

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: Павлов Иван Дмитриевич  
Группа: М8О-207Б-21  
Вариант: 11  
Преподаватель: Миронов Евгений Сергеевич  
Оценка: \_\_\_\_\_  
Дата: \_\_\_\_\_  
Подпись: \_\_\_\_\_

Москва, 2022

## **Содержание**

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

## Репозиторий

[https://github.com/Pavloffff/MAI\\_OS](https://github.com/Pavloffff/MAI_OS)

## 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.

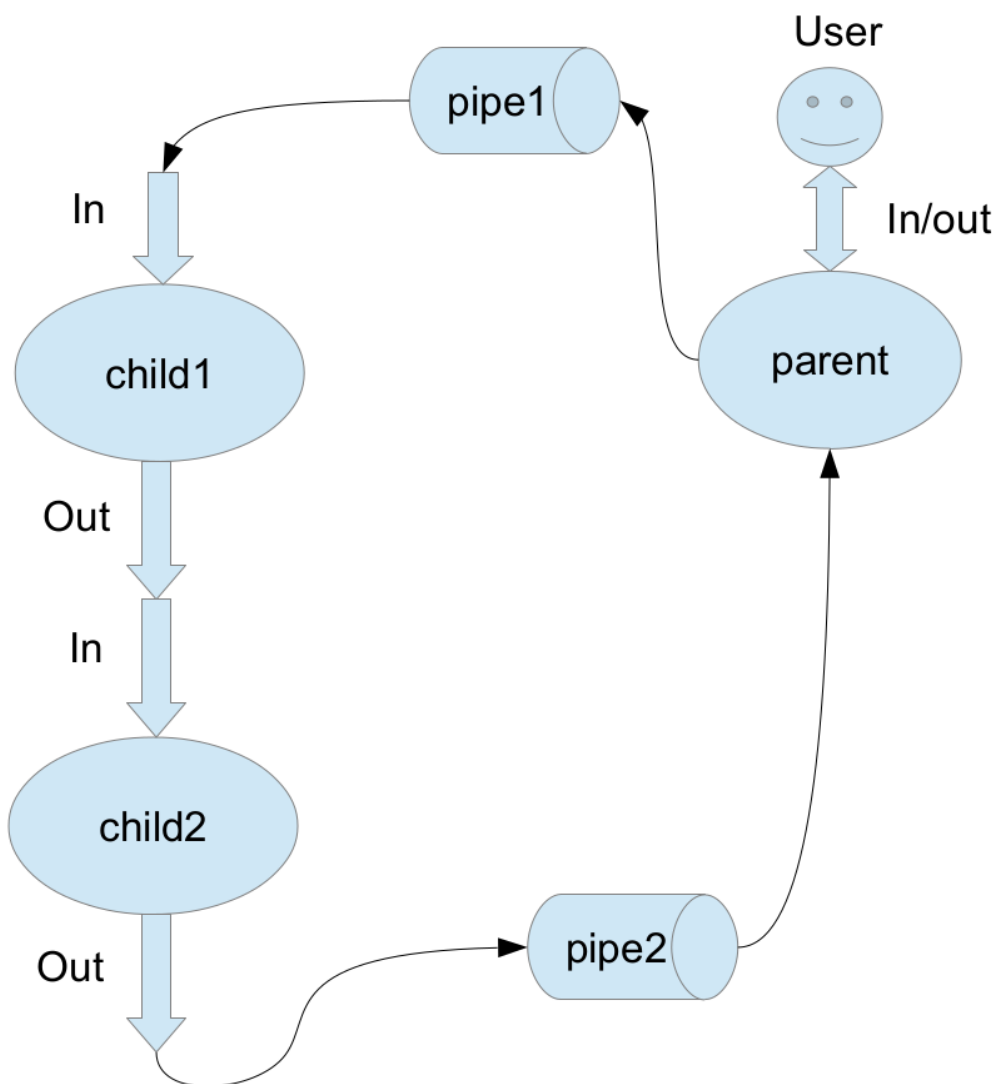
Группа вариантов 3

Родительский процесс создает два дочерних процесса. Перенаправление стандартных потоков ввода-вывода показано на картинке выше. Child1 и Child2 можно «соединить» между собой дополнительным каналом. Родительский и дочерний процесс должны быть представлены разными программами.

Родительский процесс принимает от пользователя строки произвольной длины и пересылает их в pipe1. Процесс child1 и child2 производят работу над строками. Child2 пересылает результат своей работы родительскому процессу. Родительский процесс полученный результат выводит в стандартный поток вывода.

Вариант 11

Child1 переводит строки в верхний регистр. Child2 превращает все пробельные символы в символ «\_».



### Общие сведения о программе

Программа компилируется из файла main.cpp. Также используется заголовочные файлы: unistd.h, iostream, string, algorithm. В программе используются следующие системные вызовы:

1. pipe() - существует для передачи информации между различными процессами.
2. fork() - создает новый процесс.
3. execpl() - 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

ЛР выполнена на ОС UNIX, на языке C++.

В условии допущено соединение двух дочерних процессов с помощью дополнительного канала, поэтому создам 3 pipe; дескрипторы будут лежать в двумерном массиве fd[3][2]. Далее создаем первый дочерний процесс с помощью fork(). Проверяем id, которое вернул fork(): если id != 0, то это родительский процесс.

Согласно условию, родительский процесс считывает строки вместе с пробелами, до "\n"; поэтому использую while (getline(std::cin, str)) (до этого я объявил строку из std::string). Далее необходимо передать эти строки первому дочернему процессу (child1), используя write(). Так как write() - сишная функция, приведем строку к массиву char\* методом std::string::c\_str. Кроме того передаем размер строки, увеличенный на 1 (из-за "\0").

Далее, переходим к child1 (там где id == 0). По условию, родительский и дочерний процессы должны быть представлены разными программами. Поэтому использую execvp, и передаю туда в качестве параметров название исполняемого файла дочернего процесса, и все fd, приведенные к строке в формате Си.

Код дочерних процессов представлен в файле child.cpp. Из argv[] получим все fd. Далее с помощью fork() создадим child2, и если fork вернул id != 0, то приступаю к выполнению задачи child1.

Для того, чтобы считать строку для обработки, сперва с помощью read() считываем размер строки, затем создаем буфер дополнительной памяти этого размера, записываем туда строку и присваиваем строке из C++ этот буфер, после чего чистим память.

По условию child1 преобразует строки в верхний регистр, сделаем это с помощью std::transform(), итераторов и лямбда-выражений. Передадим через другой pipe это процессу 2.

Аналогичные действия совершает и процесс 2, который по завершении итерации передает через 3-й pipe обработанные строки родительскому процессу, который выводит их на экран.

При этом не забываем обрабатывать ошибки, связанные с системными вызовами.

## Исходный код

main.cpp

```
#include <iostream>
#include <string>
#include <algorithm>
#include <unistd.h>
```

```
int main(int argc, char const *argv[])
```

5

```

{
    int fd[3][2];
    for (int i = 0; i < 3; i++) {
        if (pipe(fd[i]) == -1) {
            std::cerr << "pipe error\n";
            return -1;
        }
    }
    pid_t Child1Id = fork();
    if (Child1Id == -1) {
        std::cerr << "fork error\n";
        return -1;
    } else if (Child1Id == 0) {
        execlp("./child",
            std::to_string(fd[0][0]).c_str(),
            std::to_string(fd[0][1]).c_str(),
            std::to_string(fd[1][0]).c_str(),
            std::to_string(fd[1][1]).c_str(),
            std::to_string(fd[2][0]).c_str(),
            std::to_string(fd[2][1]).c_str(),
            NULL);
    } else {
        std::string str;
        int sz;
        while (getline(std::cin, str)) {
            fflush(stdout);
            sz = str.size() + 1;
            write(fd[0][1], &sz, sizeof(sz));
            write(fd[0][1], str.c_str(), sz);
            read(fd[2][0], &sz, sizeof(sz));
            char *buf = new char[sz];
            read(fd[2][0], buf, sz);
            str = buf;
            delete[] buf;
            std::cout << str << "\n";
        }
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 2; j++) {

```

```

        close(fd[i][j]);
    }
}
}
return 0;
}

```

child.cpp

```
#include <iostream>
```

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

```
#include <string>
```

```
#include <algorithm>
```

```
int main(int argc, char const *argv[])
```

```

{
    int fd[3][2];
    fd[0][0] = std::stoi(argv[0]);
    fd[0][1] = std::stoi(argv[1]);
    fd[1][0] = std::stoi(argv[2]);
    fd[1][1] = std::stoi(argv[3]);
    fd[2][0] = std::stoi(argv[4]);
    fd[2][1] = std::stoi(argv[5]);
    pid_t Child2Id = fork();
    if (Child2Id == -1) {
        std::cerr << "fork error\n";
        return -1;
    } else if (Child2Id == 0) {
        while (1) {
            fflush(stdout);
            std::string str;
            int sz;
            read(fd[1][0], &sz, sizeof(sz));
            char *buf = new char[sz];
            read(fd[1][0], buf, sz);
            str = buf;
            delete[] buf;
            std::transform(str.begin(),
                           str.end(),
                           str.begin(),

```

```

        [](unsigned char c) { return (c == ' ') ? '_' : c; });
    write(fd[2][1], &sz, sizeof(sz));
    write(fd[2][1], str.c_str(), sz);
}
} else {
    while (1) {
        fflush(stdout);
        std::string str;
        int sz;
        read(fd[0][0], &sz, sizeof(sz));
        char *buf = new char[sz];
        read(fd[0][0], buf, sz);
        str = buf;
        delete[] buf;
        std::transform(str.begin(),
            str.end(),
            str.begin(),
            [](unsigned char c) { return std::toupper(c); });
        write(fd[1][1], &sz, sizeof(sz));
        write(fd[1][1], str.c_str(), sz);
    }
}
for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 2; j++) {
        close(fd[i][j]);
    }
}
return 0;
}

```



```
ggame@ggame:~/OS/ready/lab2$ g++ -o child child.cpp
ggame@ggame:~/OS/ready/lab2$ g++ -o main main.cpp
ggame@ggame:~/OS/ready/lab2$ ./main
aaaaaaa bbbbbb xxxxxx qwertyy
AAAAAAA_BBBBBB_XXXXX_QWERTTY
3523hj 3523   jihhi 999
3523HJ_3523____JIHHI_999
      a
_____A
ggame@ggame:~/OS/ready/lab2$
```

## **Выводы**

В ходе выполнения ЛР №2 я познакомился с системными вызовами, процессами.