

ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ ПОЛІТЕХНІЧНИЙ УНІВЕРСИТЕТ  
ІНСТИТУТ КОМП'ЮТЕРНИХ СИСТЕМ  
КАФЕДРА «ІНФОРМАЦІЙНИХ СИСТЕМ»

Лабораторна робота №8

з дисципліни

«Операційні системи»

Тема: «Програмування керування процесами в ОС Unix»

Виконав:

Студент групи АІ-202

Баранюк Д.А.

Перевірили:

Блажко О.А.

Дрозд М.О.

Одеса 2021

## Завдання для виконання:

### Завдання 1 Перегляд інформації про процес

Створіть С-програму, яка виводить на екран таку інформацію:

- ідентифікатор групи процесів лідера сесії;
- ідентифікатор групи процесів, до якої належить процес;
- ідентифікатор процесу, що викликав цю функцію;
- ідентифікатор батьківського процесу;
- ідентифікатор користувача процесу, який викликав цю функцію;
- ідентифікатор групи користувача процесу, який викликав цю функцію.

Завершіть створення програми включенням функції `sleep(5)` для забезпечення засинання процесу на 5 секунд.

При створенні повідомлень використовуйте функцію `fprintf` з виведенням на потік

помилки.

Після компіляції запусить програму.

Додатково запусить програму в конвеєрі, наприклад:

`./info | ./info`

Порівняйте значення групи процесів.

 baranyuk\_dmitro@vpsj3leQ:~

GNU nano 2.3.1

File: info.c

```
#include <stdio.h>
#include <unistd.h>


int main(void) {
    fprintf(stderr, "I am process! gpid=%d\n", getpgrp());
    fprintf(stderr, "sid=%d\n", getsid(0));
    fprintf(stderr, "ppid=%d\n", getppid());
    fprintf(stderr, "uid=%d\n", getuid());
    fprintf(stderr, "gid=%d\n", getgid());
    return 0;
}
```

```
[baranyuk_dmitro@vpsj3IeQ ~]$ gcc info.c -o info
[baranyuk_dmitro@vpsj3IeQ ~]$ ./info ; ./info
I am process! gpid=27822
sid=25674
ppid=25674
uid=54349
git=54355
I am process! gpid=27823
sid=25674
ppid=25674
uid=54349
git=54355
[baranyuk_dmitro@vpsj3IeQ ~]$ ./info | ./info
I am process! gpid=27844
sid=25674
ppid=25674
uid=54349
git=54355
I am process! gpid=27844
sid=25674
ppid=25674
uid=54349
git=54355
```

## Завдання 2 Стандартне створення процесу

Створіть С-програму, яка створює процес-нащадок, породжуючи процес та замінюючи образ процесу. У програмі процес-батько повинен видати повідомлення типу

«Parent of Ivanov», а процес-нащадок повинен видати повідомлення типу «Child of Ivanov» через виклик команди echo, де замість слова Ivanov в повідомленні повинно бути ваше прізвище в транслітерації.

 baranyuk\_dmitro@vpsj3IeQ:~

GNU nano 2.3.1

File: create.c

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

int main(void) {
    pid_t pid = fork();
    if(pid==0)
        printf("I am child! pid=%d. ppid=%d\n",getpid(),getppid());
    else
        printf("I am parent! pid=%d, ghild pid=%d\n",getpid(),pid);
    return 0;
}
```

```
[baranyuk_dmitro@vpsj3IeQ ~]$ nano create.c
[baranyuk_dmitro@vpsj3IeQ ~]$ gcc create.c -o create
[baranyuk_dmitro@vpsj3IeQ ~]$ ./create
I am parent! pid=31017, ghild pid=31018
I am child! pid=31018. ppid=31017
[baranyuk_dmitro@vpsj3IeQ ~]$
```

baranyuk\_dmitro@vpsj3IeQ:~

GNU nano 2.3.1

File: create.c

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

extern char** environ;

int main(void) {
    char* echo_args[]={"echo","I am ECHO!\n",NULL};
    pid_t pid = fork();
    if(pid==0){
        printf("I am parent! pid=%d. ppid=%d\n",getpid(),getppid());
        execve("/bin/echo",echo_args,environ);
        fprintf(stderr,"Error!\n");
    }
    return 0;
}
```

```
[baranyuk_dmitro@vpsj3IeQ ~]$ nano create.c
[baranyuk_dmitro@vpsj3IeQ ~]$ gcc create.c -o create
[baranyuk_dmitro@vpsj3IeQ ~]$ ./create
I am parent! pid=32199. ppid=32198
[baranyuk_dmitro@vpsj3IeQ ~]$ I am ECHO!
```

```
[baranyuk_dmitro@vpsj3IeQ ~]$
```

baranyuk\_dmitro@vpsj3IeQ:~

GNU nano 2.3.1

File: create.c

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

extern char** environ;

int main(void) {
    char* echo_args[]={"echo","Child of Baranyuk!\n",NULL};
    pid_t pid = fork();
    if(pid==0){
        printf("Parent of Baranyuk! pid=%d. ppid=%d\n",getpid(),getppid());
        execve("/bin/echo",echo_args,environ);
        fprintf(stderr,"Error!\n");
    }
    return 0;
}
```

```
[baranyuk_dmitro@vpsj3IeQ ~]$ nano create.c
[baranyuk_dmitro@vpsj3IeQ ~]$ gcc create.c -o create
[baranyuk_dmitro@vpsj3IeQ ~]$ ./create
Parent of Baranyuk! pid=14902. ppid=14901
[baranyuk_dmitro@vpsj3IeQ ~]$ Child of Baranyuk!
```

### Завдання 3 Обмін сигналами між процесами

3.1 Створіть С-програму, в якій процес очікує отримання сигналу SIGUSR2 та виводить повідомлення типу «Process of Ivanov got signal» після отримання сигналу, де замість слова Ivanov в повідомленні повинно бути ваше прізвище в транслітерації.

Запустіть створену С-програму.

3.2 Створіть С-програму, яка надсилає сигнал SIGUSR2 процесу, запущеному в попередньому пункту завдання. Запустіть створену С-програму та проаналізуйте повідомлення, які виводить перша

програма. Завершіть процес, запущеному в попередньому пункту завдання.

baranyuk\_dmitro@vpsj3IeQ:~

GNU nano 2.3.1

File: get\_signal.c

```
#include <signal.h>
#include <stdio.h>
#include <unistd.h>

static void sig_usr(int signo){
    if(signo == SIGUSR1)
        fprintf(stderr, "Got signal %d\n", SIGUSR1);
}

int main(void){
    printf("pidd=%d\n", getpid());
    if(signal(SIGUSR1, sig_usr) == SIG_ERR)
        fprintf(stderr, "Error!\n");
    for(;;)
        pause();
    return 0;
}
```

baranyuk\_dmitro@vpsj3IeQ:~

GNU nano 2.3.1

File: send

```
#include <signal.h>
#include <stdio.h>

pid_t pid = 13373 ;
int main(void){
    if(!kill(pid, SIGUSR1))
        printf("OK!\n");
    else
        fprintf(stderr, "Error!\n");
    return 0;
}
```

[baranyuk\_dmitro@vpsj3IeQ ~]\$ gcc send\_signal.c -o send\_signal

[baranyuk\_dmitro@vpsj3IeQ ~]\$ ./send\_signal

Error!

[baranyuk\_dmitro@vpsj3IeQ ~]\$ ./get\_signal

pidd=13373

^C

[baranyuk\_dmitro@vpsj3IeQ ~]\$ nano send\_signal.c

[baranyuk\_dmitro@vpsj3IeQ ~]\$ gcc send\_signal.c -o send\_signal

[baranyuk\_dmitro@vpsj3IeQ ~]\$ ./send\_signal

Error!

#### Завдання 4 Створення процесу-сироти

Створіть С-програму, в якій процес-батько несподівано завершується раніше процесу-нащадку. Процес-батько повинен очікувати завершення  $n+1$  секунд. Процес-нащадок повинен в циклі  $(2*n+1)$  раз із затримкою в 1 секунду виводити повідомлення, наприклад, «Parent of Ivanov», за шаблоном як в попередньому завданні, і додатково виводити PPID процесу-батька.

Значення  $n$  – номер команди студента + номер студента в команді.

Перевірте роботу програми, вивчіть вміст таблиці процесів і зробіть відповідні висновки.

```
baranyuk_dmitro@vpsj3IeQ:~
GNU nano 2.3.1 File: sirota.c

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

int main(void){
    int i;
    pid_t pid= fork();
    if(pid!=0){
        printf("I am parent! pid=%d, child pid=%d\n",getpid(),pid);
        sleep(2);
        _exit(0);
    }
    else{
        for(i=0;i<5;i++){
            fprintf(stderr,"Iam child with pid=%d. My parent pid = %d\n",getpid(),getppid());
        }
        sleep(1);
    }
    return 0;
}

[baranyuk_dmitro@vpsj3IeQ ~]$ nano sirota.c
[baranyuk_dmitro@vpsj3IeQ ~]$ gcc sirota.c -o sirota
[baranyuk_dmitro@vpsj3IeQ ~]$ ./sirota
I am parent! pid=6212, child pid=6213
Iam child with pid=6213. My parent pid = 6212
Iam child with pid=6213. My parent pid = 6212
Iam child with pid=6213. My parent pid = 1
[baranyuk_dmitro@vpsj3IeQ ~]$ Iam child with pid=6213. My parent pid = 1
Iam child with pid=6213. My parent pid = 1
```

Висновок: під час лабораторної роботи №8 ми отримали практичні навички з програмування керування процесами в ОС Unix. Найскладнішим завданням було завдання №3.