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: Меджидли Махмуд Ибрагим оглы

Группа: М8О-208Б-20

Вариант: 17

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/loshadkaigogo/OS

**Постановка задачи**

Задача: реализовать программу, в которой родительский процесс создает два дочерних процесса. Родительский процесс принимает строки, которые отправляются в тот или иной дочерний процесс в зависимости от следующего правила: если длина строки больше 10 символов, то строка отправляется во второй дочерний процесс, в противном случае в первый дочерний процесс. Оба процесса удаляют гласные из строк.

**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”)

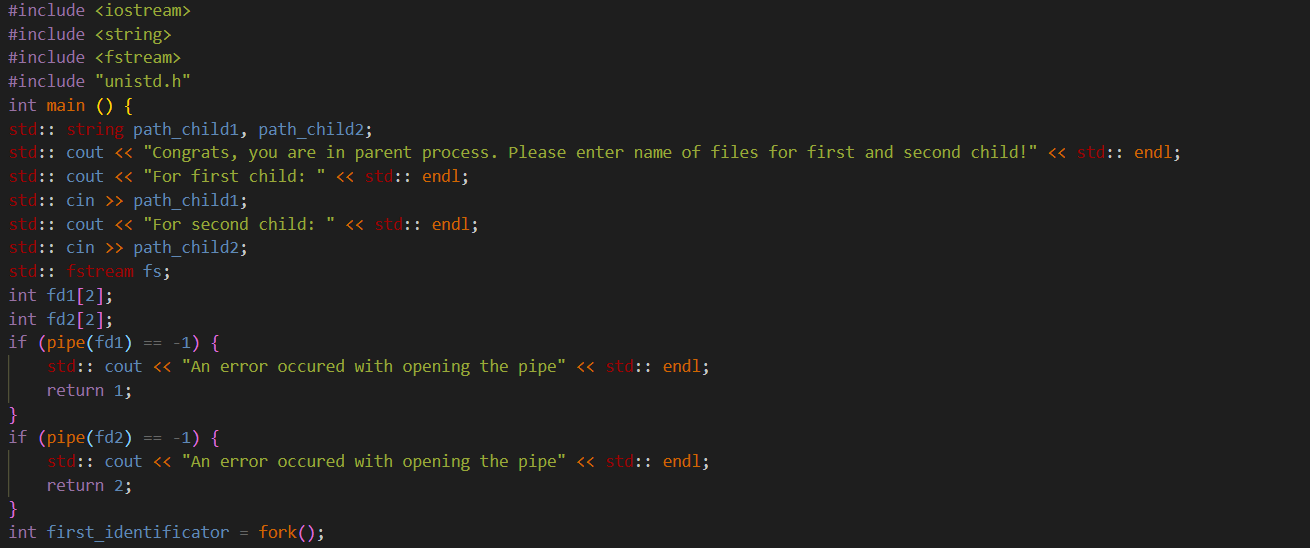
**int id = fork ()** - создание дочернего процесса, в переменной id будет лежать “специальный код” процесса (-1 - ошибка fork, 0 - дочерний процесс, >0 - родительский)  
  
**read(...)** - команда, предназначенная для чтения данных, посланных из другого процесса, принимающая на вход три параметра: элемент 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)

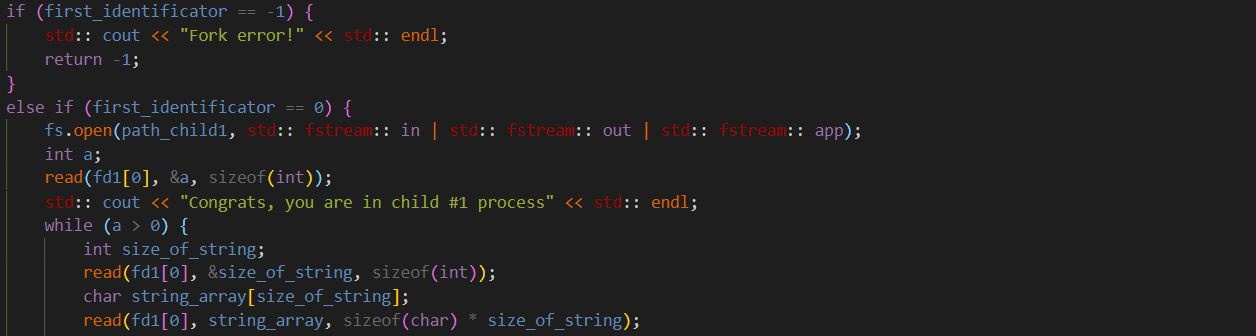
**write(...) -** команда, принимающая на вход три параметра: элемент массива дескрипторов с индексом 1, значение **посылаемого** объекта (переменной, массива и т.д.), размер **посылаемого** объекта (например, в случае переменной int - sizeof(int), в случае массива из 10 переменных типа int - sizeof(int) \* 10)  
  
**close(...)** - команда, использующаяся, когда нам больше не нужно передавать, либо считывать что-либо из другого процесса.

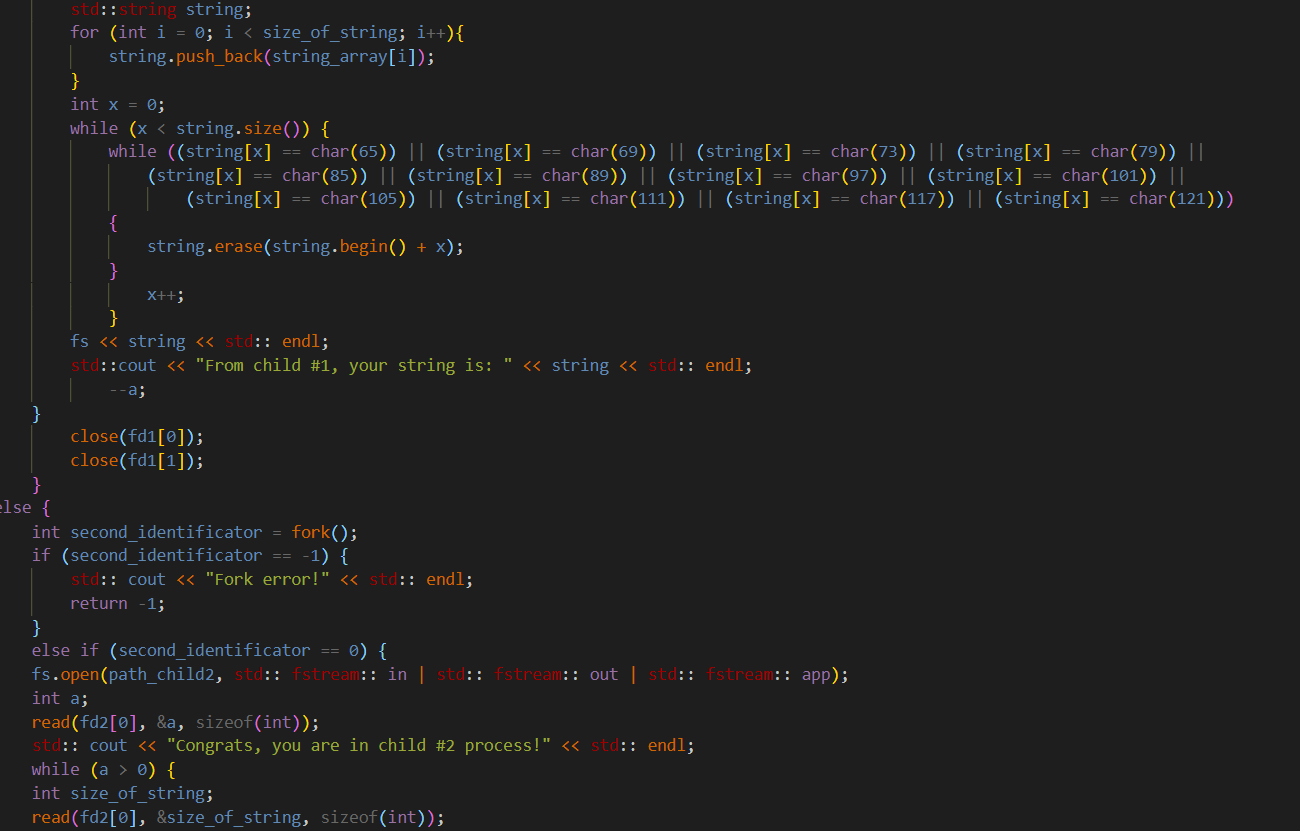
**Общий метод и алгоритм решения**

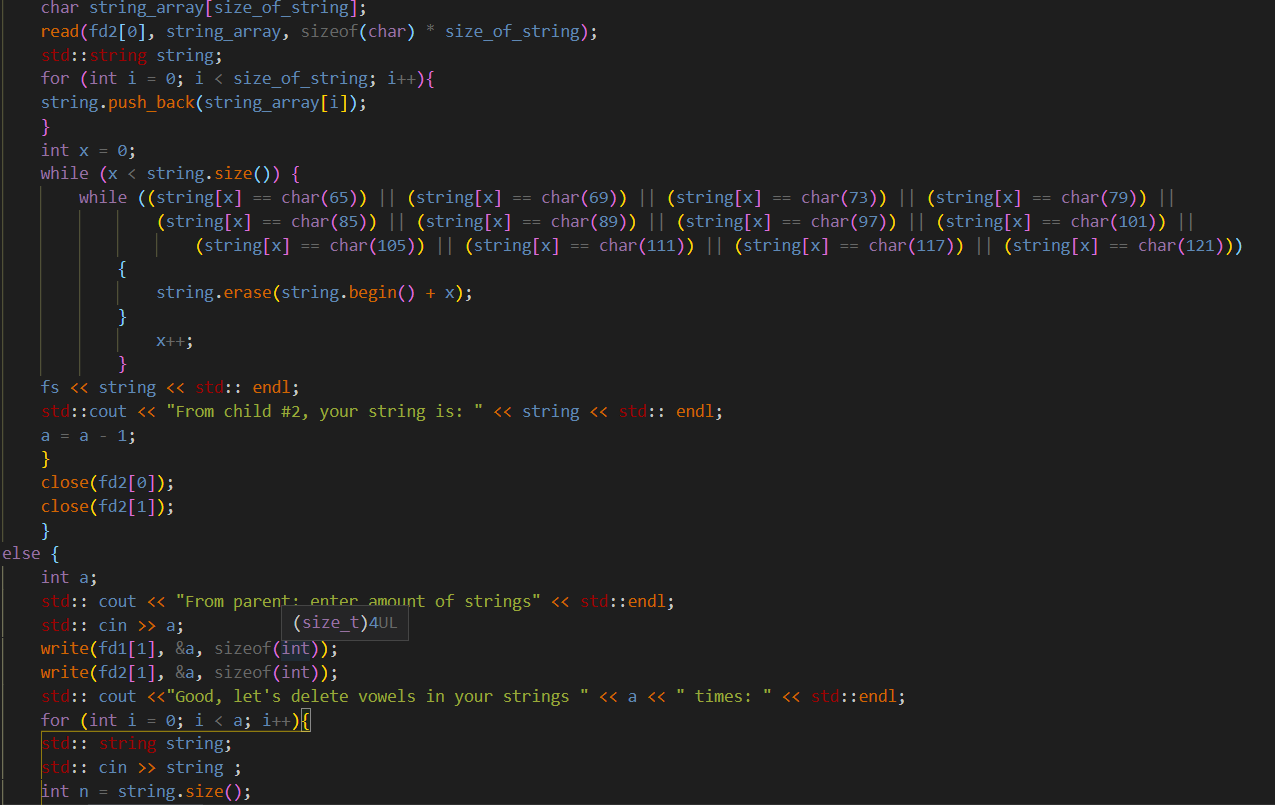
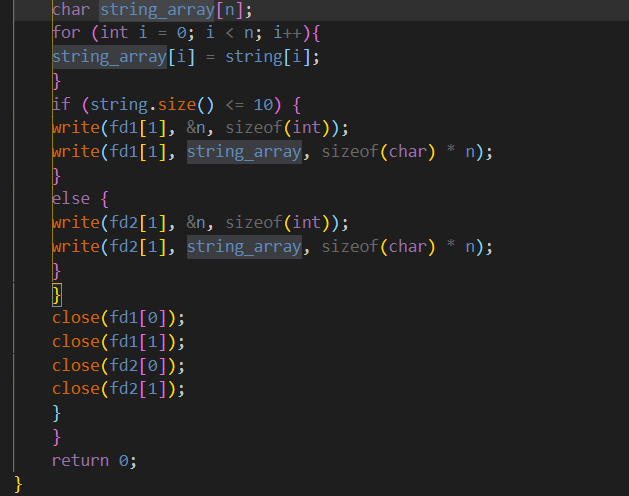
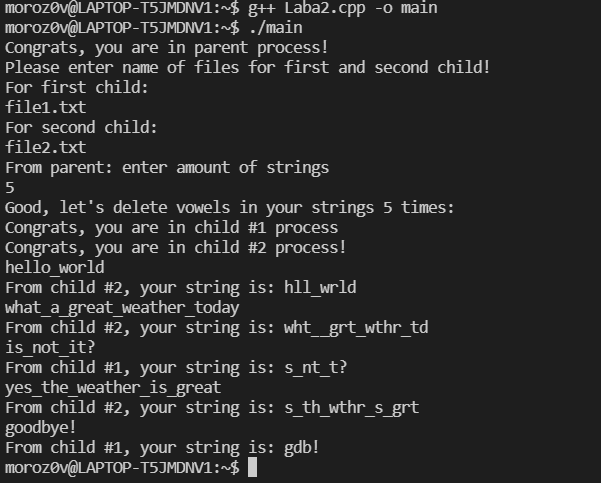
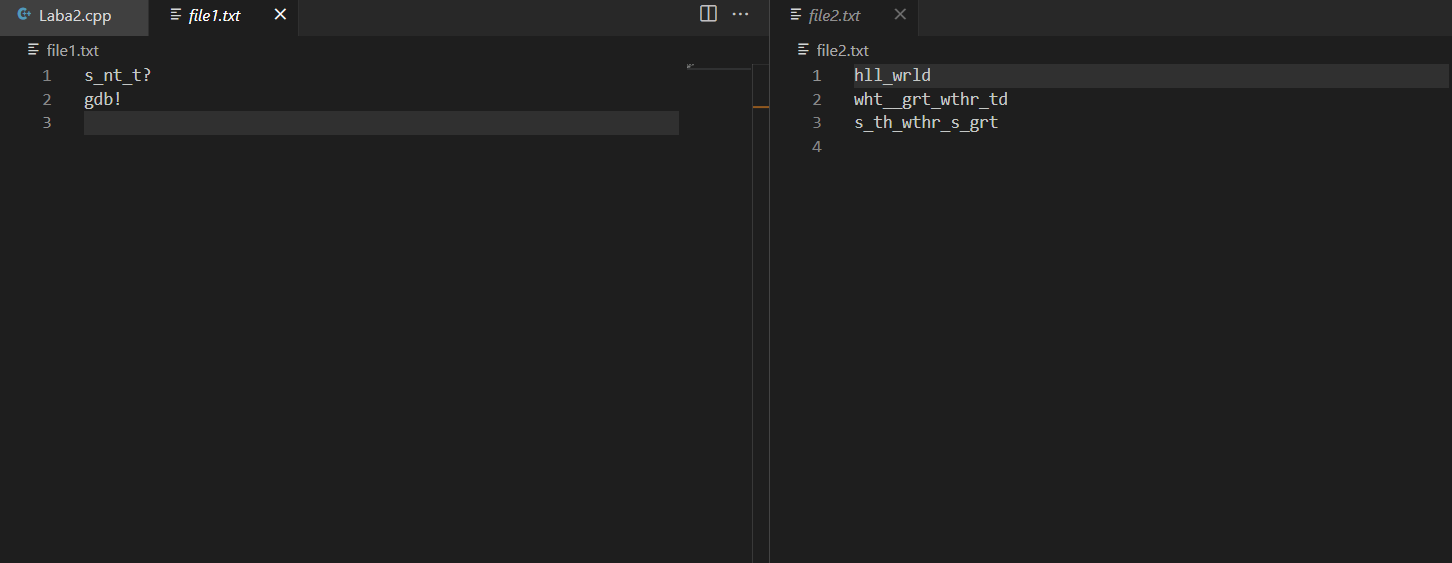
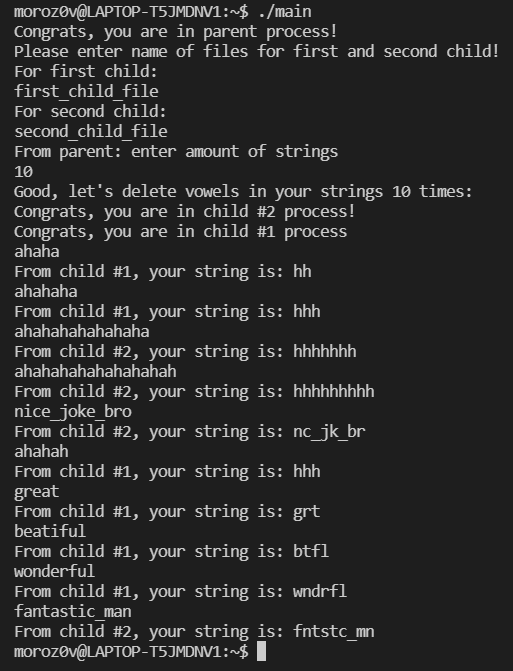
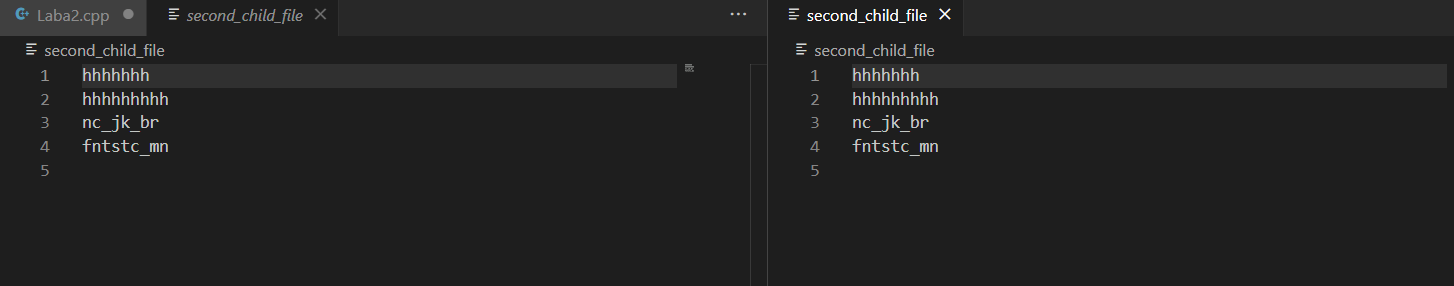
С самого начала программа получает два названия файлов для записи работы дочерних процессов. После этого эти оба файла создаются, и программа запрашивает у пользователя количество строк. Далее выполняется следующий алгоритм: после введения строки в консоль пользователь может увидеть ответ либо от первого дочернего процесса, либо от второго дочернего процесса, так как каждый процесс представляется, прежде чем вывести уже готовую строку пользователю (то есть строку с удаленными гласными). В самой программе удаление гласных представлено посредством вложенных циклов while и пробегом по строке в поиске гласной при помощи кода ASCII.  
По окончании работы программы пользователь имеет выведенные без гласных строки и в консоли, и в созданном в самом начале файле, как и требовалось в задании.  
Собирается программа при помощи команды g++ lab2.cpp -o main, запускается при помощи команды ./main.

**Исходный код**







 **  
  
  
  
Демонстрация работы программы**Тест 1. **  
**Тест 2.  
  
 **Выводы**Данная лабораторная работа помогла мне ознакомиться с тем, как устроены процессы в Linux. Я осознал принцип работы вышеперечисленных команд и системных вызовов, а также узнал некоторые тонкости работы процессоров.