Московский Авиационный Институт
(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Лабораторная работа №4 по курсу
«Операционные системы»
ПРОЦЕССЫ И ПОТОКИ. ВЗАИМОДЕЙСТВИЕ МЕЖДУ ПОТОКАМИ

Студент: Злобина Валерия Вадимовна
Группа: М8О–208Б–21
Вариант: 20
Преподаватель: Соколов Андрей Алексеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2022.

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.

Вариант 20

Правило фильтрации: строки длины больше 10 символов отправляются в pipe2, иначе в pipe1. Дочерние процессы инвертируют строки.

Общие сведения о программе

Программа компилируется из файла main.c

Исходный код

```
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <signal.h>
#include <unistd.h>
#include <sys/mman.h>
#include <semaphore.h>
#include <errno.h>
#include <stdbool.h>

#define STR_LEN 128
#define STR_MAX_QUANTITY 100

#define BUFFER_SIZE 128

#define MAPPED_SIZE STR_LEN * STR_MAX_QUANTITY

int flaccess = S_IWUSR | S_IRUSR | S_IRGRP | S_IROTH;
```

```
char* read_string(pid_t fd, char mode) {
    char* buffer = calloc(sizeof(char), BUFFER_SIZE);
    char c;
    if (read(fd, &c, 1) <= 0) {
        return NULL;
    }
    if (c == '\0') {
        return NULL;
    }
    int i = 0;
    buffer[i++] = c;
    while (read(fd, &c, 1) > 0){
        if (( c != '\0') && (c != '\n')) {
            buffer[i++] = c;
        }
        else{
            break;
        }
    }
                                            if (mode != 0)  {
                                                    buffer[i++] = '\n';
                                            }

    char *string = calloc(sizeof(char), i);
    strncpy(string, buffer, i);
    free(buffer);
    return string;
}

int child_execute(char* path, char *mapped, const char* sem_file) {

                                            char* filename = path;
                                            int cur_file_des;
    if ((cur_file_des = open(filename, O_WRONLY | O_TRUNC | O_CREAT, S_IRWXU)) < 0) {
        perror("cur_file_des");
        exit(1);
    }
                                            sem_t *semaphore = sem_open(sem_file, 0);
                                            int offset = 0;

                                            while (true) {
                                                    if (sem_wait(semaphore) == -1) {
                                                            perror("Semaphore error");
                                                            exit(EXIT_FAILURE);
                                                    }
                                                    if (mapped[offset] == '\0') {
                                                            break;
                                                    }
                                                    int i = 0;

                                                    char str_array[STR_LEN];
                                                    do {
                                                            str_array[i] = mapped[offset + i];
                                                    } while (mapped[offset + i++] != '\n');
                                                    for (int j = i - 1; j >= 0; j--) {
            write(cur_file_des, &str_array[j], 1);
                                                    }
                                                    offset += i;
                                            }
```

```
                                        return 0;

}

int main()
{
                                        const char *sem_file1 = "a.semaphore";
                                        const char *sem_file2 = "b.semaphore";
    printf("Введите имена файлов: \n");
                                        char* filename1 = read_string(0, 0);
    char* filename2 = read_string(0, 0);

    int filedes1, filedes2;

                                        sem_t *semaphore1 = sem_open(sem_file1, O_CREAT, flaccess, 0);

                                        if (semaphore1 == SEM_FAILED) {
                                                perror("Semaphore error");
                                                exit(1);
                                        }

                                        sem_t *semaphore2 = sem_open(sem_file2, O_CREAT, flaccess, 0);
                                        if (semaphore2 == SEM_FAILED) {
                                                perror("Semaphore 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 f_id1 = fork();

                                        if (f_id1 < 0) {
                                                perror("fork");
                                                exit(3);
                                        }
                                        else if (f_id1 == 0) { //child1
                                                char * childfile = filename1;
                                                child_execute(childfile, mapped1, sem_file1);
                                                return 0;
                                        }

                                        pid_t f_id2 = fork();
                                        if (f_id2 < 0) {
                                                printf("Fork error with code -1 returned in the parent, no
child_2 process is created");
                                                exit(4);
                                        } else if (f_id2 == 0) { //child2
                                                char * childfile = filename2;
                                                child_execute(childfile, mapped2, sem_file2);
                                                return 0;
                                        }

                                        if (f_id1 != 0 && f_id2 != 0){ //parent
                                                printf("\n");
                                                int stat_counter1 = 0;
                                                int stat_counter2 = 0;
                                                char *temp_string;
```

```
                int string_counter = 0;
                while (temp_string = read_string(0, 1)) {
                        if ((string_counter++) > STR_MAX_QUANTITY) {
                                printf("Too many strings\n");
                                exit(EXIT_FAILURE);
                        }
                        int length = strlen(temp_string);
                        if (length > 10) {
                                for (int j = 0; j < length; j++) {
                                        mapped1[stat_counter1 + j] =
temp_string[j];

                                }
                                sem_post(semaphore1);
                                stat_counter1 += length;

                        } else {

                                for (int j = 0; j < length; j++) {
                                        mapped2[stat_counter2 + j] =
temp_string[j];

                                }
                                sem_post(semaphore2);
                                stat_counter2 += length;
                        }
                }
                mapped1[stat_counter1] = '\0';
                mapped2[stat_counter2] = '\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);
        return 0;
}
```

Пример запуска программы: