

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ  
ФЕДЕРАЦИИ

Московский Авиационный Институт  
(Национальный Исследовательский Университет)

Институт №8 "Компьютерные науки и прикладная математика"  
Кафедра 806 "Вычислительная математика и программирование"

Лабораторная работа №1  
По курсу «Операционные системы»

Студент: Снетков Н.С.  
Группа: М8О-203Б-23  
Вариант: 15  
Преподаватель: Миронов Е. С.

Дата: \_\_\_\_\_

Оценка: \_\_\_\_\_

Подпись: \_\_\_\_\_

Москва, 2025

## **Содержание**

1. Репозиторий
2. Постановка задачи
3. Общие сведения о программе
4. Общий метод и алгоритм решения
5. Исходный код
6. Сборка программы
7. Демонстрация работы программы
8. Выводы

## Репозиторий

[https://github.com/Sapfir7/labs\\_os\\_snet/tree/main/os\\_labs/lab1](https://github.com/Sapfir7/labs_os_snet/tree/main/os_labs/lab1)

## Постановка задачи

### Цель работы

Приобретение практических навыков в:

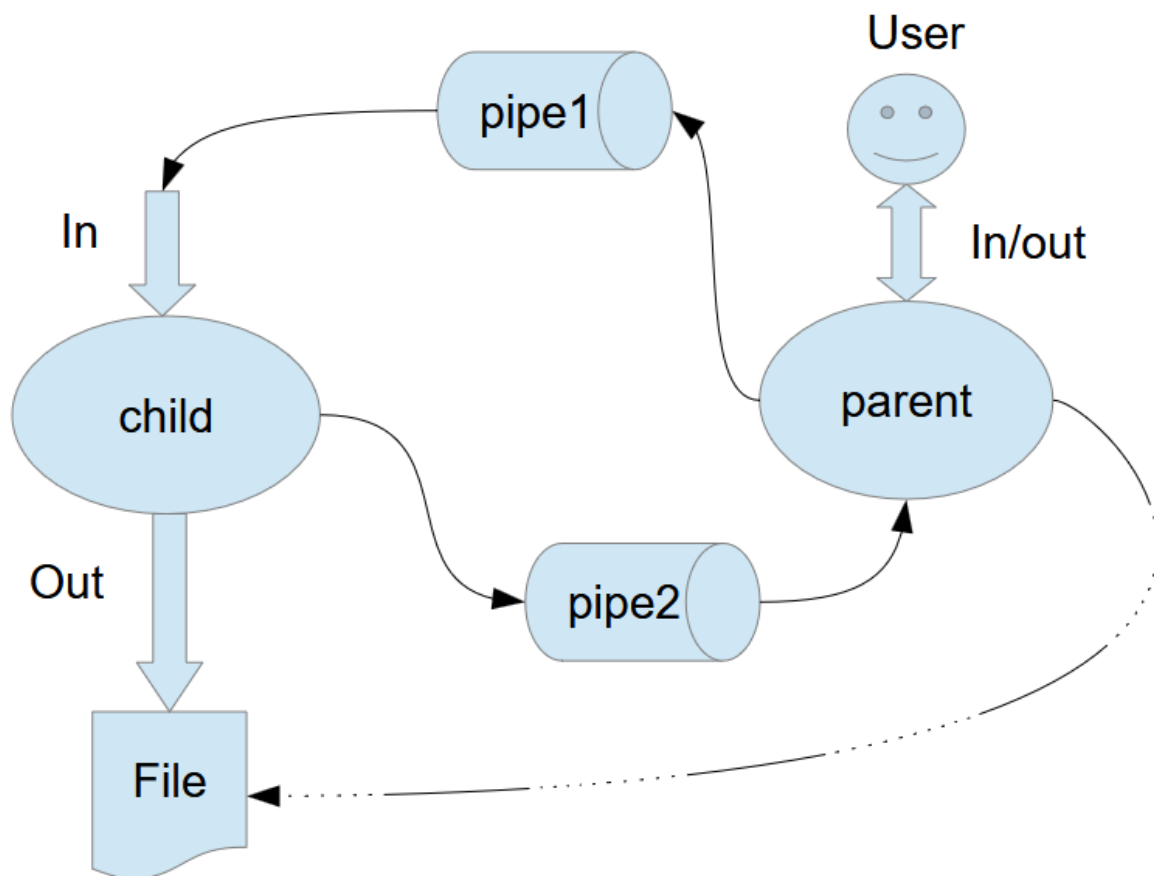
Управление процессами в ОС

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.

Группа вариантов 4



Родительский процесс создает дочерний процесс. Первой строкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия File с таким именем на запись. Перенаправление стандартных потоков ввода-вывода показано на картинке выше. Родительский и дочерний процесс должны быть представлены разными программами. Родительский процесс принимает от пользователя строки произвольной длины и пересылает их в pipe1. Процесс child проверяет строки на валидность правилу. Если строка соответствует правилу, то она выводится в стандартный поток вывода дочернего процесса, иначе в pipe2 выводится информация об ошибке. Родительский процесс полученные от child ошибки выводит в стандартный поток вывода.

Вариант 15) Правило проверки: строка должна начинаться с заглавной буквы

### **Общие сведения о программе**

Программа компилируется из файла main.cpp. В программе используются следующие системные вызовы:

1. pipe() - существует для передачи информации между различными процессами.
2. fork() - создает новый процесс.
3. execl() - передает процесс на исполнение другой программе.
4. read() - читает данные из файла.
5. write() - записывает данные в файл.
6. close() - закрывает файл.

### **Общий метод и алгоритм решения**

1. Родительский процесс (parent.cpp)

Родительский процесс выполняет следующие шаги:

Создание каналов (pipes):

Создаются два канала: pipe1 и pipe2. pipe1 будет использоваться для передачи данных от родительского процесса к дочернему, а pipe2 — для передачи данных от дочернего процесса к родительскому.

Создание дочернего процесса:

Родительский процесс создает дочерний процесс с помощью системного вызова fork(). В дочернем процессе выполняется программа child, которая обрабатывает данные, переданные через канал.

Передача данных в дочерний процесс:

Родительский процесс запрашивает у пользователя имя файла, в который будут записываться результаты, и строки текста. Каждая строка передается в дочерний процесс через канал `pipe1`.

Получение данных от дочернего процесса:

Родительский процесс читает данные из канала `pipe2`, который используется для получения результатов обработки от дочернего процесса. Если дочерний процесс возвращает ошибку (строка начинается с "Error:"), она выводится на экран. В противном случае результат записывается в файл.

Завершение работы:

После завершения ввода данных (пользователь вводит "exit"), родительский процесс закрывает каналы и ожидает завершения дочернего процесса с помощью `wait()`.

## 2. Дочерний процесс (`child.cpp`)

Дочерний процесс выполняет следующие шаги:

Получение дескрипторов каналов:

Дочерний процесс получает дескрипторы каналов `pipe1` и `pipe2` через аргументы командной строки. `pipe1` используется для чтения данных от родительского процесса, а `pipe2` — для отправки результатов обратно.

Чтение данных из канала:

Дочерний процесс читает данные из канала `pipe1`. Каждая строка проверяется на то, начинается ли она с заглавной буквы.

Проверка строки:

Если строка начинается с заглавной буквы, она отправляется обратно в родительский процесс через канал `pipe2`.

Если строка не начинается с заглавной буквы, дочерний процесс отправляет сообщение об ошибке в родительский процесс.

Завершение работы:

Дочерний процесс продолжает чтение и обработку данных до тех пор, пока родительский процесс не закроет канал. После этого дочерний процесс закрывает свои каналы и завершает работу.

## 3. Взаимодействие между процессами

Родительский процесс передает строки текста в дочерний процесс через канал `pipe1`.

Дочерний процесс проверяет каждую строку и отправляет результат проверки (либо саму строку, либо сообщение об ошибке) обратно в родительский процесс через канал `pipe2`.

Родительский процесс записывает результаты в файл или выводит сообщения об ошибке на экран.

## Исходный код

### **child.h:**

```
#ifndef CHILD_H
#define CHILD_H

#include <string>
#include <unistd.h>
#include <iostream>
#include <cctype>

#define BUFFER_SIZE 1024

void child_process(int pipe1[2], int pipe2[2]);

bool is_valid_string(const std::string& str);

#endif
```

### **parent.h:**

```
#ifndef PARENT_H
#define PARENT_H

void ParentMain();

#endif
```

### **child.cpp:**

```
#include <iostream>
#include <unistd.h>
#include <cstring>
#include <cctype>
```

```

int main(int argc, char *argv[]) {
    if (argc != 3) {
        std::cerr << "Usage: child <readPipe> <writePipe>\n";
        return 1;
    }

    int readPipe = std::stoi(argv[1]);
    int writePipe = std::stoi(argv[2]);

    char buffer[256];
    ssize_t bytesRead;

    while ((bytesRead = read(readPipe, buffer, sizeof(buffer) - 1)) > 0) {
        buffer[bytesRead] = '\0';

        if (isupper(buffer[0])) {
            write(writePipe, buffer, bytesRead);
        } else {
            const char *errorMsg = "Error: Line must start with an uppercase letter.\n";
            write(writePipe, errorMsg, strlen(errorMsg));
        }
    }

    close(readPipe);
    close(writePipe);
    return 0;
}

```

```
}
```

### **parent.c:**

```
#include <iostream>
```

```
#include <unistd.h>
```

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

```
#include <sys/wait.h>
```

```
#include <string>
```

```
#include <cstring>
```

```
#include <fstream>
```

```
void ParentMain() {
```

```
    int pipe1[2], pipe2[2];
```

```
    if (pipe(pipe1) == -1 || pipe(pipe2) == -1) {
```

```
        perror("pipe");
```

```
        return;
```

```
    }
```

```
    pid_t pid = fork();
```

```
    if (pid == -1) {
```

```
        perror("fork");
```

```
        return;
```

```
    }
```

```
    if (pid == 0) {
```

```
        close(pipe1[1]);
```

```
        close(pipe2[0]);
```



```

    execl("/home/denis/Рабочий стол/os/OS-labs-template/build/lab2/child", "child",
std::to_string(pipe1[0]).c_str(), std::to_string(pipe2[1]).c_str(), nullptr);

    perror("Execl failed");

    exit(1);
} else {

    close(pipe1[0]);

    close(pipe2[1]);

    std::string filename;

    std::cout << "Enter filename: ";

    std::getline(std::cin, filename);

    std::ofstream outfile(filename);

    if (!outfile) {

        std::cerr << "Cannot open file for writing.\n";

        return;

    }

    std::string line;

    std::cout << "Enter text lines (type 'exit' to stop):" << std::endl;

    while (std::getline(std::cin, line)) {

        if (line == "exit") break;

        write(pipe1[1], line.c_str(), line.size() + 1);

        char buffer[256];

        ssize_t bytesRead = read(pipe2[0], buffer, sizeof(buffer) - 1);

        if (bytesRead > 0) {

            buffer[bytesRead] = '\0';

            if (strncmp(buffer, "Error:", 6) == 0) {

```

```

        std::cout << "Child error: " << buffer << std::endl;
    } else {
        outfile << buffer;
    }
}

}

}

close(pipe1[1]);
close(pipe2[0]);
wait(nullptr);
}
}

```

### main.c:

```
#include "include/parent.h"
```

```

int main() {
    ParentMain();
    return 0;
}

```

## Демонстрация работы программы

```

29 | }
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
denis@denis-System-Product-Name:~/Рабочий стол/os/05-labs-template/build/lab2$ ./ProcessCommunication
Enter filename: adadad
Enter text lines (type 'exit' to stop):
Adadadada
dadada
dadada
Child error: Error: Line must start with an uppercase letter.

Adadadada
Adadadada
KFKFKFKadada
Kadadada
kdkadkdkadkDDDDADAD
Child error: Error: Line must start with an uppercase letter.

SSSADADADA
dadadadaddada
Child error: Error: Line must start with an uppercase letter.

```

## Выводы

В ходе выполнения лабораторной работы я познакомился с механизмом взаимодействия процессов через каналы (**pipe**) в UNIX-системах. Я изучил использование системных вызовов **fork()**, **dup2()**, **execl()** и **wait()**. Новым для меня стало понимание перенаправления стандартных потоков ввода/вывода и обработки данных в дочерних процессах.