

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: Друхольский Александр
Группа: М8О-207Б-21
Вариант: 22
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/ssForz/OS-labs>

Постановка задачи

Цель работы

Приобретение практических навыков в:

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

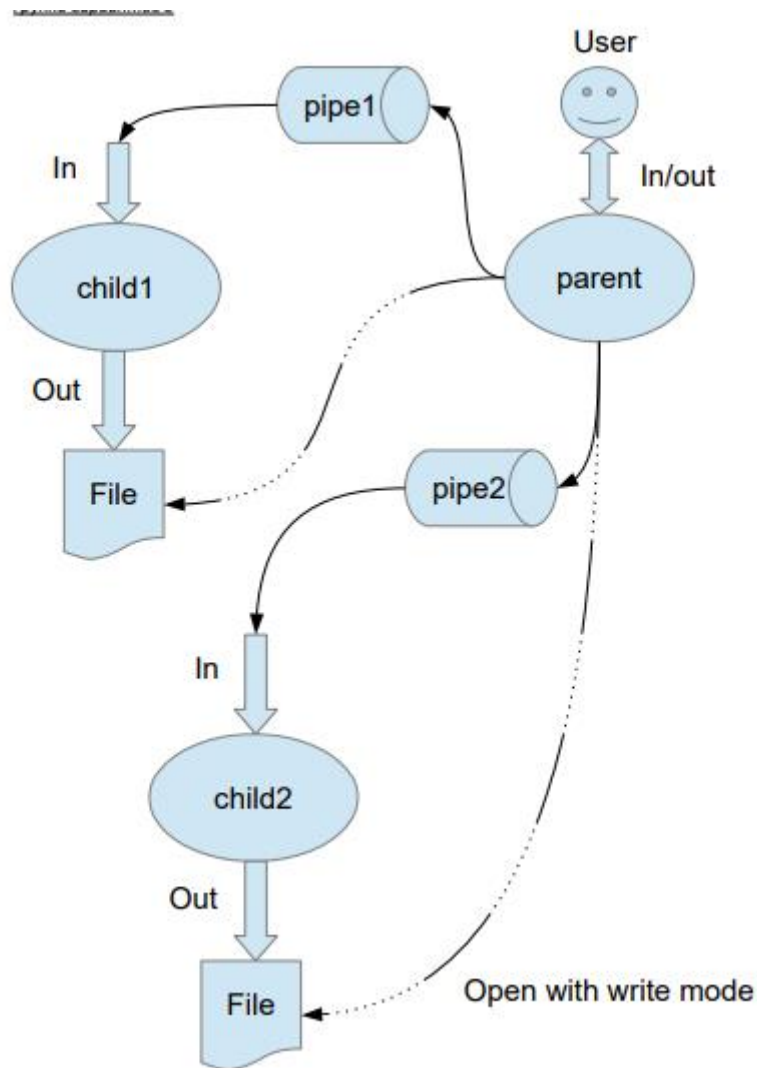
Задание

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 5:

Родительский процесс создает два дочерних процесса. Первой строкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия File с таким именем на запись для child1. Аналогично для второй строки и процесса child2. Родительский и дочерний процесс должны быть представлены разными программами. parent child1 pipe1 In/out User In Out child2 In Out File Open with write mode pipe2 File Родительский процесс принимает от пользователя строки произвольной длины и пересылает их в pipe1 или в pipe2 в зависимости от правила фильтрации. Процесс child1 и child2 производят работу над строками. Процессы пишут результаты своей работы в стандартный вывод

22) Правило фильтрации: с вероятностью 80% строки отправляются в pipe1, иначе в pipe2. Дочерние процессы инвертируют строки.



Общие сведения о программе

Программа компилируется из файла main.cpp с помощью stake. Дочерний процесс представлен в exes_child.cpp

Системные вызовы:

- `pid_t fork()` - создание дочернего процесса
- `int execl(const char *filename, char *const argv[], char *const)` - замена образа памяти процесса
- `int pipe(int pipefd[2])` - создание неименованного канала для передачи данных между процессами
- `int open(const char *pathname, int flags, mode_t mode)` - открытие\создание файла
- `int close(int fd)` - закрыть файл

Общий метод и алгоритм решения

Оба процесса выполняют одну и ту же функцию — инвертирование строк. Напишем одну программу для дочернего процесса (обычный алгоритм инвертирования строк). В главном процессе создадим два канала pipe, свяжем его с файловыми дескрипторами, которые будем передавать в функцию `execpl`. Оба дочерних процесса будут вызывать `exec_child`. Соответственно с вариантом реализуем рандом с шансом 80% на попадание строк в первый дочерний процесс. Для каждого процесса реализуем свои выходные файлы.

Исходный код

main.cpp

```
#include <iostream>
#include <string>
#include <cstdlib>
#include <sys/types.h>
#include <unistd.h>
#include <fstream>
#include <errno.h>
#include <signal.h>
#include <sys/wait.h>
```

```
using namespace std;
```

```
int main()
{
    string current_str;
    int child_tag;
    int fd[2];
    char *const child_args[] = { "./exec_child", NULL };
    fstream res_file;
```

```

string child1, child2;
cout<<"Enter the name for first child file: ";
cin>>child1;
cout<<endl;
cout<<"Enter the name for second child file: ";
cin>>child2;
cout<<endl;
int fd1[2];
int fd2[2];

if (pipe(fd1) == -1) {
    cout<<"Pipe error occurred"<<endl;
    exit(EXIT_FAILURE);
}
if (pipe(fd2) == -1) {
    cout<<"Pipe error occurred"<<endl;
    exit(EXIT_FAILURE);
}

pid_t f_id1 = fork();
if (f_id1 == -1) {
    cout<<"Fork error with code -1 returned in the parent, no child_1 process is
created"<<endl;
    exit(EXIT_FAILURE);
} else if (f_id1 == 0) { //child1
    child_tag = 1;
    fd[1] = fd1[1];
    fd[0] = fd1 [0];
    string child = child1;

```

```

        execlp(child_args[0], to_string(fd[0]).c_str(), to_string(fd[1]).c_str(), "1",
child.c_str(), NULL);

        perror("Execlp error");

    }

    pid_t f_id2 = fork();
    if (f_id2 == -1) {
        cout<<"Fork error with code -1 returned in the parent, no child_2 process is
created"<<endl;

        exit(EXIT_FAILURE);
    } else if (f_id2 == 0) { //child2
        child_tag = 2;
        fd[1] = fd2[1];
        fd[0] = fd2[0];
        string child = child2;

        execlp(child_args[0], to_string(fd[0]).c_str(), to_string(fd[1]).c_str(), "2",
child.c_str(), NULL);

        perror("Execlp error");

    }

    else { //parent
        cout<<"Enter amount of strings: ";

        int cnt;

        cin>>cnt;

        cout<<endl;

        for (int i = 0; i < cnt; ++i) {
            cin>>current_str;

```

```

    int s_size = current_str.size();
    char str_array[s_size];
    for (int k = 0; k < s_size; ++k) {
        str_array[k] = current_str[k];
    }
    if (rand() % 6 < 5) {
        write(fd1[1], &s_size, sizeof(int));
        write(fd1[1], str_array, sizeof(char)*s_size);

    } else {
        write(fd2[1], &s_size, sizeof(int));
        write(fd2[1], str_array, sizeof(char)*s_size);

    }
}
}
close(fd2[1]);
close(fd1[1]);
close(fd2[0]);
close(fd1[0]);
/*kill(f_id1, SIGTERM);
kill(f_id2, SIGTERM);*/
return 0;
}

```

exec_child.cpp

```

#include <iostream>
#include <string>
#include <cstdlib>
#include <sys/types.h>
#include <unistd.h>

```



```

#include <fstream>

#include <errno.h>

#include <string.h>


using namespace std;

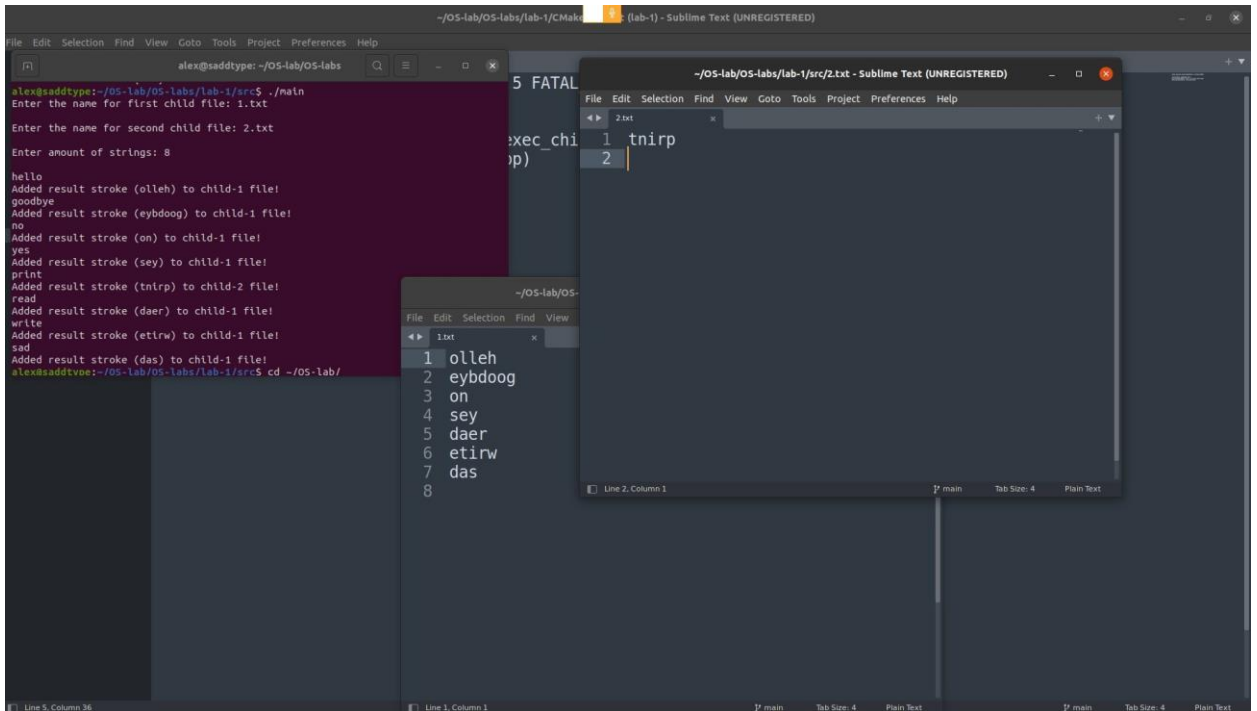

int main(int argc, char const *argv[])
{
    string filename = argv[3];
    int fd[2];
    fd[0] = stoi(argv[0]);
    fd[1] = stoi(argv[1]);
    fstream cur_file;
    cur_file.open(filename, fstream::in | fstream::out | fstream::app);
    while (true) {
        int size_of_str;
        read(fd[0], &size_of_str, sizeof(int));
        char str_array[size_of_str];
        read(fd[0], &str_array, sizeof(char)*size_of_str);
        string result_str;
        for (int i = size_of_str - 1; i >= 0; i--) {
            result_str.push_back(str_array[i]);
        }
        cur_file << result_str << endl;
        cout<<"Added result stroke ("<<result_str<<") to child-"<<argv[2]<<" file!"<<endl;
    }


    return 0;


}

```

Демонстрация работы программы



The screenshot shows a terminal window with the following output:

```
alex@saddtype: ~/OS-lab/OS-labs
alex@saddtype:~/OS-lab/OS-labs/1/src$ ./main
Enter the name for first child file: 1.txt
Enter the name for second child file: 2.txt
Enter amount of strings: 8
hello
Added result stroke (olleh) to child-1 file!
goodbye
Added result stroke (eybdoog) to child-1 file!
no
Added result stroke (on) to child-1 file!
yes
Added result stroke (sey) to child-1 file!
print
Added result stroke (tnirp) to child-2 file!
read
Added result stroke (daer) to child-1 file!
write
Added result stroke (etirw) to child-1 file!
sad
Added result stroke (das) to child-1 file!
alex@saddtype:~/OS-lab/OS-labs/1/src$ cd ~/OS-lab/
```

Two Sublime Text editors are open:

- `1.txt` (located at `~/OS-lab/OS-labs/1/src/1.txt`) contains:

```
1 olleh
2 eybdoog
3 on
4 sey
5 daer
6 etirw
7 das
8
```
- `2.txt` (located at `~/OS-lab/OS-labs/1/src/2.txt`) contains:

```
1 tnirp
2
```

Выводы

В ходе данной лабораторной работы я познакомился с некоторыми системными вызовами в Unix. Я научился создавать `pipe` и работать с файловым дескриптором. Понял как обращаться к дочернему процессу, передавать в него данные. В целом, я разобрался как управлять процессами и информацией, передающейся между ними.