

**«Санкт-Петербургский государственный электротехнический
университет «ЛЭТИ» им. В.И.Ульянова (Ленина)»
СПбГЭТУ «ЛЭТИ»**

Кафедра вычислительной техники

Отчет по по лабораторной работе №3 по дисциплине
«Организация процессов и программирование в среде Linux»
Тема: «СОЗДАНИЕ И ИДЕНТИФИКАЦИЯ ПРОЦЕССОВ»

Студент гр.8306

Преподаватель

Слепов А.Э.

Разумосвский Г.В.

Санкт-Петербург

2021

Цель работы

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

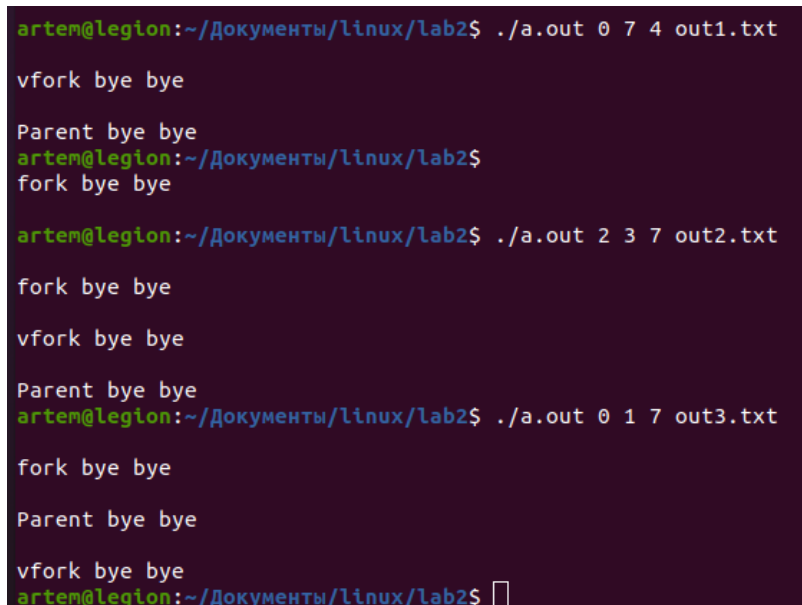
Задание

1. 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.

2. Откомпилировать программу и запустить ее 3 раза с различными сочетаниями задержек

Порядок выполнения работы

Запуск программы с разными временными задержками представлен на рисунке 1. Процессы перед завершением выводят текстовое сообщение, по ним этими сообщениями можно увидеть порядок завершения работы.



```
artem@legion:~/Документы/linux/lab2$ ./a.out 0 7 4 out1.txt
vfork bye bye
Parent bye bye
artem@legion:~/Документы/linux/lab2$
fork bye bye
artem@legion:~/Документы/linux/lab2$ ./a.out 2 3 7 out2.txt
fork bye bye
vfork bye bye
Parent bye bye
artem@legion:~/Документы/linux/lab2$ ./a.out 0 1 7 out3.txt
fork bye bye
Parent bye bye
vfork bye bye
artem@legion:~/Документы/linux/lab2$
```

Рисунок 1 – Запуск программы с разными задержками

Результат первого запуска представлен на рисунке 2. В программе-родителе сначала создается процесс с помощью `fork`, этот процесс ждет 7 секунд. Затем родитель создает процесс через `vfork`, особенность `vfork` в том, что он блокирует вызывающий процесс. Поэтому пока `vfork` ждем свои 4 секунды, родитель заблокирован. По окончании этого времени родитель выводит свои атрибуты, а процесс `vfork` передает управление другой программе через `exec1` и тоже выводит свои атрибуты. Последним в файл в пишет процесс `fork`, так как у него была выставлена самая большая задержка.

```
artem@legion:~/документы/linux/lab2$ cat out1.txt
Задержка родителя: 0; Задержка fork: 7; Задержка vfork: 4
Процесс-родитель:
PID: 16185
PPID: 3766
SID: 3766
PGID: 16185
UID: 1000
EUID: 1000
GID: 1000
EGID: 1000

vfork процесс, запущенный через подмену программы:
PID: 16187
PPID: 16185
SID: 3766
PGID: 16185
UID: 1000
EUID: 1000
GID: 1000
EGID: 1000

Процесс fork:
PID: 16186
PPID: 1468
SID: 3766
PGID: 16185
UID: 1000
EUID: 1000
GID: 1000
EGID: 1000
```

Рисунок 2 – Результаты первого запуска программы

Результат второго запуска представлен на рисунке 3. В программе-родителе сначала создается процесс с помощью `fork`, этот процесс ждет 3 секунды и первым пишет в файл. Затем родитель создает процесс через `vfork`, который `vfork` ждет 7 секунд, блокируя родителя. По окончании этого времени родитель разблокируется и переходит к своему ожиданию(2 секунды). За это время `vfork` пишет свои атрибуты. Последним в файл в пишет процесс родитель, так как он был заблокирован на время ожидания `vfork`.

```

artem@legion:~/Документы/linux/lab2$ cat out2.txt
Задержка родителя: 2; Задержка fork: 3; Задержка vfork: 7
Процесс fork:
PID: 16235
PPID: 16234
SID: 3766
PGID: 16234
UID: 1000
EUID: 1000
GID: 1000
EGID: 1000

vfork процесс, запущенный через подмену программы:
PID: 16236
PPID: 16234
SID: 3766
PGID: 16234
UID: 1000
EUID: 1000
GID: 1000
EGID: 1000

Процесс-родитель:
PID: 16234
PPID: 3766
SID: 3766
PGID: 16234
UID: 1000
EUID: 1000
GID: 1000
EGID: 1000

```

Рисунок 3 – Результаты второго запуска программы

Результат третьего запуска представлен на рисунке 4. В программе-родителе сначала создается процесс с помощью `fork`, этот процесс ждет 1 секунду и первым пишет в файл. Затем родитель создает процесс через `vfork`, который `vfork` ждет 7 секунды, блокируя родителя. По окончании этого времени родитель разблокируется и сразу пишет в файл. В это время `vfork` передает управление другой программе, которая пишет в файл последней.

```

artem@legion:~/Документы/linux/lab2$ cat out3.txt
Задержка родителя: 0; Задержка fork: 1; Задержка vfork: 7
Процесс fork:
PID: 16254
PPID: 16253
SID: 3766
PGID: 16253
UID: 1000
EUID: 1000
GID: 1000
EGID: 1000

Процесс-родитель:
PID: 16253
PPID: 3766
SID: 3766
PGID: 16253
UID: 1000
EUID: 1000
GID: 1000
EGID: 1000

vfork процесс, запущенный через подмену программы:
PID: 16255
PPID: 1468
SID: 3766
PGID: 16253
UID: 1000
EUID: 1000
GID: 1000
EGID: 1000

```

Рисунок 4 – Результаты третьего запуска программы

Выводы

В ходе работы были изучены механизмы порождения процессов с помощью функций `fork` и `vfork`, а также были изучены функции идентификации процессов, которые позволяют узнать `id` процесса, его родителя и другие атрибуты процессов.

ПРИЛОЖЕНИЕ А

Текст программы-родителя

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types>

void print_attributes(char* processName, FILE* file) {
    pid_t pid = getpid();
    fprintf(file, "%s:\nPID: %d\nPPID: %d\nSID: %d\nPGID: %d\nUID: %d\nEUID: %d\nGID: %d\nEGID: %d\n\n",
            processName, pid, getppid(), getsid(pid),
            getpgid(pid), getuid(), geteuid(), getgid(), getegid());
}

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, "Задержка родителя: %d; Задержка fork: %d; Задержка
vfork: %d\n", parent_delay, fork_delay, vfork_delay);
        fclose(file); //Открыли файл, очистили содержимое, вывели новые
времена задержек

        if((file = fopen(argv[4], "a")) ){ //открыли файл в режиме append,
дескриптор файла наследуется процессами, поэтому будем передавать его
        pid_t processFork = fork();
        if(processFork == 0){ //если мы находимся в потоке-приемнике
fork
            sleep(fork_delay);
            print_attributes("Процесс fork", file);
            printf("\nfork bye bye\n");
            exit(EXIT_SUCCESS);
        }
        if(processFork < 0) printf("fork процесс не был создан");
        pid_t processVfork = vfork();
        if(processVfork == 0){ //если мы находимся в потоке-приемнике
vfork
            sleep(vfork_delay);
            execl("lab2_1", "lab2_1", argv[4], NULL);
        }
        if(processVfork < 0) printf("vfork процесс не был создан");
        sleep(parent_delay);
        print_attributes("Процесс-родитель", file);
        fclose(file);
        printf("\nParent bye bye\n");
    }
    else{
        printf("Ошибка открытия файла %s!", argv[4]);
    }
}
else{
    printf("Неправильное число аргументов!\n");
    printf("Формат вызова: задержка_родителя задержка_fork
задержка_vfork имя_файла\n");
}

    return 0;
}
```

ПРИЛОЖЕНИЕ Б

Текст программы-подмены vfork

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types>

void print_attributes(char* processName, FILE* file) {
    pid_t pid = getpid();
    fprintf(file, "%s:\nPID: %d\nPPID: %d\nSID: %d\nPGID: %d\nUID: %d\nEUID: %d\n\n",
        processName, pid, getppid(), getsid(pid),
        getpgid(pid), getuid(), geteuid(), getgid(), getegid());
}

int main(int argc, char* argv[]){
    FILE *file;

    if((file = fopen(argv[1], "a")) ){
        print_attributes("vfork процесс, запущенный через подмену программы",
            file);
        printf("\nvfork bye bye\n");
    }
    return 0;
}
```