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: Волков Евгений Андреевич
Группа: М8О-207Б-21
Вариант: 17
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2023

Содержание

Репозиторий

https://github.com/VolkovEvgeny/OC-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.

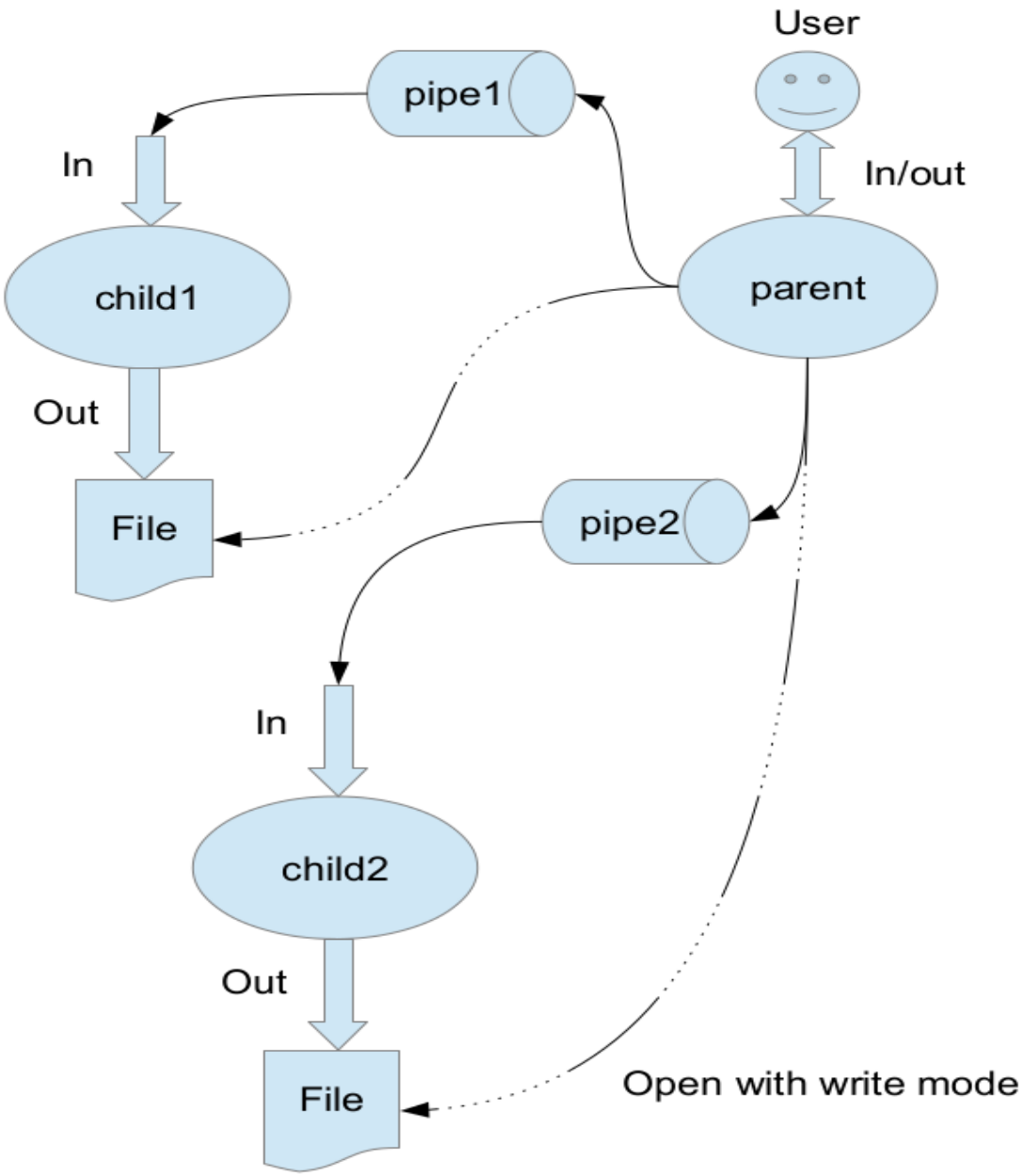
Группа вариантов 5

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.

Вариант 17

Правило фильтрации: строки длины больше 10 символов отправляются в pipe2, иначе в pipe1. Дочерние процессы удаляют все гласные из строк.

Группа вариантов 5



Общие сведения о программе

Программа компилируется из файла main.cpp. Также используется заголовочные файлы: unistd.h для всех необходимых системных вызовов, iostream для потоков ввода / вывода, string для строк, fstream для работы с файлами, set для реализации алгоритма удаления гласных букв. 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++.

Для реализации функций обеих дочерних процессов использовался 1 файл, который мы будем запускать двумя дочерними процессами с помощью вызова execlp, передавая разные параметры.

В программе main.cpp водим из терминала названия файлов для строк длиной <=10 и >10, которые потом передадим в дочерние процессы. main.cpp объявляем файловые дескрипторы для pipe1 и pipe2, с помощью системного вызова pipe() создаем 2 pipe. Потом с помощью системного вызова fork() создадим 2 дочерних процесса, с помощью системного вызова execlp() передаем их на исполнение файлу ./child, скомпилированному из child.cpp. В качестве параметров передаем файловые дескрипторы pipe1 (pipe2) и название первого (второго) файла.

В программе child.cpp принимаем из массива argv файловые дескрипторы, создаем от них pipe. Также создадим файловый поток, с помощью которого откроем необходимый файл .txt.

После этого в основном процессе в программе main.cpp считываем строки из терминала, пишем их в pipe1 если их длина <=10, или в pipe2, если длина >10 с помощью вызова write(). В это время дочерние процессы ждут ввода строки в бесконечном цикле, после получения убирают из них гласные и записывают в нужный файл.

Исходный код

main.cpp

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

using namespace std;

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

```
    string current_str; // строка, которую мы будем передавать в дочерний процесс через pipe
    string child1, child2;  // объявляем и вводим названия для двух файлов
    cout << "Enter the name for first child file: ";
    cin >> child1;
    cout << "Enter the name for second child file: ";
    cin >> child2;
    int fd1[2]; // файловые дескрипторы для pipe, связанного с первым дочерним процессом
    int fd2[2]; // файловые дескрипторы для pipe, связанного с вторым дочерним процессом

    if (pipe(fd1) == -1)    // создаем pipe для 1-го процесса
    {
        cout << "Pipe error occured" << endl;
        exit(EXIT_FAILURE);
    }
    if (pipe(fd2) == -1)    // создаем pipe для 2-го процесса
    {
        cout << "Pipe error occured" << endl;
        exit(EXIT_FAILURE);
    }

    pid_t f_id1 = fork();   // создаем первый дочерний процесс

    if (f_id1 == -1)    // если ошибка, выходим из программы
    {
        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)    // если f_id1 == 0, то мы находимся в первом дочернем процессе
    {
        // передаем первый дочерний процесс на исполнение другой программе, скомпилированной из child.cpp
        // на вход передаем файловые дескрипторы pipe1 и название первого файла
```

```
        execlp("./child", to_string(fd1[0]).c_str(), to_string(fd1[1]).c_str(), child1.c_str(),
NULL);
        perror("Execlp error"); // в случае если execlp не сработал, выводим ошибку
        return 0;   // завершаем работу дочернего процесса в этой программе
    }
    else {  // если f_id1 != 0, то мы находимся в основном процессе (f_id1 равен
id 1 дочернего процесса)
        pid_t f_id2 = fork();   // создаем второй дочерний процесс
        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)
        {
            // передаем второй дочерний процесс на исполнение другой
программе, скомпилированной из child.cpp
            // на вход передаем файловые дескрипторы pipe2 и название второго
файла
            execlp("./child", to_string(fd2[0]).c_str(), to_string(fd2[1]).c_str(),
child2.c_str(), NULL);
            perror("Execlp error"); // в случае если execlp не сработал, выводим
ошибку
            return 0;
        }

        else    // основной процесс
        {
            while (getline(std::cin, current_str))  // считываем строку из терминала до
'\n'
            {
                int s_size = current_str.size() + 1;   // получаем размер строки
                if (current_str.size() <= 10)   // отправляем в первый процесс
                {
                    write(fd1[1], &s_size, sizeof(int));    // отправляем размер строки в
pipe
```

```
                write(fd1[1], current_str.c_str(), s_size); // отправляем саму строку в
pipe
            }
            else    // во второй процесс
            {
                write(fd2[1], &s_size, sizeof(int));
                write(fd2[1], current_str.c_str(), s_size);
            }
        }
    }
    close(fd2[1]);  // закрываем файлы pipe-ов
    close(fd1[1]);
    close(fd2[0]);
    close(fd1[0]);
    return 0;
}
```

child.cpp

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

using namespace std;

int main(int argc, char const *argv[])
{
    std::string vovels = "aoueiy";  // гласные буквы
    std::set<char> volSet(vovels.begin(), vovels.end());    // создаем множество всех гласных букв
    string filename = argv[2];  // получили название файла
    int fd[2];  // создаем дескрипторы для pipe в дочернем процессе
    fd[0] = stoi(argv[0]);  // отсюда читаем
    fd[1] = stoi(argv[1]);  // сюда пишем
    fstream cur_file;   // создаем файловый поток чтобы открыть txt файл
    cur_file.open(filename, fstream::in | fstream::out | fstream::app); // открываем файл на чтение и
запись
```

```
    while (true)
    {
        int size_of_str;
        read(fd[0], &size_of_str, sizeof(int)); // получаем размер строки
        char str_array[size_of_str];
        read(fd[0], &str_array, sizeof(char) * size_of_str);    // получаем исходную строку
        string result_str; // объ
        // алгоритм удаления гласных букв
        for (int i = 0; i < size_of_str; i++) {
            if (volSet.find(std::tolower(str_array[i])) == volSet.cend()) {
                result_str.push_back(str_array[i]); // в result_str только гласные
            }
        }
        cur_file << result_str << endl; // пишем result_str в файл
    }
    return 0;
}
```

Демонстрация работы программы

```
ggame@ggame:~/hubs/newos/evgeny/OC-labs/lab2/build$ cmake ..
-- Configuring done
-- Generating done
-- Build files have been written to: /home/ggame/hubs/newos/evgeny/OC-labs/lab2/build
ggame@ggame:~/hubs/newos/evgeny/OC-labs/lab2/build$ make
Consolidate compiler generated dependencies of target child
[ 25%] Building CXX object CMakeFiles/child.dir/src/child.cpp.o
[ 50%] Linking CXX executable child
[ 50%] Built target child
Consolidate compiler generated dependencies of target main
[ 75%] Building CXX object CMakeFiles/main.dir/src/main.cpp.o
[100%] Linking CXX executable main
[100%] Built target main
ggame@ggame:~/hubs/newos/evgeny/OC-labs/lab2/build$ ./main
Enter the name for first child file: 1.txt
Enter the name for second child file: 2.txt
qwygfuyqwfyuewgufygewufgewfygweufewu
aass
vbnm
```

hkwegbiwb

aaaa

ghjhklpoooooo

dxfcgvbhjnmkdfcgvbhjnkm

s

qwerty

ggame@ggame:~/hubs/newos/evgeny/OC-labs/lab2/build$ cat 1.txt

ss

vbnm

hkwgbwb

s

qwrt

ggame@ggame:~/hubs/newos/evgeny/OC-labs/lab2/build$ cat 2.txt

qwgfqwfwgfgwfgwfgwfw

ghjhklp

dxfcgvbhjnmkdfcgvbhjnkm

В файле 1.txt строки с изначальной длиной <=10, в файле 2.txt строки с изначальной длиной >10, без гласных.

Выводы

В ходе выполнения лабораторной работы были изучены основные механизмы работы с процессами в операционной системе Unix/Linux. Был рассмотрен системный вызов fork(), который позволяет создать новый процесс. Также был изучен системный вызов pipe(), который позволяет создать канал для передачи данных между процессами. Это позволяет передавать данные между процессами, работающими параллельно. Были изучены функции execlp(), read() и write(), которые используются для выполнения новых процессов, чтения данных из канала и записи данных в канал соответственно. Эти функции помогают реализовать взаимодействие между процессами и обеспечить передачу данных между ними.