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: Знай Артемий Олегович
Группа: М8О-201Б-21
Вариант: 18
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2022

**Содержание**

## Репозиторий

https://github.com/znako/OS_LABS/

## 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) 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.

## Общие сведения о программе

Программа родительского процесса компилируется из parent.c, использует заголовочные файлы parent.h, sys/mman.h, semaphore.h, signal.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. mmap() - создание отражения файла в памяти
7. munmap() - удаление отражения файла в памяти

## Общий метод и алгоритм решения

Родительский процесс получает строки, определяет в какой дочерний ее отправить, с помощью маппирования отображает строку в соответствующий сегмент данных. Дочерний блокирует семафор в процессе считывания данных, проводит фильтрацию гласных букв и выводит строку в свой файл.

## Исходный код

**Lab4.c**

```
#include "parent.h"
#include "sys/mman.h"
#include "semaphore.h"
#include "signal.h"
```

```
char* ReadString(FILE* stream)
{
    if(feof(stream)) {
        return NULL;
    }

    const int chunkSize = 256;
    char* buffer = (char*)malloc(chunkSize);
    int bufferSize = chunkSize;

    if(buffer == NULL)
    {
        printf("Couldn't allocate buffer");
        exit(EXIT_FAILURE);
    }

    int readChar;
    int idx = 0;

    while ((readChar = getc(stream)) != EOF)
    {
        buffer[idx++] = readChar;

        if (idx == bufferSize)
        {
            buffer = realloc(buffer, bufferSize + chunkSize);
            bufferSize += chunkSize;
        }

        if (readChar == '\n') {
            break;
        }
    }

    buffer[idx] = '\0';

    return buffer;
}

int ChoosePipe(int len){
    return (len - 1) % 2 != 0;
}

void ParentRoutine(FILE* fin)
{
    const int STR_LEN = 128;
    const int MAPPED_SIZE = 512;

    char* fileName1 = ReadString(fin);
    char* fileName2 = ReadString(fin);

    fileName1[strlen(fileName1) - 1] = '\0';
    fileName2[strlen(fileName2) - 1] = '\0';

    unlink(fileName1);
    unlink(fileName2);

    int outputFile1, outputFile2;

    if ((outputFile1 = open(fileName1, O_WRONLY | O_CREAT, S_IRWXU)) < 0)
    {
        perror("opening output file 1 error )");
        exit(EXIT_FAILURE);
    }

    if ((outputFile2 = open(fileName2, O_WRONLY | O_CREAT, S_IRWXU)) < 0)
    {
        perror("opening output file 2 error )");
```

```
        exit(EXIT_FAILURE);
    }
    free(fileName1);
    free(fileName2);

    const char *sem_file1 = "semaphore1";
    const char *sem_file2 = "semaphore2";

    sem_t *semaphore1 = sem_open(sem_file1, O_CREAT, S_IRUSR | S_IWUSR, 0);
    if (semaphore1 == SEM_FAILED) {
        perror("Semaphore1 error");
        exit(1);
    }

    sem_t *semaphore2 = sem_open(sem_file2, O_CREAT, S_IRUSR | S_IWUSR, 0);
    if (semaphore2 == SEM_FAILED) {
        perror("Semaphore2 error");
        exit(2);
    }

    char* mapped1 = (char *)mmap(0, MAPPED_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED |
MAP_ANONYMOUS, -1, 0);
    char* mapped2 = (char *)mmap(0, MAPPED_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED |
MAP_ANONYMOUS, -1, 0);

    pid_t pid1 = fork();
    pid_t pid2 = 1;

    if (pid1 > 0){
        pid2 = fork();
    }

    if (pid1 < 0 || pid2 < 0)
    {
        perror("process error");
        exit(EXIT_FAILURE);
    }

    if (pid1 == 0)
    {

        if (dup2(outputFile1, 1) < 0)
        {
            perror("duping output file error");
            exit(EXIT_FAILURE);
        }


        sem_t *semaphore = sem_open(sem_file1, 0);

        while(1)
        {
            sem_wait(semaphore);

            if (mapped1[0] == '\0')
            {
                break;
            }

            int i = 0;

            char* strArray = (char*)malloc(STR_LEN);
            int len = STR_LEN;

            do{

                strArray[i] = mapped1[i];
```

```
            if (i == len)
            {
                strArray = realloc(strArray, len + STR_LEN);
                len += STR_LEN;
            }
        } while (mapped1[i++] != '\n');

        char* vowels = {"AEIOUYaeiouy"};

        for (int j = 0; j < i; ++j)
        {
            int isVowel = 0;

            for (int k = 0; k < (int)strlen(vowels); ++k){
                if (strArray[j] == vowels[k])
                {
                    isVowel = 1;
                    break;
                }
            }

            if (isVowel == 0) {
                write(1, &strArray[j], 1);
            }
        }
        free(strArray);
    }

    close(outputFile1);
}

else if (pid2 == 0)
{
    if (dup2(outputFile2, 1) < 0)
    {
        perror("duping output file error");
        exit(EXIT_FAILURE);
    }

    sem_t *semaphore = sem_open(sem_file2, 0);

    while(1)
    {

        sem_wait(semaphore);

        if (mapped2[0] == '\0')
        {
            break;
        }

        int i = 0;

        char* strArray = (char*)malloc(STR_LEN);
        int len = STR_LEN;

        do{

            strArray[i] = mapped2[i];

            if (i == len)
            {
                strArray = realloc(strArray, len + STR_LEN);
                len += STR_LEN;
            }
        } while (mapped2[i++] != '\n');

        char* vowels = {"AEIOUYaeiouy"};
```

```
            for (int j = 0; j < i; ++j)
            {
                int isVowel = 0;

                for (int k = 0; k < (int)strlen(vowels); ++k){
                    if (strArray[j] == vowels[k])
                    {
                        isVowel = 1;
                        break;
                    }
                }

                if (isVowel == 0) {
                    write(1, &strArray[j], 1);
                }
            }
            free(strArray);
        }
        close(outputFile2);
    }
    else
    {
        int stat_counter1 = 0;
        int stat_counter2 = 0;

        char* strInput = NULL;

        while ((strInput = ReadString(fin)) != NULL)
        {
            int strSize = strlen(strInput);

            if (strSize > 0)
            {
                if (ChoosePipe(strSize))
                {
                    stat_counter1 = 0;

                    for (int j = 0; j < strSize; j++) {
                        mapped1[stat_counter1++] = strInput[j];
                    }

                    sem_post(semaphore1);


                }
                else
                {
                    stat_counter2 = 0;

                    for (int j = 0; j < strSize; j++) {
                        mapped2[stat_counter2++] = strInput[j];
                    }
                    sem_post(semaphore2);
                }
            }

            free(strInput);
        }
        mapped1[0] = '\0';
        mapped2[0] = '\0';
        sem_post(semaphore1);
        sem_post(semaphore2);
    }
    munmap(mapped1, MAPPED_SIZE);
    munmap(mapped2, MAPPED_SIZE);
    sem_close(semaphore1);
    sem_close(semaphore2);
```

```
    sem_unlink(sem_file1);
    sem_unlink(sem_file2);
}
```

## Демонстрация работы программы

```
znako@znako-VirtualBox:~/UtoW/OS_LABS/build/lab4$ ./lab4

o1.txt

o2.txt

akjgsakg;

alasg

akjhwoahjoaw

znako@znako-VirtualBox:~/UtoW/OS_LABS/build/lab4$ cat o1.txt

kjgskg;

lsg

znako@znako-VirtualBox:~/UtoW/OS_LABS/build/lab4$ cat o2.txt

kjhwhjw
```

## 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».