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: Зинин Владислав Владимирович

Группа: М8О-208Б-20

Вариант: 5

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/frankeloff/os>

**Постановка задачи**

Задача: Родительский процесс создает дочерний процесс. Первой строчкой пользователь в консоль родительского процесса пишет имя файла, которое будет передано при создании дочернего процесса. Родительский и дочерний процесс должны быть представлены разными программами. Родительский процесс передает команды пользователя через pipe1, который связан с стандартным входным потоком дочернего процесса. Дочерний процесс при необходимости передает данные в родительский процесс через pipe2. Результаты своей работы дочерний процесс пишет в созданный им файл. Пользователь вводит команды вида: «число». Далее это число передается от родительского процесса в дочерний. Дочерний процесс производит проверку на простоту. Если число составное, то в это число записывается в файл. Если число отрицательное или простое, то тогда дочерний и родительский процессы завершаются.

**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**

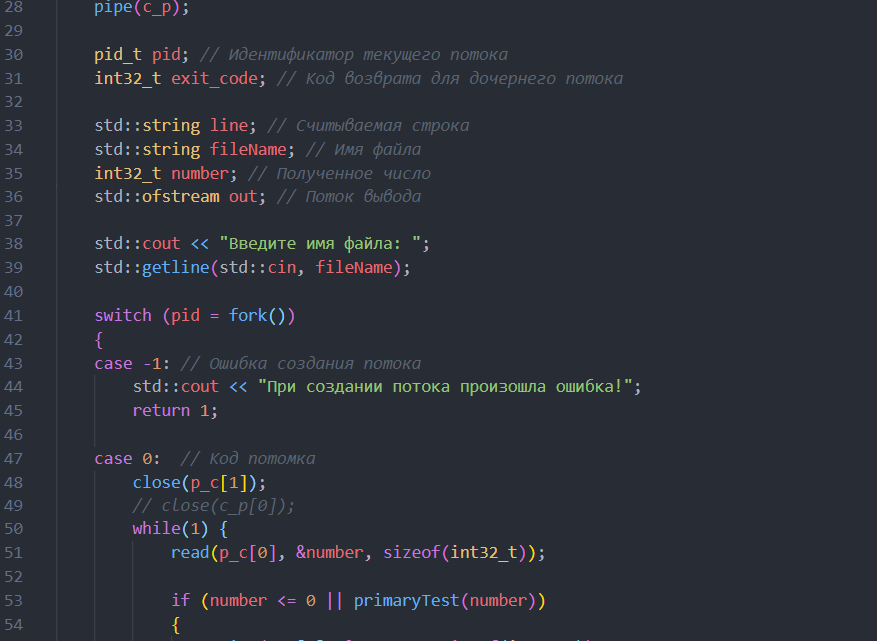
Программа получает на вход имя файла, потом число. После число проверяется в дочернем процессе, и если оно удовлетворяет условиям задачи, то создается файл и в него записывается данное число, и родительскому процессу отправляется EXITOK, который имеет значение 0. Каждое последующее число, если оно пройдет проверку, попадет в файл. Иначе родительский процесс отправляет родителю EXITNOTOK и завершается, и, вследствие чего, родительский процесс принимает значение EXITNOTOK и тоже завершается. Программа завершена.

Лабораторная работа была выполнена в среде Visual Studio code, название файла - laba2.cpp.

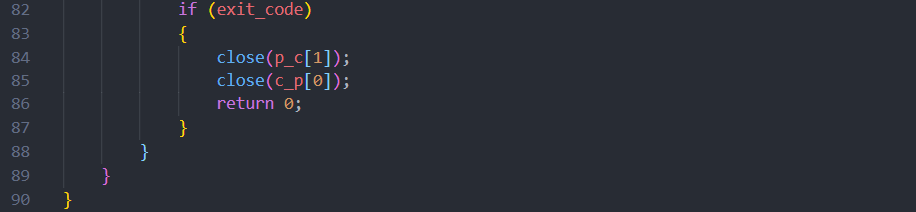
Собирается программа при помощи команды g++ Laba2.cpp, запускается при помощи команды ./a.out.

**Исходный код**

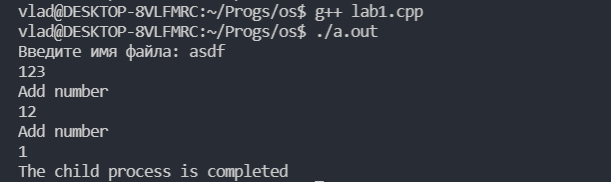
****

****

****

****

**Демонстрация работы программы**



**Выводы**

После выполнения данной лабораторной работы я с уверенностью могу сказать, что хорошо ознакомился с темой создания процессов в Linux. Я на примере собственного задания осознал принципы работы вышеперечисленных команд pipe, fork, write, read, close, научился ими пользоваться и даже познал некоторые тонкости (например, все, что было создано в родительском процессе, есть и в дочернем). Уверен, что полученные навыки помогут мне дальше осваивать курс по операционным системам.