Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика”

Кафедра №806 “Вычислительная математика и программирование”

Лабораторная работа №3 по курсу

«Операционные системы»

Группа: М8О-206Б-23

Студент: Абдыкалыков Н. А.

Преподаватель: Миронов Е. С.

Оценка: _____

Дата: 28.02.25

Москва, 2025

Постановка задачи

Вариант 6.

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 каналы (pipe). Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы.

В файле записаны команды вида: «число число число». Дочерний процесс считает их сумму и выводит результат в стандартный поток вывода. Числа имеют тип int. Количество чисел может быть произвольным.

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.

Общий метод и алгоритм решения

Общий метод:

1) Родительский процесс запрашивает у пользователя имя файла, содержащего команды (наборы целых чисел).

2) Родитель создаёт объект отображаемой памяти (shared memory) с помощью shm_open, устанавливает его размер и отображает в своё адресное пространство (mmap).

3) Родитель создаёт дочерний процесс с помощью fork() и вызывает внешнюю программу (второй исполняемый файл) через execl() или execv().

4) При запуске дочернего процесса стандартный поток ввода перенаправляется на файл с командами. Дочерний процесс построчно считывает числа, вычисляет их сумму и записывает результат в отображаемую память.

5) Родитель, дождавшись завершения дочернего процесса, считывает результаты из общей памяти и выводит их в консоль.

Код программы

Child.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <unistd.h>

#define SHM_SIZE 1024

int main(int argc, char* argv[]) {
    if (argc < 2) {
        fprintf(stderr, "Не передано имя отображаемого файла\n");
        return 1;
    }

    const char* shm_name = argv[1];

    int shm_fd = shm_open(shm_name, O_RDWR, 0666);
    if (shm_fd == -1) {
        perror("Ошибка при открытии shared memory");
        return 1;
    }

    char* shm_ptr = mmap(0, SHM_SIZE, PROT_WRITE, MAP_SHARED, shm_fd, 0);
    if (shm_ptr == MAP_FAILED) {
        perror("Ошибка при mmap в дочернем процессе");
```

```
        return 1;
    }

    char line[256];
    char result[SHM_SIZE] = "";

    while (fgets(line, sizeof(line), stdin)) {
        int sum = 0;
        char* token = strtok(line, " \t\n");
        while (token != NULL) {
            sum += atoi(token);
            token = strtok(NULL, " \t\n");
        }
        char buffer[64];
        snprintf(buffer, sizeof(buffer), "%d\n", sum);
        strncat(result, buffer, sizeof(result) - strlen(result) - 1);
    }

    strncpy(shm_ptr, result, SHM_SIZE - 1);
    munmap(shm_ptr, SHM_SIZE);

    return 0;
}
```

Parent.c

```
#define _GNU_SOURCE
#include <stdio.h>
#include <stdlib.h>
```

```
#include <fcntl.h>
#include <sys/mman.h>
#include <unistd.h>
#include <sys/wait.h>
#include <string.h>
#include <errno.h>

#define SHM_NAME "/my_shared_memory"
#define SHM_SIZE 1024

int main() {
    char filename[256];
    printf("Введите имя файла: ");
    if (scanf("%255s", filename) != 1) {
        perror("Ошибка ввода имени файла");
        return 1;
    }

    // создание отображаемого файла (shared memory object)
    int shm_fd = shm_open(SHM_NAME, O_CREAT | O_RDWR, 0666);
    if (shm_fd == -1) {
        perror("Ошибка при создании shared memory");
        return 1;
    }

    if (ftruncate(shm_fd, SHM_SIZE) == -1) {
        perror("Ошибка при установке размера shared memory");
        return 1;
```

```
    }

    // отображение в память
    char* shm_ptr = mmap(0, SHM_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, shm_fd, 0);
    if (shm_ptr == MAP_FAILED) {
        perror("Ошибка при mmap");
        return 1;
    }

    pid_t pid = fork();
    if (pid < 0) {
        perror("Ошибка при fork");
        return 1;
    }

    if (pid == 0) {
        // дочерний процесс
        int input_fd = open(filename, O_RDONLY);
        if (input_fd == -1) {
            perror("Ошибка при открытии файла в дочернем процессе");
            exit(1);
        }

        dup2(input_fd, STDIN_FILENO); // stdin ← файл
        close(input_fd);

        execl("./child", "./child", SHM_NAME, NULL);
        perror("Ошибка при запуске дочернего процесса");
```

```
        exit(1);
    } else {
        // родительский процесс
        wait(NULL); // ждём завершения дочернего

        printf("Результаты из отображаемого файла:\n%s", shm_ptr);

        // очистка
        munmap(shm_ptr, SHM_SIZE);
        shm_unlink(SHM_NAME);
    }

    return 0;
}
```

Вывод:

В процессе выполнения лабораторной работы я научился работать с несколькими процессами, а также взаимодействовать между ними с помощью отображаемых файлов. Освоил основные системные вызовы, такие как fork, exec, shm_open, mmap, а также научился перенаправлять ввод и вывод между процессами. Это помогло лучше понять работу операционной системы на уровне процессов и памяти.