## МИНОБРНАУКИ РОССИИ

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

## «ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

**Кафедра Вычислительной техники**

## ОТЧЕТ

**по лабораторной работе №3**

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

**Тема: Создание и идентификация процессов**

|  |  |  |
| --- | --- | --- |
| Студент гр. 9308 |  | Яловега Н.В. |
| Преподаватель |  | Разумовский Г.В. |

Санкт-Петербург

2022

# Цель работы.

Целью лабораторной работы является изучение и использование системных функций, обеспечивающих порождение и идентификацию процессов.

# Задание.

Разработать программу, которая порождает 2 потомка. Первый потомок 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 сочетаниями задержек.

# Выполнение работы

Компиляция программ:

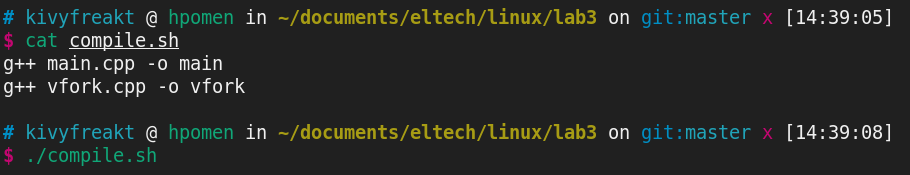


Рисунок 1. Компиляция программ

Запуск программы с тремя разными значениями задержек:

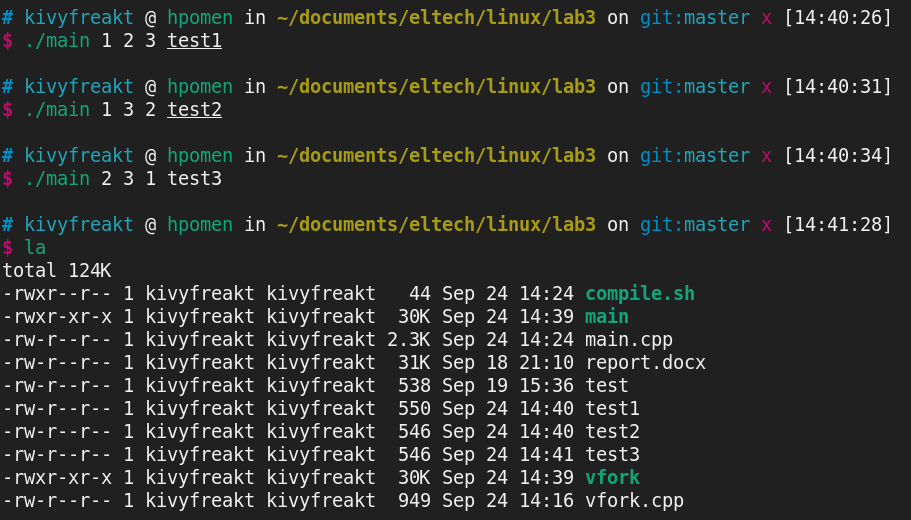
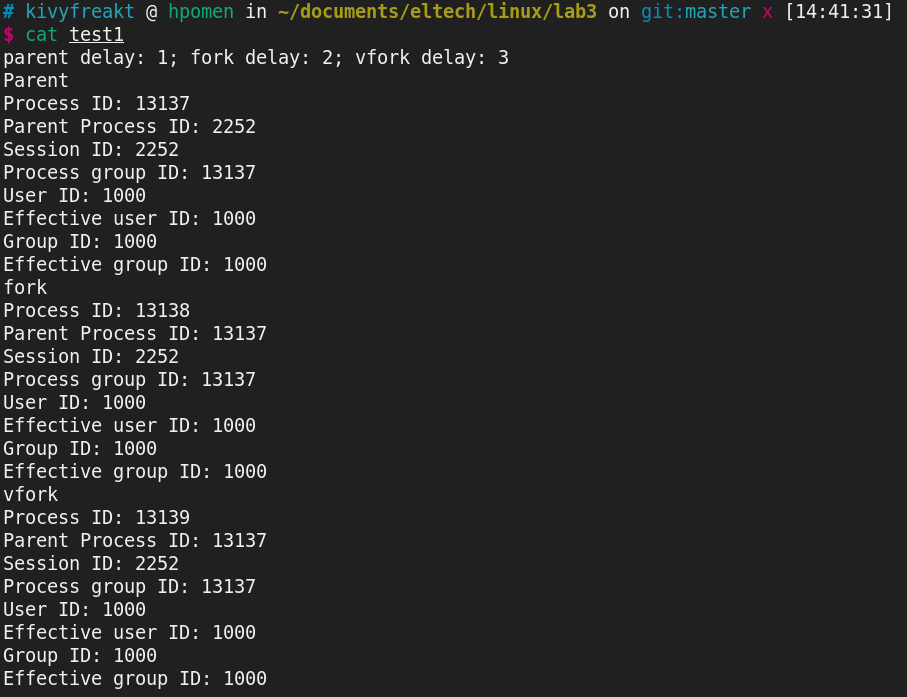


Рисунок 2. Запуск программ

Результат первого запуска представлен на рисунке 3. В программе-родителе сначала создается процесс с помощью fork, этот процесс начинает ждать 2 секунды. Затем родитель создает процесс через vfork, который ждет 3 секунды. После чего программа родитель сама ожидает 1 секунду. После истечения этого времени она пишет в файл. Потом истекает ожидание потомка fork, и он пишет в файл вторым. Третим пишет потомок vfork, так как у него наибольшее ожидание.



*Рисунок 3. Результат выполнения*

Результат второго запуска представлен на рисунке 4. В программе-родителе сначала создается процесс с помощью fork, этот процесс начинает ждать 3 секунды. Затем родитель создает процесс через vfork, который ждет 2 секунды. После чего программа родитель сама ожидает 1 секунду. После истечения этого времени она пишет в файл. Потом истекает ожидание потомка vfork, и он пишет в файл вторым. После завершения vfork, программа родитель сама завершает работу. Третим пишет потомок fork, так как у него наибольшее ожидание. У этого потомка можно заметить PPID равным 1, так как на момент завершения программы, процесса родителя уже не существовало.

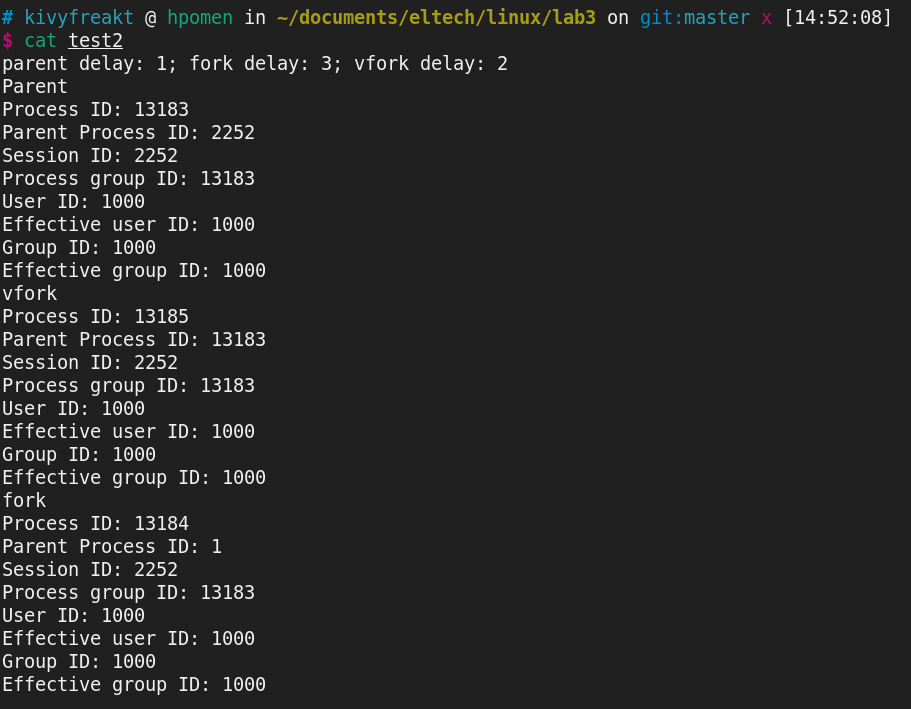


Рисунок 4. Результат выполнения

Результат второго запуска представлен на рисунке 4. В программе-родителе сначала создается процесс с помощью fork, этот процесс начинает ждать 3 секунды. Затем родитель создает процесс через vfork, который ждет 1 секунду. После чего программа родитель сама ожидает 2 секунды. Первым пишет в файл потомок vfork. Потом истекает ожидание родителя и он пишет в файл вторым. Так же программа родитель завершает работу. Третим пишет потомок fork, так как у него наибольшее ожидание. У этого потомка можно заметить PPID равным 1, так как на момент завершения программы, процесса родителя уже не существовало.

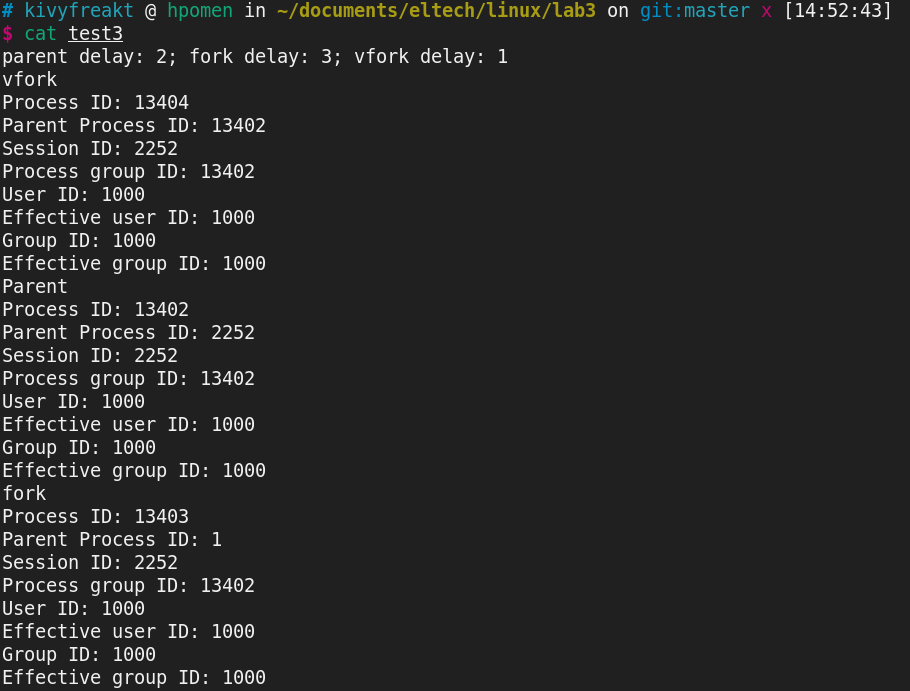


Рисунок 5. Результат выполнения

# Вывод

При выполнении лабораторной работы мы ознакомились с функциями создания процессов в системе Linux, а также с особенностями их работы и взаимоотношений предков с потомками. Кроме того мы смогли произвести запуск кода сторонней программы из основной, используя при этом дочерний процесс.

# Приложение

# main.cpp

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <cstdlib>

#include <iostream>

#include <sstream>

void printAttributes(std::string process\_name, FILE\* file)

{

std::stringstream out;

pid\_t pid = getpid();

out << process\_name << std::endl;

out << "Process ID: " << pid << std::endl;

out << "Parent Process ID: " << getppid() << std::endl;

out << "Session ID: " << getsid(pid) << std::endl;

out << "Process group ID: " << getpgid(pid) << std::endl;

out << "User ID: " << getuid() << std::endl;

out << "Effective user ID: " << geteuid() << std::endl;

out << "Group ID: " << getgid() << std::endl;

out << "Effective group ID: " << getegid() << std::endl;

fprintf(file, out.str().c\_str());

}

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

{

if(argc == 5)

{

int parent\_delay, fork\_delay, vfork\_delay;

parent\_delay = atoi(argv[1]);

fork\_delay = atoi(argv[2]);

vfork\_delay = atoi(argv[3]);

FILE\* file = fopen(argv[4], "w");

fprintf(file, "parent delay: %d; fork delay: %d; vfork delay: %d\n", parent\_delay, fork\_delay, vfork\_delay);

fclose(file);

if((file = fopen(argv[4], "a")))

{

pid\_t process\_fork = fork();

if(process\_fork == 0)

{

sleep(fork\_delay);

printAttributes("fork", file);

return 0;

}

if(process\_fork < 0)

std::cout << "fork process was not created" << std::endl;

pid\_t process\_vfork = vfork();

if(process\_vfork == 0)

execl("vfork", "vfork", argv[3], argv[4], NULL);

if(process\_vfork < 0)

std::cout << "vfork process was not created" << std::endl;

sleep(parent\_delay);

printAttributes("Parent", file);

fclose(file);

waitpid(process\_vfork, NULL, 0);

}

else

std::cout << "Error opening file " << argv[4] << std::endl;

}

else

{

std::cout << "Incorrect number of arguments!\n" << std::endl;

std::cout << "Example: program parent\_delay fork\_delay vfork\_delay file\_name" << std::endl;

return 0;

}

# vfork.cpp

# #include <unistd.h>

# #include <sys/types.h>

# #include <cstdlib>

# #include <iostream>

# #include <sstream>

# void printAttributes(std::string process\_name, FILE\* file)

# {

# std::stringstream out;

# pid\_t pid = getpid();

# out << process\_name << std::endl;

# out << "Process ID: " << pid << std::endl;

# out << "Parent Process ID: " << getppid() << std::endl;

# out << "Session ID: " << getsid(pid) << std::endl;

# out << "Process group ID: " << getpgid(pid) << std::endl;

# out << "User ID: " << getuid() << std::endl;

# out << "Effective user ID: " << geteuid() << std::endl;

# out << "Group ID: " << getgid() << std::endl;

# out << "Effective group ID: " << getegid() << std::endl;

# fprintf(file, out.str().c\_str());

# }

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

# {

# FILE \*file;

# if (file = fopen(argv[2], "a"))

# {

# sleep(atoi(argv[1]));

# printAttributes("vfork", file);

# }

# fclose(file);

# return 0;

# }