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: Шумилова Александра
Группа: М8О-207Б-21
Вариант: 20
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2022

**Содержание**

## Репозиторий

https://github.com/tonsoleils/OS

## 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.

## Общие сведения о программе

Программа компилируется из файла main.c с помощью CMakeFile.txt. Также используется заголовочные файлы: stdio.h, stdlib.h, sys/types.h, sys/stat.h, sys/mman.h, fcntl.h, unistd.h, string.h, semaphore.h, assert.h, errno.h. В программе используются следующие системные вызовы:

1. mmap(…) – отображение в память (файла/анонимного участка памяти);
2. sem_init(…) – инициализировать семафор;
3. fork(…) – создать дочерний процесс;
4. sem_wait(…) – ожидание семафора
5. sem_post(…) – увеличение счётчика семафора на единицу

## Общий метод и алгоритм решения

Родительский процесс считывает названия двух файлов, открывает их для чтения и создаёт 2 дочерних потока. Затем родительский процесс в цикле (пока не прекратили ввод) считывает строки и, в зависимости от их длины, копирует строку в отображенную память, соответствующую нужному дочернему процессу, после увеличивает счетчик семафора на 1. Дочерний процесс, дождавшись увеличение счётчика соответствующего семафора, копирует из отображённой памяти сроку, производит её реверс и записывает в соответствующий файл.

## Исходный код

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
```

```c
#include <sys/stat.h>
#include <sys/mman.h>
#include <fcntl.h>
#include <unistd.h>
#include <string.h>
#include <semaphore.h>
#include <assert.h>
#include <errno.h>

void reverse(char *string) { // Функция для инвертирования строки
    int length = strlen(string); // получаем длинну строки
    int middle = (length - 1) / 2; // получаем индекс середины строки (игнорируя последний терминальный символ)ы

    char temp;

    for (int i = 0; i < middle; i++) {
        // переставляем местами символы
        temp = string[i];
        string[i] = string[length - i - 2];  // -2 потому что игнорируем последний терминальный символ
        string[length - i - 2] = temp;
    }
}

int main() {
    // отображаем в памяти семафоры
    sem_t *sem1 = mmap(NULL, sizeof(*sem1), PROT_READ|PROT_WRITE, MAP_SHARED|MAP_ANONYMOUS, -1, 0);
    sem_t *sem2 = mmap(NULL, sizeof(*sem2), PROT_READ|PROT_WRITE, MAP_SHARED|MAP_ANONYMOUS, -1, 0);

    // создаём отображение в памяти для первого процесса
    char* mmap1 = mmap(NULL, 512, PROT_READ|PROT_WRITE, MAP_SHARED|MAP_ANONYMOUS, -1, 0);
    if (mmap1 == MAP_FAILED) {
        fprintf(stderr, "Error mmap1 in parent");
        exit(3);
    }

    // и для второго
    char* mmap2 = mmap(NULL, 512, PROT_READ|PROT_WRITE, MAP_SHARED|MAP_ANONYMOUS, -1, 0);
    if (mmap2 == MAP_FAILED) {
        fprintf(stderr, "Error mmap2 in parent");
        exit(4);
    }

    // инициализируем семафор
    // 1 параметр - сам семафор
    // 2 параметр - 0 или 1: если 0 - семафор для потоков в пределах процесса, если 1 - для процессов
    sem_init(sem1, 1, 0);
    sem_init(sem2, 1, 0);

    pid_t pid, pid2; // идентификаторы процессов

    int file_a; // дескриптор файла А
    int file_b; // дескриптор файла Б
```

```
    char file_name_a[256]; // Имя файла A
    char file_name_b[256]; // Имя файла B
    fgets(file_name_a, sizeof file_name_a, stdin); // Считываем первое имя
файла
    fgets(file_name_b, sizeof file_name_b, stdin); // Считываем второе имя
файла
    file_name_a[strcspn(file_name_a, "\n")] = 0; // удаляем символ перевода
каретки
    file_name_b[strcspn(file_name_b, "\n")] = 0;
    file_a = open(file_name_a, O_RDWR | O_CREAT | O_TRUNC, 0777); // откры-
ваем файл для записи
    if (!file_a) {
        fprintf(stderr, "Could not open file\n");
        exit(1);
    }
    file_b = open(file_name_b, O_RDWR | O_CREAT | O_TRUNC, 0777); // откры-
ваем файл для записи
    if (!file_b) {
        fprintf(stderr, "Could not open file\n");
        exit(2);
    }

    pid = fork();

    if (pid < 0) { // если не удалось форкнуть
        fprintf(stderr, "child process A not created\n");
        exit(3);
    } else if (pid == 0) {
        // CHILD 1
        while(1) {
            if (sem_wait(sem1) == 0) { // ждём семафор (счётчик уменьшается
на 1 если дождались)
                char buffer[512]; // если дождались, создаём буффер
                memcpy(buffer, mmap1, strlen(mmap1)); // достаём данные из
отображения
                reverse(buffer); // реверсим
                write(file_a, buffer, strlen(buffer)); // и пишем в файл
            }
        }
    } else {
        pid2 = fork();

        if (pid2 < 0) {
            fprintf(stderr, "child process B not created\n");
            exit(4);
        } else if (pid2 == 0) {
            // CHILD 2
            while(1) {
                if (sem_wait(sem2) == 0) { // всё тоже самое для второго до-
чернего процесса
                    char buffer[512];
                    memcpy(buffer, mmap2, strlen(mmap2));
                    reverse(buffer);
                    write(file_b, buffer, strlen(buffer));
                }
            }
        } else {
            // PARENT
```

```
            char buffer[512]; // заводим буффер, в который будем считывать
            memset(buffer, 0, sizeof(char) * 512); // отчищаем буффер
            while(fgets(buffer, sizeof(char) * 512, stdin) != NULL) { // пока
нам вводят строки
                if (strlen(buffer) > 10) { // если строка больше 10 символов
                    memcpy(mmap1, buffer, strlen(buffer)); // копируем её в
отображение (ммап1)
                    sem_post(sem1); // увеличиваем счётчик семафора на 1
                } else { // всё тоже самое для строк меньше или равно 10 сим-
волов
                    memcpy(mmap2, buffer, strlen(buffer));
                    sem_post(sem2);
                }
            }
        }
    }
    close(file_a); // всё закрываем и освобождаем
    close(file_b);
    sem_destroy(sem1);
    sem_destroy(sem2);
    munmap(sem1, sizeof(sem1));
    munmap(sem2, sizeof(sem2));
    return 0;
}
```

## Демонстрация работы программы

**tonsoleils@LAPTOP-31GE9NQM:/mnt/d/OS/lab1$** ./main

a.txt

b.txt

Run child A

Run child B

abc

def

super long string

another super long string

short

string

**tonsoleils@LAPTOP-31GE9NQM:/mnt/d/OS/lab1$** cat a.txt

cba

fed

trohs

gnirts

6

**tonsoleils@LAPTOP-31GE9NQM:/mnt/d/OS/lab1$** cat b.txt

gnirts gnol repus

gnirts gnol repus rehtona

## Выводы

Проделав данную работу, я научилась принципам работы с файловыми системами и обменау данных между процессами посредством технологии «File mapping». Так же научилась синхронизировать процессы с помощью семафоров.