**МИНОБРНАУКИ РОССИИ**

**САНКТ–ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра ВТ**

**ОТЧЕТ**

**по лабораторной работе №3**

**по дисциплине *«Организация процессов и программирование в среде Linux»***

**Тема: «Создание и идентификация процессов»**

Cтудентка гр. 7306 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Мищенко А.В.

Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Разумовский Г.В.

Санкт–Петербург

2020

**Цель работы:**

Изучение и использование системных функций, обеспечивающих порождение и идентификацию процессов.

**Задание:**

Разработать программу, которая порождает 2 потомка. Первый потом порождается с помощью fork, второй - с помощью vfork с последующей заменой на другую программу. Все 3 процесса должны вывести в один файл свои атрибуты с предварительным указанием имени процесса (например: Предок, Потомок1, Потомок2). Имя выходного файла задается при запуске программы. Порядок вывода атрибутов в файл должен определяться задержками процессов, которые задаются в качестве параметров программы и выводятся в начало файла.

Откомпилировать программу и запустить ее 3 раза с различными сочитаниями задержек.

**Ход работы:**

Для выполнения лабораторной работы было написано приложение, состоящее из трех файлов: main.cpp - исходный код главной программы, создающая процессы-потомки, writeFile.cpp - исходный код программы, которую будет выполнять процесс-потомок, порожденный функцией vfork, и вспомогательный файл body.h, в которой для удобства определена функция, записывающая в файл необходимую информацию. Код каждого из файлов приведен в приложении с соответствующими комментариями.

При запуске приложения пользователю необходимо ввести имя файла, время задержки в секундах для процесса-родителя, первого потомка (fork) и второго потомка (vfork). На рисунке 1 показан ввод имени файла file.txt, задержки в 4 секунды для родительского процесса, 3 секунды для первого процесса-потомка и 2 секунды для второго процесса-потомка.

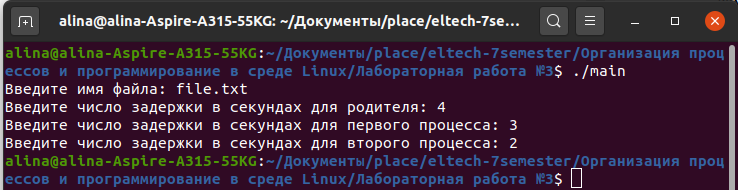


Рисунок 1. Запуск приложения с вводом задержек 4, 3, 2

Соответственно на рисунке 2 показано содержимое файла file.txt при текущих настройках задержки.

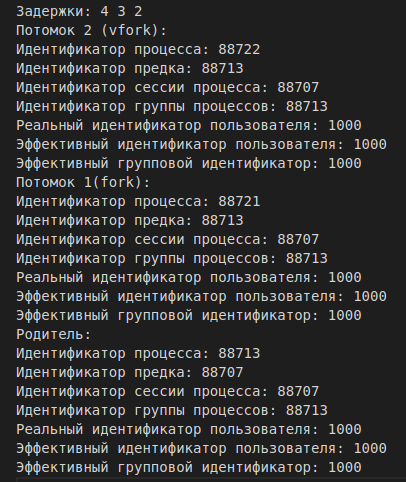


Рисунок 2. Содержимое файла file.txt при задержках 4, 3, 2

Раньше всех сработал второй процесс-потомок, так как его задержка меньше остальных процессов, потом выполнился первый потомок, а затем процесс-родитель, поскольку он имел самую большую задержку.

Если же мы укажем 0 секунд задержки для родителя, 2 секунды для первого потомка и 5 секунд для второго потомка, то увидим соответствующий порядок вызовов на рисунке 3.

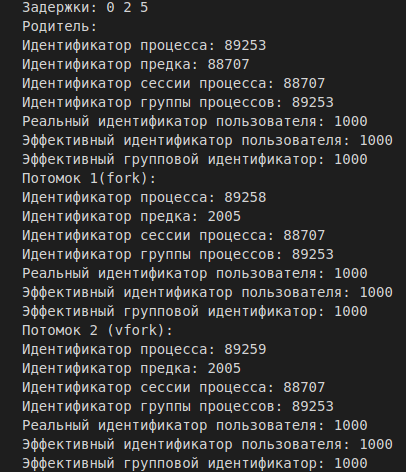


Рисунок 3. Содержимое файла file.txt при задержках 0, 2, 5

И наконец, если указать 3 секунды задержки для родительского процесса, 1 секунду для первого процесса и 5 секунд для второго процесса, то сначала выполнится процесс с меньшей задержкой - первый потомок, затем родитель и второй потомок, имеющий наибольшую задержку (Рисунок 4)

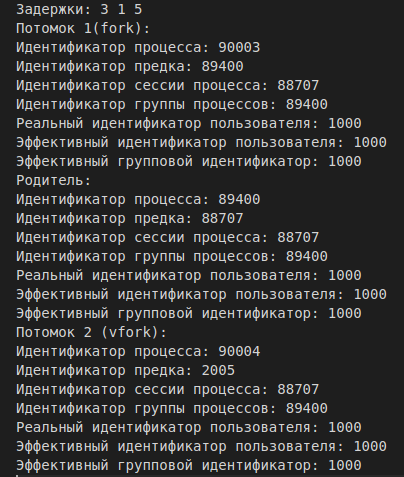


Рисунок 4. Содержимое файла file.txt при задержках 3, 1, 5

**Вывод:**

В ходе лабораторной работы я научилась создавать новые процессы с помощью функций vfork и fork, выполнять другую скомпилированную программу из основной программы и передавать ей аргументы с помощью функции exec и задерживать процесс, передавая процессорное время другому процессу, с помощью функции sleep.

**Приложения**

Приложение 1. Файл main.cpp

#include <iostream>

#include "body.h"

#include "sys/types.h"

using namespace std;

int main(){

char nameFile[20], //имя файла

delayParent[3], //задержка родителя в секундах

delayVFork[3], //задержка первого процесса vfork в секундах

delayFork[3]; //задержка второго процесса fork в секундах

//Вводим имя файла с консоли

cout << "Введите имя файла: ";

cin >> nameFile;

//Вводим числа задержек с консоли

cout << "Введите число задержки в секундах для родителя: ";

cin >> delayParent;

cout << "Введите число задержки в секундах для первого процесса: ";

cin >> delayFork;

cout << "Введите число задержки в секундах для второго процесса: ";

cin >> delayVFork;

//открываем файл nameFile, предварительно его очистив

ofstream file(nameFile, ios\_base::trunc);

//если успешно его открыли

if(file.is\_open()){

//записываем в него времена задержек

file << "Задержки: " << delayParent << " " << delayFork << " " << delayVFork << "\n";

//закрываем файл

file.close();

//создаем процесс с помощью fork()

if(fork() == 0){

//останавливаем процесс первого потомка с соответствии с введенным временем, другие процессы могут выполняться

sleep(atoi(delayFork));

//запускаем функцию body, которая записываем в файл информацию о процессе

body(nameFile, "Потомок 1(fork)");

}

//создаем процесс с помощью vfork()

else if (vfork() == 0){

//инициализируем аргументы для передачи в функцию main программы writeFile

char\* argv[] = {nameFile, delayVFork, "Потомок 2 (vfork)", nullptr};

//передаем управление в программу writeFile

execv("./writeFile",argv);

}

else {

//останавливаем процесс родителя с соответствии с введеным временем, другие процессы могут выполняться

sleep(atoi(delayParent));

//запускаем функцию body, которая записывает в файл информацию о процессе

body(nameFile, "Родитель");

}

}

else //если по каким-то причинам файл не удалось открыть - выводим сообщение об ошибке

cout << "Не удалось создать файл!";

return 0;

}

Приложение 2. Файл writeFile.cpp

#include <stdlib.h>

#include "body.h"

using namespace std;

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

sleep(atoi(argv[1])); //задержка

//запускаем функцию body, котора запишет информацию о процессе в файл

body(argv[0], argv[2]);

exit(EXIT\_SUCCESS);

}

Приложение 3. Файл body.h

include <string>

#include <fstream>

#include "unistd.h"

using namespace std;

//функция записи в файл информации о процессе

void body(char\* nameFile, char\* process){

ofstream file(string(nameFile), ios\_base::app); //открываем файл для записи

if(file.is\_open()){ //если файл успешно открыт

file << process << ":\n" << //записываем тип процесса: Родитель/Потомок

"Идентификатор процесса: " << getpid() << "\n" <<

"Идентификатор предка: " << getppid() << "\n" <<

"Идентификатор сессии процесса: " << getsid(getpid()) << "\n" <<

"Идентификатор группы процессов: " << getpgid(getpid()) << "\n" <<

"Реальный идентификатор пользователя: " << getuid() << "\n" <<

"Эффективный идентификатор пользователя: " << getgid() << "\n" <<

"Эффективный групповой идентификатор: " << getegid() << "\n";

file.close();

}

}