Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №1 по курсу**

**«Операционные системы»**

# СОЗДАНИЕ ДОЧЕРНИХ ПРОЦЕССОВ И ВЗАИМОДЕЙСТВИЕ МЕЖДУ НИМИ ПРИ ПОМОЩИ PIPE

Студент: Забелкин Андрей Алексеевич
Группа: М8О–210Б–22
Вариант: 6
Преподаватель: Соколов Андрей Алексеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2023.

## Постановка задачи

**Цель работы**

Целью является приобретение практических навыков в:

- Обеспечении межпроцессорного взаимодействия посредствам технологии pipe
- Освоение принципов работы с файловыми системами

**Задание**

Составить программу на языке Си, осуществляющую работу с процессами и их взаимодействие в ОС на базе UNIX.

Родительский процесс должен открыть файл из которого дочерний процесс читает все числа типа int и передает родительскому процессу их сумму.

## Общие сведения о программе

Программа компилируется из с помощью Makefile, сгенерированным cmake.

Также используется заголовочные файлы:lab1.c, child_process.c. В лабораторной работе используются:

1. **pthread_create()** - эта функция запускает новый поток.
2. **pthread_join()** - эта функция ожидает завершения процесса, указанного в аргументах.

## Общий метод и алгоритм решения.

Для реализации поставленной задачи необходимо:

1. Изучить принципы работы многопоточных программ.
2. Продумать реализацию функций, по возможности, без блокировок и простаивании процессора.
3. Написать генератор матриц.
4. Написать и отладить работу основной функции по созданию процессов и их работе.
5. Придумать тесты и ответы к этим тестам.
6. Написать bash-скрипт, который запускает и проверяет программу на тестах.

## Основные файлы программы

## lab1.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <fcntl.h>

//пользоваться strace

int main(int argc, char *argv[]) {
    int pipefd[2];
    int sum;
    // Создаем канал для обмена данными между процессами
    if (pipe(pipefd) == -1) {
        perror("pipe");
        exit(EXIT_FAILURE);
    }
    pid_t pid;
    pid = fork();
    if (pid == -1) {
        perror("fork");
        exit(EXIT_FAILURE);
    }
    if (pid == 0) {
        // Дочерний процесс
        pid = getpid();
        // Закрываем конец канала, который не используется дочерним процессом
        close(pipefd[0]);
        // Открываем файл для чтения
        int file_fd = open(argv[1], O_RDONLY);
        //где то после этого портятся данные
        if (file_fd == -1) {
            perror("open");
            close(pipefd[1]); // Закрываем запись в канал
            exit(EXIT_FAILURE);
        }
        // Перенаправляем стандартный вывод в канал
        dup2(file_fd, STDIN_FILENO);
        dup2(pipefd[1], STDOUT_FILENO);
        close(pipefd[1]);
        execl("../src/build/child", "child", NULL);
        // Этот код выполняется только в случае неудачи запуска execl
        perror("execl");
        exit(EXIT_FAILURE);
    } else {
        // Родительский процесс
        pid = getpid();
        close(pipefd[1]);
        // Ждем завершения дочернего процесса
        wait(NULL);
        // Читаем число из канала
        char buffer; // Буфер для чтения данных
        ssize_t bytes_read;
        while ((bytes_read = read(pipefd[0], &buffer, sizeof(buffer))) > 0) {
```

```
            write(STDOUT_FILENO, &buffer, bytes_read);
        }
        char endline = '\n';
        write(STDOUT_FILENO, &endline, 1);
        close(pipefd[0]);
        exit(EXIT_SUCCESS);
    }

    return 0;
}
```

## Пример работы

**./run_test.sh**

**Запуск программы для файла test1.txt:**

**-4**

**Пройдено**

**----------**

**Запуск программы для файла test2.txt:**

**30**

**Пройдено**

**----------**

**Запуск программы для файла test3.txt:**

**297**

**Пройдено**

**----------**

**Запуск программы для файла test4.txt:**

**0**

**Пройдено**

**----------**

## Вывод

Во время выполнения это лабораторной работы я узнал, что привычный мне ввод через scanf на самом деле очень удобен и многофункционален, является оберткой над read.

Соответственно в какой то степени мне пришлось реализовать конкретную обертку над чтением байтов из pipe для выполнения лабораторной работы. В остальном же особой сложности с использованием pipe не обнаружил.