Московский Авиационный Институт
(Национальный Исследовательский Университет)
Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

**Лабораторная работа №2 по курсу**
**«Операционные системы»**

Студент:Люгге Т.В,
Группа: М8О-201Б-21
Вариант: 2
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2022

**Содержание**

## Репозиторий

https://github.com/MonkeDLyugge/LabOS

## Постановка задачи

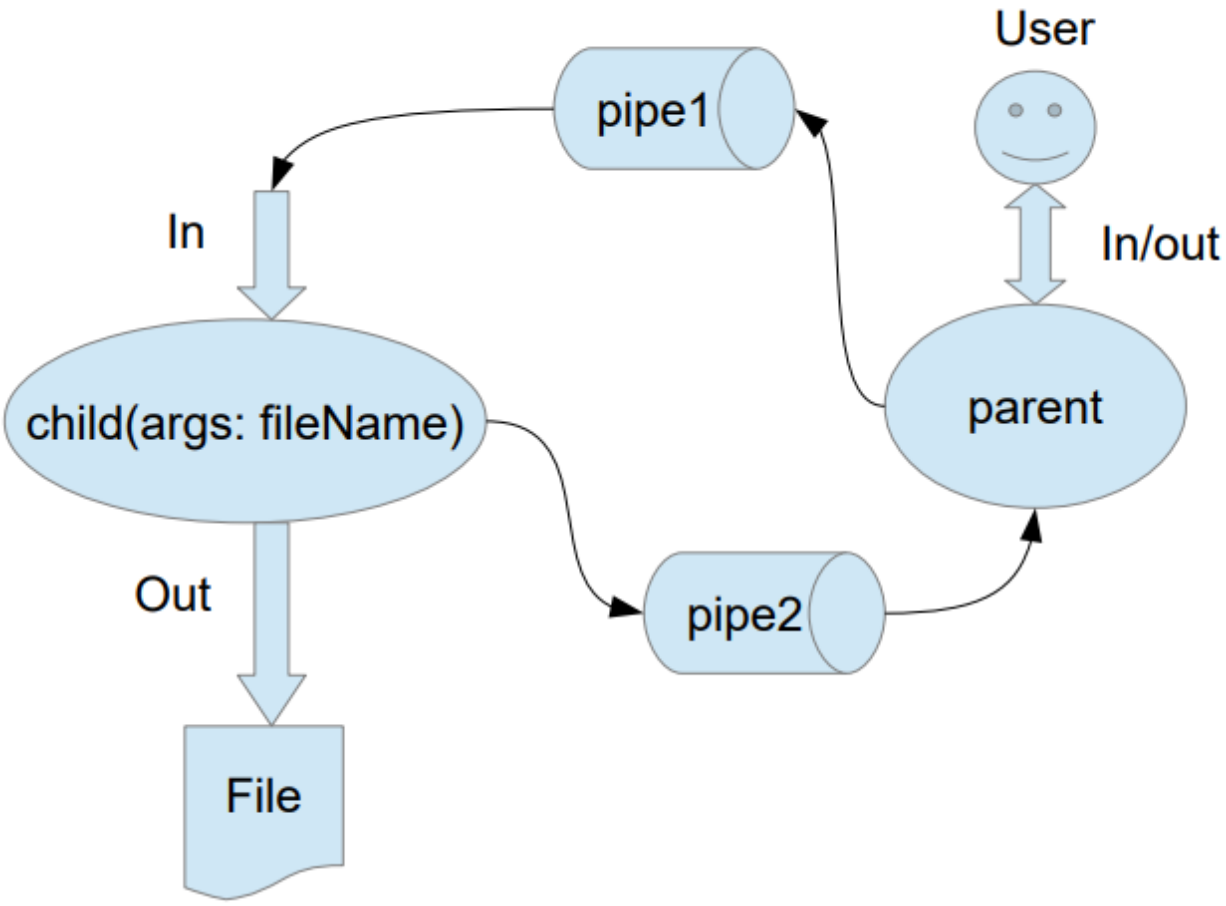
### Цель работы

Приобретение практических навыков в:

- Управление процессами в ОС
- Обеспечение обмена данных между процессами посредством каналов

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




Родительский процесс создает дочерний процесс. Первой строчкой пользователь в консоль родительского процесса пишет имя файла, которое будет передано при создании дочернего процесса. Родительский и дочерний процесс должны быть представлены разными программами. Родительский процесс передает команды пользователя через pipe1, который связан с стандартным

входным потоком дочернего процесса. Дочерний процесс принеобходимости передает данные в родительский процесс через pipe2. Результаты своей работы дочерний процесс пишет в созданный им файл. Допускается просто открыть файл и писать туда, не перенаправляя стандартный поток вывода

## Общие сведения о программе

Программа родительского процесса компилируется из parent.cpp. Программа дочернего процесса компилируется из child.cpp. В программах используются следующие системные вызовы:

1. mkfifo() – создание именованного канала
2. unlink() – удаление имени из файловой системы
3. fork() – создание дочернего процесса
4. open() – открытие файла
5. close() – закрытие файла
6. write() – запись последовательности байт
7. read() – чтение последовательности байт
8. execl() – замена образа памяти процесса
9. dup2() – переназначение файлового дескриптора

## Общий метод и алгоритм решения

Родительский процесс получает имя файла, после чего создаётся дочерний процесс, при вызове execl() полученное имя файла передаётся в дочерний процесс в качестве аргументов командной строки. После того как оба процесса открыли каналы, они входят в циклы, условие выхода из которых – конец ввода. Родительский процесс передаёт введённое число в дочерний, после чего дочерний процесс обновляет сумму введенных чисел.

## Исходный код

**parent.cpp**

```
// Parent process would provide data collecting and transmission to the child process
#include <cstdlib>
#include <iostream>

#include "unistd.h"
#include "stdio.h"
#include "sys/stat.h"
#include "sys/wait.h"
#include "fcntl.h"

#include "parent.hpp"

int ParentProcess(FILE* standartInput, const std::string& path) {
    // The entry point to the parent process

    // If last execution complited with no closing/deleting pipe
    unlink("pipe");

    // Creating new pipes
    if (mkfifo("pipe", S_IREAD | S_IWRITE) == -1) {
        // Creating pipe error
        perror("file didnt create");
        return -1;
    }
```

```
    size_t n = 0;
    char* fileName;

    // Reading file name
    int charactersCount = getline(&fileName, &n, standartInput);

    if (charactersCount <= 0) {
        // Reading file name error
        return -1;
    }

    int pid = fork();
    if (pid == -1) {
        // Fork error
        return -1;
    }
    if (pid != 0) {
        // Opening pipe
        int pipe = open("pipe", O_WRONLY);

        if (pipe == -1) {
            // Opening pipe error
            return -1;
        }

        char* str = nullptr;
        size_t k = 0;

        charactersCount = getline(&str, &k, standartInput);
        while (charactersCount > 0) {
            if (write(pipe, str, charactersCount) == -1) {
                // Sending data from parent process error
                return -1;
            }
            free(str);
            str = nullptr;
            charactersCount = getline(&str, &k, standartInput);
        }

        // Closing/deleting pipe
        close(pipe);
        wait(NULL);
    } else {
        // Deleting \n from file name
        fileName[charactersCount - 1] = '\0';

        int pipe = open("pipe", O_RDONLY);

        dup2(pipe, 0);

        char* argv[3];
        sprintf(argv[0], "%s", "child.cpp");
        argv[1] = fileName;
        argv[2] = NULL;

        return execv(path.c_str(), argv);
    }
    return 0;
}
```

**child.cpp**

```
#include "unistd.h"
#include "stdio.h"
#include "stdlib.h"
#include "sys/stat.h"
#include "fcntl.h"

#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char* argv[]) {
    //int pipe = open("pipe", O_RDONLY);
    if (argc < 2) {
        // Missing some arguments
        return -1;
    }
    /*(if (pipe == -1) {
        // Opening pipe error
        return -1;
    }*/
    unlink(argv[1]);
    int fd = open(argv[1], O_CREAT | O_WRONLY, S_IREAD | S_IWRITE);
    if (fd == -1) {
        // Opening file error
        return -1;
    }

    if (dup2(fd, 1) == -1) {
        // Dup2 error
        return -1;
    }

    float ans = 0;

    float num;

    while (std::cin >> num) {
        ans += num;
    }

    printf("%.3f", ans);

    // close(pipe);
    close(fd);
    return 0;
}
```

## Демонстрация работы программы

microhacker@microhacker-HLYL-WXX9:~/Desktop/LabOS$ ./parent.out

file.txt

1 2 3

0.5 0.7

0.2

microhacker@microhacker-HLYL-WXX9:~/Desktop/LabOS$ cat file.txt

7.400

## Выводы

Составлена и отлажена программа на языке Си, осуществляющая работу с процессами. Тем самым, приобретены навыки в управлении процессами в ОС и обеспечении обмена данных между процессами посредством каналов.