

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

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

**Тема работы
“Изучение взаимодействий между процессами”**

Студент: Маринин Иван Сергеевич
Группа: М8О-208Б-20
Вариант: 22
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2021

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/ISMarinin/OS.git>

Постановка задачи

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.

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.

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

Общие сведения о программе

Реализация программы была бы невозможна без специальной библиотеки `<unistd.h>` для операционной системы Linux, которая позволяет работать с процессами и системными вызовами.

По мере реализации задания используются такие строки(команды), как:

int fd[2] - создание массива из 2 дескрипторов, 0 - чтение (read), 1 - передача (write)

pipe(fd) - конвейер, с помощью которого выход одной команды подается на вход другой (оно же “труба”)

int child_id = fork () - создание дочернего процесса, в переменной id будет лежать “специальный код” процесса (-1 - ошибка fork, 0 - дочерний процесс, >0 - родительский)

read(...) - команда, предназначенная для чтения данных, посланных из другого процесса, принимающая на вход три параметра: элемент массива дескрипторов с индексом 0, значение **получаемого** объекта (переменной, массива и т.д.), размер **получаемого** объекта (например, в случае переменной int - sizeof(int), в случае массива из 10 переменных типа int - sizeof(int) * 10)

write(...) - команда, принимающая на вход три параметра: элемент массива дескрипторов с индексом 1, значение **посылаемого** объекта (переменной, массива и т.д.), размер **посылаемого** объекта (например, в случае переменной int - sizeof(int), в случае массива из 10 переменных типа int - sizeof(int) * 10)

close(...) - команда, используемая, когда нам больше не нужно передавать, либо считывать что-либо из другого процесса.

Общий метод и алгоритм решения

С самого начала программа получает два названия файлов для записи работы дочерних процессов. После этого эти оба файла создаются, и программа запрашивает у пользователя количество строк. Далее выполняется следующий алгоритм: после введения строки в консоль пользователь может увидеть ответ либо от первого дочернего процесса, либо от второго дочернего процесса, так как каждый процесс представляется, прежде чем вывести уже готовую строку пользователю (то есть инвертированную строку). В самой программе инвертирование строки представлено посредством цикла **for** и пробегом по строке (реализация будет представлена ниже в графе “Исходный код”)

По окончании работы программы пользователь имеет выведенные инвертированной строки и в консоли, и в созданном в самом начале файле, как и требовалось в задании.

Лабораторная работа была выполнена в среде **Visual Studio Code**, название файла - **task_22.cpp**.

Собирается программа при помощи команды **g++ task_22.cpp -o a.exe**, запускается при помощи команды **./a.exe**.

Исходный код

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

using namespace std;

int main() {

    string filename1;
    string filename2;

    cout << "Enter enter the file names:" << endl;
    cin >> filename1;
    cin >> filename2;
    fstream fs;

    int fd1[2];
    pipe(fd1);
    int fd2[2];
    pipe (fd2);

    if (pipe(fd1) == -1 || pipe(fd2) == -1) {
        cout << "pipe error!" << endl;
        return 1;
    }

    int child_id = fork();
    if (child_id == -1) {
        cout << "Fork error!" << endl;
        return -1;
    }

    else if (child_id == 0) {
        fs.open(filename1, fstream :: in | fstream :: out |
fstream :: app);
        int n;
```

```

    read (fd1[0], &n, sizeof(int));
    while (n > 0) {
        int size;
        read(fd1[0], &size, sizeof(int));
        char str[size];
        read(fd1[0], str, sizeof(char) * size);
        string s;
        for (int i = 0; i < size; ++i) {
            s.push_back(str[i]);
        }

        int j = (int)s.size() - 1;
        for (int i = 0; i < s.size() / 2; ++i) {
            char tmp = s[i];
            s[i] = s[j];
            s[j] = tmp;
            j--;
        }

        fs << s << endl;
        cout << "[CHILD1]" << s << endl;
        n--;
    }

    close(fd1[0]);
    close(fd1[1]);
}

else {

    int child_id2 = fork();

    if (child_id2 == -1) {
        cout << "Fork error!" << endl;
        return -1;
    }

    else if (child_id2 == 0) {
        fs.open(filename2, fstream :: in | fstream :: out |
fstream :: app);

```

```

        int n;
        read(fd2[0], &n, sizeof(int));
        while (n > 0) {
            int size;
            read(fd2[0], &size, sizeof(int));
            char str[size];
            read(fd2[0], str, sizeof(char) * size);
            string s;

            for (int i = 0; i < size; ++i) {
                s.push_back(str[i]);
            }

            int j = (int)s.size() - 1;

            for (int i = 0; i < s.size() / 2; ++i) {
                char tmp = s[i];
                s[i] = s[j];
                s[j] = tmp;
                j--;
            }
            fs << s << endl;
            cout << "[CHILD2] " << s << endl;
            n--;
        }

        close(fd2[0]);
        close(fd2[1]);
    }

    else {

        int n;
        cout << "[PARENT] Enter number of strings" << endl;
        cin >> n;
        write(fd1[1], &n, sizeof(int));
        write(fd2[1], &n, sizeof(int));
        cout << "[PARENT] Enter " << n << " string(s): " <<
endl;

```



```

for (int i = 0; i < n; ++i) {
    string s1;
    cin >> s1;
    int k = s1.size();
    char str_ch[k];
    for (int i = 0; i < k; ++i) {
        str_ch[i] = s1[i];
    }

    int r = rand() % 10 + 1;

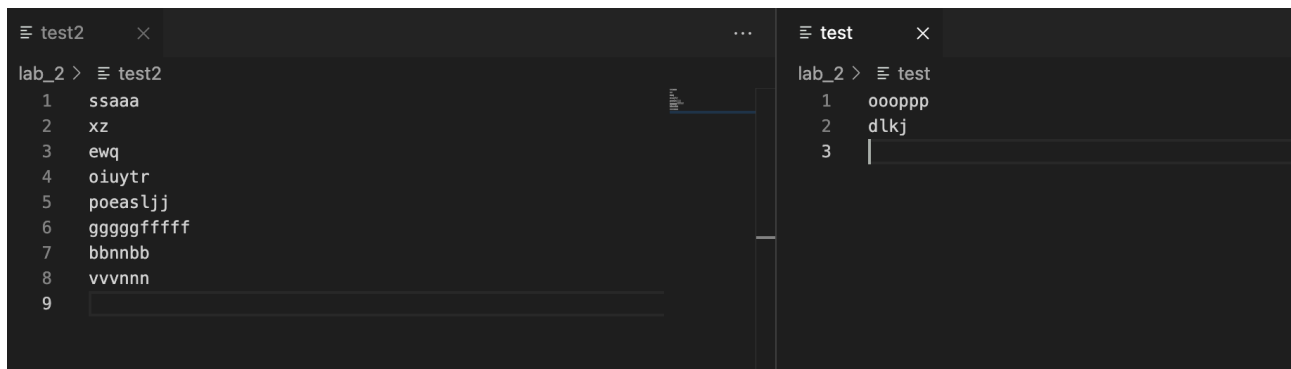
    if (r == 1 || r == 2 || r == 3) {
        write(fd1[1], &k, sizeof(int));
        write(fd1[1], str_ch, sizeof(char) * k);
    }
    else {
        write(fd2[1], &k, sizeof(int));
        write(fd2[1], str_ch, sizeof(char) * k);
    }
}

close(fd1[0]);
close(fd1[1]);
close(fd2[0]);
close(fd2[1]);
}
}
return 0;
}

```

Демонстрация работы программы

```
ivanmarinin@MacBook-Air-Ivan lab_2 % ./a.exe
Enter enter the file names:
test
test2
[PARENT] Enter number of strings
7
[PARENT] Enter 7 string(s):
aaaaasssssss
[CHILD2] sssssssaaaaa
ppppoooooo
[CHILD2] oooooopppp
hhhhhjjj
[CHILD2] jjjhhhhh
kl
[CHILD2] lk
qwe
[CHILD1]ewq
nnnnbbb
[CHILD1]bbnnnn
opp
[CHILD2] ppo
ivanmarinin@MacBook-Air-Ivan lab_2 % ./a.exe
Enter enter the file names:
test
test2
[PARENT] Enter number of strings
10
[PARENT] Enter 10 string(s):
aaass
[CHILD2] ssaaa
zx
[CHILD2] xz
qwe
[CHILD2] ewq
rtyuio
[CHILD2] oiuytr
pppooo
[CHILD1]oooppp
jkl
[CHILD1]dlkj
jllsaeop
[CHILD2] poeaslj
fffffgggg
[CHILD2] gggggfffff
bbnnbb
[CHILD2] bbnnbb
nnnvvv
[CHILD2] vvvnnn
ivanmarinin@MacBook-Air-Ivan lab_2 %
```



```
lab_2 > test2
1  ssaaa
2  xz
3  ewq
4  oiuytr
5  poeaslj
6  gggggfffff
7  bbnnbb
8  vvvnnn
9

lab_2 > test
1  oooppp
2  dlkj
3
```

Выводы

После выполнения данной лабораторной работы я с уверенностью могу сказать, что хорошо ознакомился с темой создания процессов в Linux. На примере собственного задания осознал принципы работы вышеперечисленных команд, научился ими пользоваться и даже узнал некоторые тонкости работы процессоров. Также я приобрёл навыки в обеспечении обмена данных между процессами при помощи каналов.