

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
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:

```
# kivyfreakt @ hpomen in ~/documents/eltech/linux/lab3 on git:master x [14:39:05]
$ cat compile.sh
g++ main.cpp -o main
g++ vfork.cpp -o vfork

# kivyfreakt @ hpomen in ~/documents/eltech/linux/lab3 on git:master x [14:39:08]
$ ./compile.sh
```

Рисунок 1. Компиляция программ

Запуск программы с тремя разными значениями задержек:

```
# kivyfreakt @ hpomen in ~/documents/eltech/linux/lab3 on git:master x [14:40:26]
$ ./main 1 2 3 test1

# kivyfreakt @ hpomen in ~/documents/eltech/linux/lab3 on git:master x [14:40:31]
$ ./main 1 3 2 test2

# kivyfreakt @ hpomen in ~/documents/eltech/linux/lab3 on git:master x [14:40:34]
$ ./main 2 3 1 test3

# kivyfreakt @ hpomen in ~/documents/eltech/linux/lab3 on git:master x [14:41:28]
$ la
total 124K
-rwxr--r-- 1 kivyfreakt kivyfreakt  44 Sep 24 14:24 compile.sh
-rwxr-xr-x 1 kivyfreakt kivyfreakt 30K Sep 24 14:39 main
-rw-r--r-- 1 kivyfreakt kivyfreakt 2.3K Sep 24 14:24 main.cpp
-rw-r--r-- 1 kivyfreakt kivyfreakt 31K Sep 18 21:10 report.docx
-rw-r--r-- 1 kivyfreakt kivyfreakt  538 Sep 19 15:36 test
-rw-r--r-- 1 kivyfreakt kivyfreakt  550 Sep 24 14:40 test1
-rw-r--r-- 1 kivyfreakt kivyfreakt  546 Sep 24 14:40 test2
-rw-r--r-- 1 kivyfreakt kivyfreakt  546 Sep 24 14:41 test3
-rwxr-xr-x 1 kivyfreakt kivyfreakt 30K Sep 24 14:39 vfork
-rw-r--r-- 1 kivyfreakt kivyfreakt  949 Sep 24 14:16 vfork.cpp
```

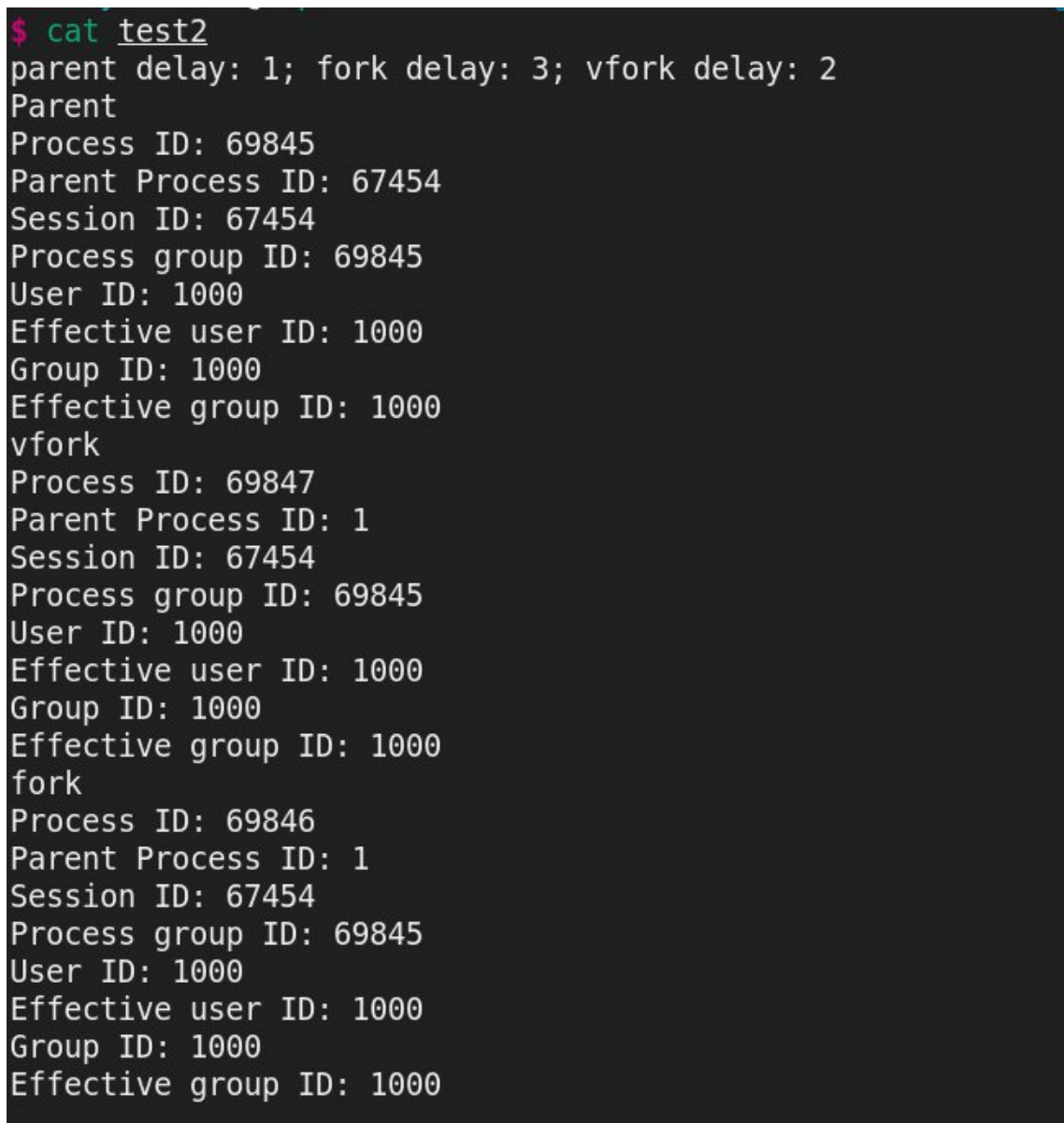
Рисунок 2. Запуск программ

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.

```
$ cat test1
parent delay: 1; fork delay: 2; vfork delay: 3
Parent
Process ID: 69603
Parent Process ID: 67454
Session ID: 67454
Process group ID: 69603
User ID: 1000
Effective user ID: 1000
Group ID: 1000
Effective group ID: 1000
fork
Process ID: 69604
Parent Process ID: 1
Session ID: 67454
Process group ID: 69603
User ID: 1000
Effective user ID: 1000
Group ID: 1000
Effective group ID: 1000
vfork
Process ID: 69605
Parent Process ID: 1
Session ID: 67454
Process group ID: 69603
User ID: 1000
Effective user ID: 1000
Group ID: 1000
Effective group ID: 1000
```

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. Третий 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.



```
$ cat test2
parent delay: 1; fork delay: 3; vfork delay: 2
Parent
Process ID: 69845
Parent Process ID: 67454
Session ID: 67454
Process group ID: 69845
User ID: 1000
Effective user ID: 1000
Group ID: 1000
Effective group ID: 1000
vfork
Process ID: 69847
Parent Process ID: 1
Session ID: 67454
Process group ID: 69845
User ID: 1000
Effective user ID: 1000
Group ID: 1000
Effective group ID: 1000
fork
Process ID: 69846
Parent Process ID: 1
Session ID: 67454
Process group ID: 69845
User ID: 1000
Effective user ID: 1000
Group ID: 1000
Effective group ID: 1000
```

Рисунок 4. Результат выполнения

Результат третьего 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. Также программа родитель завершает работу. Третьим 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.

```
$ cat test3
parent delay: 3; fork delay: 2; vfork delay: 1
vfork
Process ID: 69875
Parent Process ID: 69873
Session ID: 67454
Process group ID: 69873
User ID: 1000
Effective user ID: 1000
Group ID: 1000
Effective group ID: 1000
fork
Process ID: 69874
Parent Process ID: 69873
Session ID: 67454
Process group ID: 69873
User ID: 1000
Effective user ID: 1000
Group ID: 1000
Effective group ID: 1000
Parent
Process ID: 69873
Parent Process ID: 67454
Session ID: 67454
Process group ID: 69873
User ID: 1000
Effective user ID: 1000
Group ID: 1000
Effective group ID: 1000
```

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);
}
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;
}
```