

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: Ползикова Алина Владимировна  
Группа: М8О-208Б-21  
Вариант: 5  
Преподаватель: Миронов Евгений Сергеевич  
Оценка: \_\_\_\_\_  
Дата: \_\_\_\_\_  
Подпись: \_\_\_\_\_

Москва, 2022

## **Содержание**

1. Репозиторий
2. Постановка задачи
3. Общие сведения о программе
4. Общий метод и алгоритм решения
5. Исходный код
6. Демонстрация работы программы
7. Выводы

## Репозиторий

[https://github.com/polzzzik/lab\\_os](https://github.com/polzzzik/lab_os)

## Постановка задачи

### Цель работы

Приобретение практических навыков в:

1. Освоение принципов работы с файловыми системами
2. 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) должна создать для решения 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 родительского процесса компилируется из `file_mapping.c`, использует заголовочные файлы `stdio.h`, `stdlib.h`, `unistd.h`, `sys/mman.h`, `fcntl.h`, `semaphore.h`, `string.h`, `errno.h`. В программе используются следующие системные вызовы:

1. `unlink()` – удаление имени из файловой системы
2. `fork()` – создание дочернего процесса
3. `open()` – открытие файла
4. `close()` – закрытие файла
5. `write()` – запись последовательности байт
6. `lseek()` - установка смещения в файловом дескрипторе
7. `mmap()` - создание отражения файла в памяти
8. `munmap()` - удаление отражения файла в памяти

## Общий метод и алгоритм решения

Родительский процесс крутится в бесконечном цикле, пока не получит на вход пустую строку — знак завершения работы. Аналогично в цикле находится и дочерний процесс — обработчик чисел. Синхронизация процессов достигается по средствам 2 семафоров, так после прочтения числа и записи её в образ файла родительский процесс открывает семафор 1 и начинает ждать открытия семафора 2. Открытие семафора 1 позволяет дочернему процессу обработать число, записать результат в образ второго файла, открыть семафор 2 и закрыть семафор 1. Тем самым продолжается работа родительского процесса, который считывает результат из образа второго файла и выводит ошибку, если она была.

## Исходный код

### Lab4.c

```
#include "stdio.h"
#include "stdlib.h"
#include "unistd.h"
#include "fcntl.h"
#include "sys/mman.h"
#include "string.h"
#include "errno.h"
#include "semaphore.h"
#include "signal.h"

#include "lab4.h"

int IsPrime(long long n) {
    if (n <= 1) {
        return 1;
    }
    for (long long i = 2; i*i <= n; i++) {
        if (n%i == 0) {
            return 0;
        }
    }
    return 1;
}

int ParentRoutine(FILE* stream)
{
    const int SIZE = sizeof(long long);
    unlink("file1");
    unlink("file2");
    int file1 = open("file1", O_RDWR | O_CREAT, S_IRUSR | S_IWUSR);
    int file2 = open("file2", O_RDWR | O_CREAT, S_IRUSR | S_IWUSR);
    if ( file1 == -1 || file2 == -1 ) {
        perror("open error");
        return EXIT_FAILURE;
    }
    if ( ftruncate(file1, SIZE) == -1 ) {
        perror("ftruncate");
        return EXIT_FAILURE;
    }
    if ( ftruncate(file2, SIZE) == -1 ) {
        perror("ftruncate");
        return EXIT_FAILURE;
    }

    sem_t* sem1 = sem_open("semaphore1", O_CREAT, S_IRUSR | S_IWUSR, 0);
    sem_t* sem2 = sem_open("semaphore2", O_CREAT, S_IRUSR | S_IWUSR, 0);

    if (sem1 == SEM_FAILED || sem2 == SEM_FAILED){
        perror("sem_open error");
        return EXIT_FAILURE;
    }
    int id = fork();
    if ( id == -1 ) {
        perror("Parent: fork error");
        return EXIT_FAILURE;
    }
}

// child
if (id == 0)
{
    void* in = mmap(NULL, SIZE, PROT_READ, MAP_SHARED, file1, 0);
    void* ans = mmap(NULL, SIZE, PROT_WRITE, MAP_SHARED, file2, 0);
    if (in == MAP_FAILED || ans == MAP_FAILED)
    {
```

```

        perror("mmap error");
        return EXIT_FAILURE;
    }
    unlink("result.txt");
    int fout = open("result.txt", O_CREAT | O_WRONLY, S_IRUSR);
    if (fout == -1)
    {
        perror("open error");
        return EXIT_FAILURE;
    }

    if (dup2(fout, 1) == -1) {
        perror("Child: dup error");
        return EXIT_FAILURE;
    }
    while (1)
    {
        sem_wait(sem1);
        long long num;
        memcpy(&num, in, sizeof(long long));
        if (IsPrime(num) == 1)
        {
            int k = 404;
            memcpy(ans, &k, sizeof(long long));
            sem_post(sem2);
            munmap(in, SIZE);
            munmap(ans, SIZE);
            close(fout);
            break;
        }
        else
        {
            printf("%lld\n", num);
            fflush(stdout);
            sem_post(sem2);
        }
    }
    munmap(in, SIZE);
    munmap(ans, SIZE);
    close(fout);
}

// parent
else
{
    void* out = mmap(NULL, SIZE, PROT_WRITE, MAP_SHARED, file1, 0);
    void* ans = mmap(NULL, SIZE, PROT_READ, MAP_SHARED, file2, 0);
    if (out == MAP_FAILED || ans == MAP_FAILED)
    {
        perror("mmap error");
        return EXIT_FAILURE;
    }
    long long number;
    char* str;
    size_t s = 0;
    int n = getline(&str, &s, stream);
    while ( n > 0)
    {
        number = atol(str);
        memcpy(out, &number, sizeof(long long));
        sem_post(sem1);
        sem_wait(sem2);
        int k;
        memcpy(&k, ans, sizeof(int));
        if (k != 0)
        {
            break;
        }
    }
}

```

```

        n = getline(&str, &s, stream);
    }
    kill(id, SIGKILL);
    free(str);
    munmap(out, SIZE);
    munmap(ans, SIZE);
    sem_close(sem1);
    sem_close(sem2);
    close(file1);
    close(file2);
    unlink("file1");
    unlink("file2");
}
return EXIT_SUCCESS;
}

```

## Демонстрация работы программы

```

OS-labs/build/lab4$ ./lab4
80
100000000000000000
1234
1337
13
OS-labs/build/lab4$ cat result.txt
80
100000000000000000
1234
1337

```

## Выводы

Составлена и отлажена программа на языке Си, осуществляющая работу и взаимодействие между процессами с использованием отображаемых файлов. Так, получены навыки в обеспечении обмена данных между процессами посредством технологии «File mapping».