

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: Клитная Анастасия  
Викторовна  
Группа: М8О-208Б-20  
Вариант: 20  
Преподаватель: Миронов Евгений Сергеевич  
Оценка: \_\_\_\_\_  
Дата: \_\_\_\_\_  
Подпись: \_\_\_\_\_

Москва, 2021

## **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/klitnaya/OS>

## Постановка задачи

Задача: реализовать программу, в которой родительский процесс создает два дочерних процесса. Родительский процесс принимает строки, которые отправляются в тот или иной дочерний процесс в зависимости от следующего правила: если длина строки больше 10 символов, то строка отправляется во второй дочерний процесс, в противном случае в первый дочерний процесс. Оба процесса удаляют гласные из строк.

Межпроцессорное взаимодействие осуществляется посредством отображаемых файлов (memory-mapped files).

## Общие сведения о программе

Для реализации поставленной задачи нам нужны следующие библиотеки:

<unistd.h> - для работы с системными вызовами в Linux.

<stdlib.h> - для того, чтобы можно было пользоваться функциями, отвечающими за работу с памятью.

<limits.h> - для определения характеристик общих типов переменных.

<sys/mman.h> - для работы с memory-mapped files.

<pthread.h> - для работы с потоками.

<ctype.h> - для классификации и преобразования отдельных символов.

<sys/stat.h> - для доступа к файлам.

<fcntl.h> - для работы с файловым дескриптором.

<sys/wait.h> - для использования символических констант.

<fstream> - для работы с файлами C++.

<string.h> - для использования функций над строками.

<stdio.h> - для использования взаимодействия с физическими устройствами (клавиатура и т.д)

<iostream> - использования потока ввода и вывода

<signal.h> - для указания того, как программа обрабатывает сигналы во время ее выполнения

<sstream> - для организации работы со строками

Данная лабораторная работа сделана на основе второй лабораторной работы, посвященной работе с процессами. Для работы с memory-mapped files согласно заданию помимо основы второй лабораторной работы и использования специальных библиотек у меня в программе также есть использование следующих системных вызовов:

mmap(...) - системный вызов, позволяющий выполнить отображение файла или устройства на память. принимающий следующие аргументы: адрес памяти для размещения, текущий размер файла, права на чтение и запись, права на то, чтобы делиться данным маппингом, сам файловый дескриптор и начальную позицию, с которого пойдет считывание).

munmap(...) - системный вызов, удаляющий маппинг из адресного пространства.

ftruncate(filedesc, size\_t bites) - системный вызов, увеличивающий память файла до size\_t bites.

### **Общий метод и алгоритм решения**

С самого начала выполнения программы требуется 2 названия для дочерних процессов - куда они будут писать строки без гласных.

Далее создаются 2 файла: f1.txt и f2.txt. Это те самые файлы, куда мы посредством file-mapping будем писать файлы для потомков. Строки длиной меньше-равно 10 будут идти в f1.txt, иначе в f2.txt. При этом посредством

системного вызова `ftruncate` память всегда будет увеличиваться динамически после добавления каждой строки.

После считывания всех строк дочерние процессы принимают из `map-files` строки и удаляют в них гласные, выводя строки без гласных в каждый из своих файлов. После завершения работы `mapped-files` удаляются из памяти при помощи системного вызова `munmap`.

Собирается программа при помощи команды `g++ lab4.cpp -o main`, запускается при помощи команды `./main`.

### **Исходный код**

```
#include <iostream>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <fcntl.h>
#include <ctype.h>
#include <sys/mman.h>
#include <sys/wait.h>
#include <sys/stat.h>
#include <string.h>
#include <fstream>
#include <string>
#include <sstream>
#include <signal.h>
#include <pthread.h>
#define ll long long
int main() {
    int a;
```

```

    std::cout << "Congrats, you are in parent process. Please enter amount of
strings: " << std::endl;

    std::cin >> a;

    int less_than_ten = 0;
    int more_than_ten = 0;
    int first_pos = 0;
    int second_pos = 0;
    int first_length = 0;
    int second_length = 0;
    int fd1;
    int fd2;
    std::fstream fs;
    std::string path_child1, path_child2;
    std::cout << "Enter name of file for first child: " << std::endl;
    std::cin >> path_child1;
    std::cout << "For second child: " << std::endl;
    std::cin >> path_child2;
    std::string string;
    if ((fd1 = open("f1.txt", O_RDWR| O_CREAT, 0777)) == -1)
    {
        std::cout << "Error: can not open the f1.txt. Try again later." << std::endl;
        exit(EXIT_FAILURE);
    }
    if ((fd2 = open("f2.txt", O_RDWR| O_CREAT, 0777)) == -1)
    {
        std::cout << "Error: can not open the f2.txt. Try again later." << std::endl;
        exit(EXIT_FAILURE);
    }

```

```
char *mapped_file1 = (char *)mmap(nullptr, getpagesize(), PROT_READ |  
PROT_WRITE, MAP_SHARED, fd1, 0); // при помощи мемори маппа  
отображаем mapped file на оперативную память
```

```
char *mapped_file2 = (char *)mmap(nullptr, getpagesize(), PROT_READ |  
PROT_WRITE, MAP_SHARED, fd2, 0); // при помощи мемори маппа  
отображаем mapped file на оперативную память
```

```
if (mapped_file1 == MAP_FAILED)  
{  
    std::cout << "An error with mmap function one has been detected" <<  
std::endl;  
    exit(EXIT_FAILURE);  
}  
if (mapped_file2 == MAP_FAILED)  
{  
    std::cout << "An error with mmap function two has been detected" <<  
std::endl;  
    exit(EXIT_FAILURE);  
}  
std::cout << "Good. Please enter your strings: " << std::endl;  
while (a > 0)  
{  
    std::cin >> string;  
    string = string + "\n";  
    if (string.size() <= 10)  
    {  
        less_than_ten++;  
        first_length += string.size();  
        if (ftruncate(fd1, first_length))  
        {  
            std::cout << "Error during ftruncate with mf1 has been detected" <<  
std::endl;
```

```

        exit(EXIT_FAILURE);
    }
    for (int i = 0; i < string.size(); ++i)
    {
        mapped_file1[first_pos++] = string[i];
    }
}
else
{
    more_than_ten++;
    second_length += string.size();
    if (ftruncate(fd2, second_length))
    {
        std::cout << "Error during ftruncate with mf2 has been detected" <<
std::endl;
        exit(EXIT_FAILURE);
    }
    for (int i = 0; i < string.size(); ++i)
    {
        mapped_file2[second_pos++] = string[i];
    }
}
a--;
}
int first_identificator = fork();
if (first_identificator == -1)
{
    std::cout << "Fork error!" << std::endl;
    exit(EXIT_FAILURE);
}

```



```

}
else if (first_identificator == 0)
{
    fs.open(path_child1, std::fstream::in | std::fstream::out | std::fstream::app);
    if (!fs.is_open())
    {
        exit(EXIT_FAILURE);
    }
    std::cout << "Congrats, you are in child #1 process" << std::endl;
    int i = 0;
    while (less_than_ten > 0)
    {
        std::string string;
        while (mapped_file1[i] != '\n')
        {
            string += mapped_file1[i];
            i++;
        }
        if (mapped_file1[i] == '\n')
            i++;

        for (int i = 0; i < string.size()/2; i++){ //invert
            char tmp = string[i];
            string[i] = string[string.size()-i-1];
            string[string.size()-i-1] = tmp;
        }
        fs << string << std::endl;
        less_than_ten--;
    }
}

```

```

}
else
{
    int second_identificator = fork();
    if (second_identificator == -1)
    {
        std::cout << "Fork error!" << std::endl;
        return 4;
    }
    else if (second_identificator == 0)
    {
        fs.open(path_child2, std::fstream::in | std::fstream::out | std::fstream::app);
        if (!fs.is_open())
        {
            exit(EXIT_FAILURE);
        }
        std::cout << "Congrats, you are in child2 process" << std::endl;
        int i = 0;
        while (more_than_ten > 0)
        {
            std::string string;
            while (mapped_file2[i] != '\n')
            {
                string += mapped_file2[i];
                i++;
            }
            if (mapped_file2[i] == '\n')
                i++;
            int x = 0;

```

```

        for (int i = 0; i<string.size()/2; i++){ //invert
            char tmp = string[i];
            string[i] = string[string.size()-i-1];
            string[string.size()-i-1] = tmp;
        }
        fs << string << std::endl;
        more_than_ten--;
    }
}
else
{
    if (munmap(mapped_file1, getpagesize()) == -1)
    {
        std::cout << "Munmap1 error has been dedcted!" << std::endl;
        exit(EXIT_FAILURE);
    }
    if (munmap(mapped_file2, getpagesize()) == -1)
    {
        std::cout << "Munmap2 error has been dedcted!" << std::endl;
        exit(EXIT_FAILURE);
    }
    close(fd1);
    close(fd2);
    remove("f1.txt");
    remove("f2.txt");
    return 0;
}
}
}

```

## Демонстрация работы программы

```
root@Owl:/mnt/c/Users/Настя# g++ -fsanitize=address lab44.cpp -o lab4
```

```
root@Owl:/mnt/c/Users/Настя# ./lab4
```

Congrats, you are in parent process. Please enter name of first child:

t

For second child:

r

```
root@Owl:/mnt/c/Users/Настя# vi r
```

```
root@Owl:/mnt/c/Users/Настя# vi t
```

```
root@Owl:/mnt/c/Users/Настя# vi test.txt
```

```
root@Owl:/mnt/c/Users/Настя# ./lab4
```

test:

aaaaaaaasssssssss

sssdd

f

ddffffffffddddd

uuuuuuuuuuud

T;

ddsss

f

R:

ssssssssaaaaaaa

hddddffffffffdd

duuuuuuuuuuu

## Выводы

Данная лабораторная работа, на мой взгляд, служит отличным дополнением

ко второй лабораторной работе. Благодаря поставленному заданию я расширил свой функционал работы с процессами и освоил принцип реализации file-mapping.