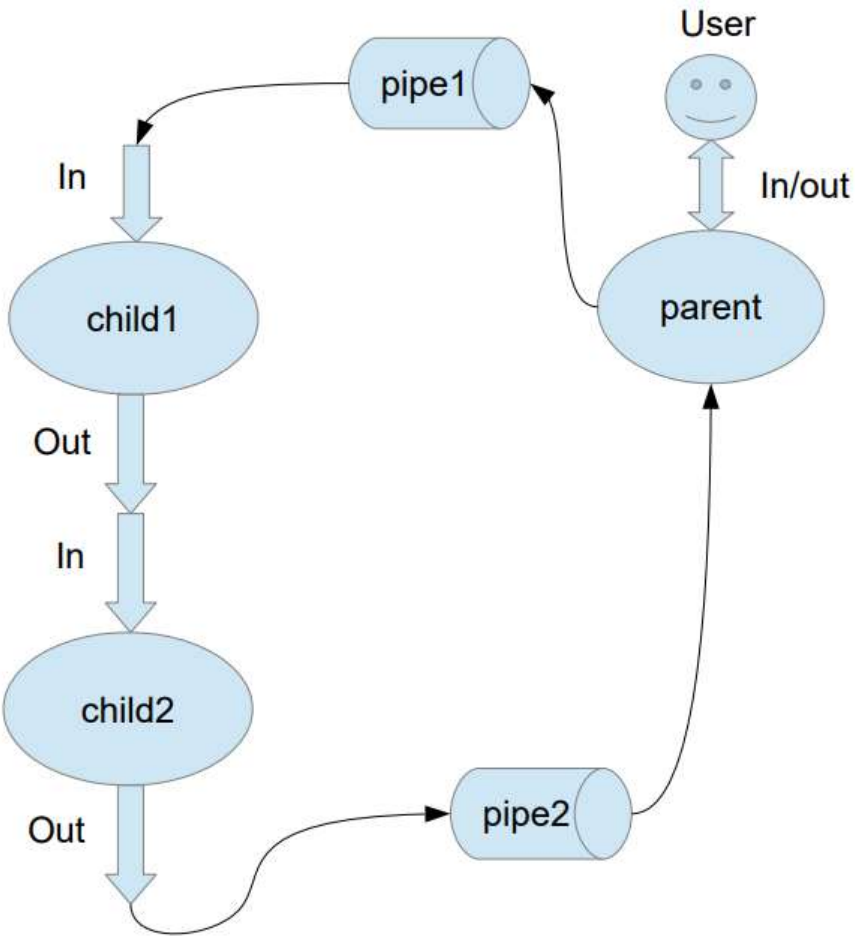
https://github.com/Peter1811/OS/tree/main/lab5

## 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.



12 вариант) Child1 переводит строки в верхний регистр. Child2 преобразует все пробелы в знаки нижнего подчеркивания

## Общие сведения о программе

Программа компилируется из файла main.cpp. 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. shm_open - создаёт/открывает объекты общей памяти POSIX.
2. sem_open - инициализирует и открывает именованный семафор.
3. ftruncate - обрезает файл до заданного размера.
4. mmap, munmap - отображает файлы или устройства в памяти, или удаляет их отображение.
5. memset - заполнение памяти значением определённого байта.
6. sem_getvalue - возвращает значение семафора.
7. close - закрывает файловый дескриптор.
8. sem_close - закрывает именованный семафор.
9. execl - запуск файла на исполнение.
10. sem_getvalue - возвращает значение семафора.
11. sem_wait - блокирует семафор.
12. sem_post - разблокирует семафор.

## 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).

## Исходный код

main.cpp

```
#include <unistd.h>
#include <iostream>
#include <string>
#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>

using namespace std;

void change_spaces(char* src, int size) {
    int j = 0;
    for(int i = 0; i < size ; ++i) {
        if(src[i] == ' ')
        src[i] = '_';
    }
}
```

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

    if(argc != 2) {
        printf("INVALID COUNT OF ARGS\nUSAGE: %s <file>\n", argv[0]);
        exit(-1);
    }

    int fd_0 = -1;
    int fd_1 = -1;
    int fd_2 = -1;
    int fd_3 = -1;
    char* src;
    char* src2;
    char* src3;
    char* src4;
    char* src5;
    struct stat statbuf;
    string s;

    FILE* f1 = fopen("file.txt", "r");
    fd_0 = fileno(f1);
    // printf("%d", fd_1);
    char c;

    if((fd_1 = open("file1.txt", O_CREAT | O_RDWR, S_IRUSR | S_IWUSR)) <
0) {
        printf("OPEN ERROR\n");
        exit(-1);
    }
```

```
    if((fd_2 = open("file2.txt", O_CREAT | O_RDWR, S_IRUSR | S_IWUSR)) <
0) {
        printf("OPEN ERROR\n");
        exit(-1);
    }

    if((fd_3 = open("file3.txt", 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);
    }

    if(ftruncate(fd_2, statbuf.st_size) < 0) {
        printf("FTRUNCATE ERROR\n");
        exit(-1);
    }

    if(ftruncate(fd_3, statbuf.st_size) < 0) {
        printf("FTRUNCATE ERROR\n");
        exit(-1);
```

```
}


// char buff[statbuf.st_size];
char* buff = (char*) malloc(sizeof(char)*statbuf.st_size);

if(read(fd_0, buff, statbuf.st_size) != statbuf.st_size) {
    printf("READ ERROR1\n");
    exit(-1);
}

if(write(fd_1, buff, statbuf.st_size) != statbuf.st_size) {
    printf("READ ERROR\n");
    exit(-1);
}
// for (int i = 0; i < statbuf.st_size; i++) printf("%c", buff[i]);


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) { //Parent
    if((pid_1 = fork()) > 0) { //Parent
        // sem_wait(&semaphore);
        sleep(2);
```

```
            waitpid(pid_1, &status_1, WNOHANG);
            waitpid(pid_0, &status_0, WNOHANG);
            src5 = (char*)mmap(0, statbuf.st_size, PROT_READ, MAP_SHARED,
fd_3, 0);
            if(src5 == MAP_FAILED) {
                printf("MMAP ERROR5\n");
                exit(-1);
            }

            for(int i = 0; i < statbuf.st_size; ++i) { printf("%c",
src5[i]); }
            printf("\n");

            if(munmap(src5, statbuf.st_size) != 0) {
                printf("MUNMAP ERROR\n");
                exit(-1);
            }
            // sleep(2);
            // sem_post(&semaphore);
        }

        else if(pid_1 == 0) { //Child2
            sem_wait(&semaphore);
            // sleep(1);
            src3 = (char*)mmap(0, statbuf.st_size, PROT_READ | PROT_WRITE,
MAP_SHARED, fd_2, 0);
            if(src3 == MAP_FAILED) {
                printf("MMAP ERROR3\n");
                exit(-1);
            }
```

```
            src4 = (char*)mmap(0, statbuf.st_size, PROT_READ | PROT_WRITE,
MAP_SHARED, fd_3, 0);
            if(src4 == MAP_FAILED) {
                printf("MMAP ERROR4\n");
                exit(-1);
            }
            change_spaces(src3, statbuf.st_size);
            for (int i = 0; i < statbuf.st_size; i++) src4[i] = src3[i];
            if(munmap(src3, statbuf.st_size) != 0) {
                printf("MUNMAP ERROR\n");
                exit(-1);
            }
            if(munmap(src4, statbuf.st_size) != 0) {
                printf("MUNMAP ERROR3\n");
                exit(-1);
            }
            // sleep(1);
            sem_post(&semaphore);
        }
        else {
            printf("FORK ERROR 1\n");
            exit(-1);
        }
    }
    else if (pid_0 == 0) { //Child1
        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 ERROR1\n");
```

```
            exit(-1);
        }
        src2 = (char*)mmap(0, statbuf.st_size, PROT_WRITE, MAP_SHARED,
fd_2, 0);
        if(src2 == MAP_FAILED) {
            printf("MMAP ERROR2\n");
            exit(-1);
        }
        for(int i = 0; i < statbuf.st_size; ++i) { src2[i] =
toupper(src[i]); }
        if(munmap(src, statbuf.st_size) != 0) {
            printf("MUNMAP ERROR\n");
            exit(-1);
        }
        if(munmap(src2, statbuf.st_size) != 0) {
            printf("MUNMAP ERROR\n");
            exit(-1);
        }
        // sleep(1);
        sem_post(&semaphore);
    }
    else {
        printf("FORK ERROR 2\n");
        exit(-1);
    }

    remove("file1.txt");
    remove("file2.txt");
    remove("file3.txt");
    sem_destroy(&semaphore);
    close(fd_0);
```

```
    close(fd_1);
    close(fd_2);
    close(fd_3);
    free(buff);
    return 0;
}
```

## Демонстрация работы программы

Для удобства исходные строки записываются в файл file.txt

```
peter@DESKTOP-V53N291:$ cat test.txt
hhadghfdf fdgfdgf
sdfgsdg sdgfsg
sdfgsgsg sdfgdsg
peter@DESKTOP-V53N291:$ gcc -pthread -lrt main.cpp
peter@DESKTOP-V53N291:$ ./a.out
HHADGHFDF_FDGFDGF
SDFGSDG_SDGFSG____
SDFGSGSG_SDFGDSG__________
```

## Выводы

В C++ 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.