

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

Лабораторна робота № 8  
з дисципліни «Операційні  
системи»

Тема: «Програмування керуванням процесами в ОС Unix»

**Виконала:**

Студентка групи AI-202  
Гребенік Анжеліка Олександрівна

Одеса-2021

**Мета роботи:** отримання навичок в управлінні процесами в ОС Unix на рівні мови програмування C.

## **2. Завдання**

### Завдання 1 Перегляд інформації про процес

Створіть C-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 помилок.

Після компіляції запустіть програму.

Додатково запустіть програму в конвеєрі, наприклад:

```
./info | ./info
```

Порівняйте значення групи процесів.

### Завдання 2 Стандартне створення процесу

Створіть C-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.

### Завдання 3 Обмін сигналами між процесами

3.1 Створіть C-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 C-програму.

3.2 Створіть C-програму, яка надсилає сигнал `SIGUSR2` процесу, запущеному в попередньому пункту завдання.

Запустіть створену C-програму та проаналізуйте повідомлення, які виводить перша програма.

Завершіть процес, запущеному в попередньому пункту завдання.

### Завдання 4 Створення процесу-сироти

Створіть C-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.

### **Хід роботи:**

#### Завдання 1 Перегляд інформації про процес

Створіть C-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 помилок.

Після компіляції запусіть програму.

Додатково запусіть програму в конвеєрі, наприклад:

`./info | ./info`

Порівняйте значення групи процесів.

```
mc [grebenik_anzhelika@vpsj3IeQ.s-host.com.ua]:~
info.c [-----] 4 L:[ 1+11 12/ 13] *(341 / 352b) 0114 0x072 [*][X]
#include <stdio.h>
#include <unistd.h>

int main (void) {
    fprintf(stderr, "I am process, gpid=%d\n", getpgrp());
    fprintf(stderr, "pid=%d\n", getpid());
    fprintf(stderr, "ppid=%d\n", getppid());
    fprintf(stderr, "uid=%d\n", getuid());
    fprintf(stderr, "gid=%d\n", getgid());
    fprintf(stderr, "sid=%d\n", getsid(0));
    sleep(5);
    return 0;
}
```

```
grebenik_anzhelika@vpsj3IeQ:~
uid=54347
gid=54353
sid=25524
I am process, gpid=27634
pid=27634
ppid=25524
uid=54347
gid=54353
sid=25524
[grebenik_anzhelika@vpsj3IeQ ~]$ gcc info.c -o info
[grebenik_anzhelika@vpsj3IeQ ~]$ ./info | ./info
I am process, gpid=28114
pid=28115
ppid=25524
uid=54347
gid=54353
sid=25524
I am process, gpid=28114
pid=28114
ppid=25524
uid=54347
gid=54353
sid=25524
[grebenik_anzhelika@vpsj3IeQ ~]$
```

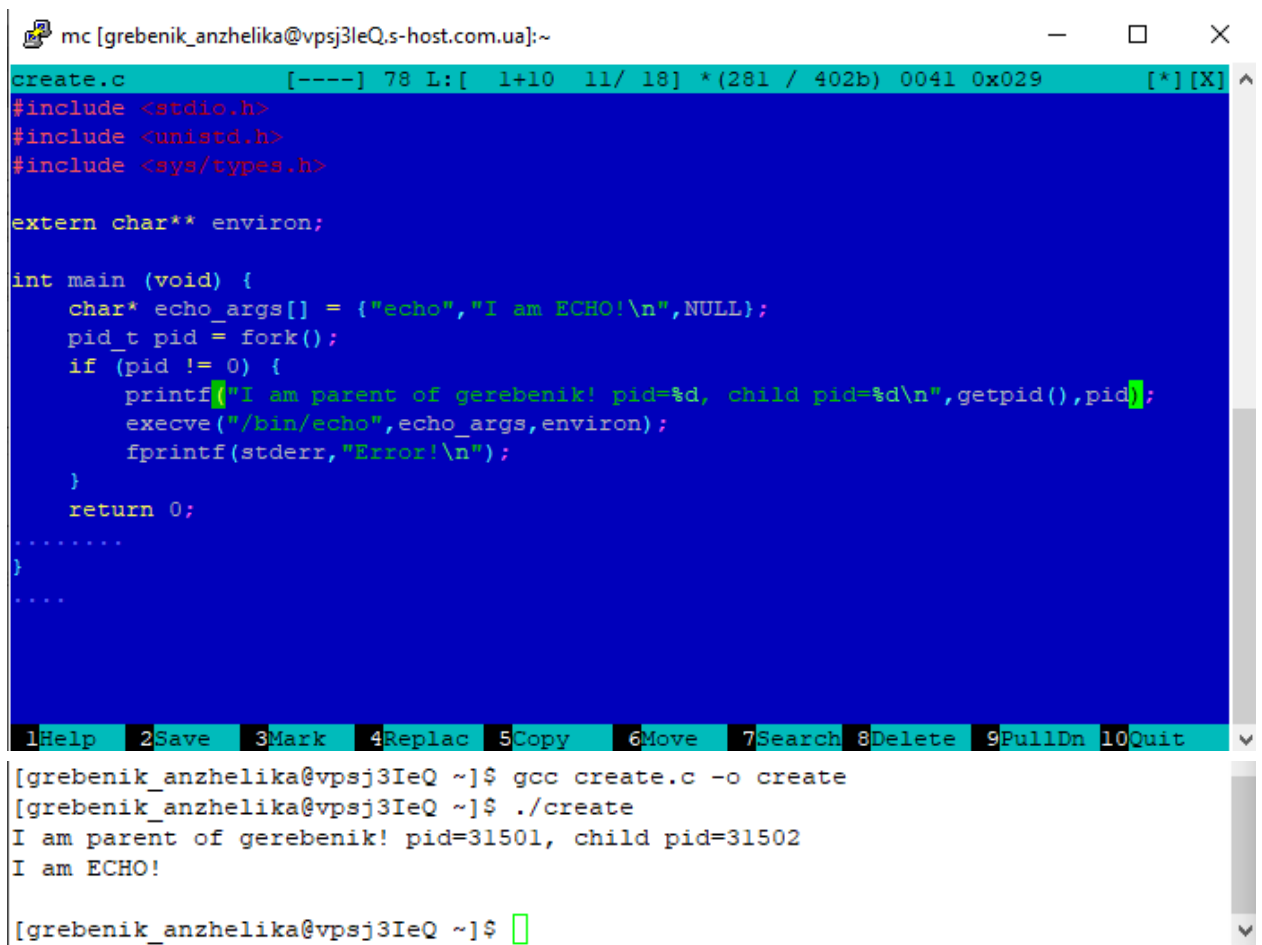
*Командою gcc ми компілюємо програму, та потім її запускаємо (./info | ./info) використовуючи конвеєр команд.*

*Отримуємо різні pid процесів, але все інше однакове.*

## 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.



```
mc [grebenik_anzhelika@vpsj3IeQ.s-host.com.ua]:~
create.c [----] 78 L:[ 1+10 11/ 18] *(281 / 402b) 0041 0x029 [*] [X] ^
#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 of gerebenik! pid=%d, child pid=%d\n", getpid(), pid);
        execve("/bin/echo", echo_args, environ);
        fprintf(stderr, "Error!\n");
    }
    return 0;
}
.....
[grebenik_anzhelika@vpsj3IeQ ~]$ gcc create.c -o create
[grebenik_anzhelika@vpsj3IeQ ~]$ ./create
I am parent of gerebenik! pid=31501, child pid=31502
I am ECHO!
[grebenik_anzhelika@vpsj3IeQ ~]$
```

### Завдання 3 Обмін сигналами між процесами

3.1 Створіть C-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 C-програму.

3.2 Створіть C-програму, яка надсилає сигнал SIGUSR2 процесу, запущеному в попередньому пункту завдання.

Запустіть створену C-програму та проаналізуйте повідомлення, які виводить перша програма.

```
mc [grebenik_anzhelika@vpsj3IeQ.s-host.com.ua]:~
get_signal.c [----] 1 L:[ 1+16 17/ 17] *(382 / 382b) <EOF> [*][X] ^
#include <signal.h>
#include <stdio.h>
#include <unistd.h>

static void sig_usr(int signo) {
    if(signo == SIGUSR2)
        fprintf(stderr, "Process of Grebenik got signal %d\n",SIGUSR2);
}

int main(void) {
    printf("pid=%d\n",getpid());
    if(signal(SIGUSR2,sig_usr) == SIG_ERR)
        fprintf(stderr, "Error!\n");
    for( ; ; )
        pause();
    return 0;
}

1Help 2Save 3Mark 4Replac 5Copy 6Move 7Search 8Delete 9PullDn 10Quit
```

Створюємо програму, компілюємо та запускаємо. На екран виводиться `pid`, який ми потім вводимо до програми `send_signal`.

```
[grebenik_anzhelika@vpsj3IeQ ~]$ gcc get_signal.c -o get_signal
[grebenik_anzhelika@vpsj3IeQ ~]$ ./get_signal
pid=8700
```

Не закриваючи `get_signal` відкриваємо інше вікно, де запускаємо програму `send_signal`

```
mc [grebenik_anzhelika@vpsj3IeQ.s-host.com.ua]:~
send_signal.c [----] 16 L:[ 1+ 3 4/ 12] *(56 / 245b) 0059 0x03B [*][X] ^
#include <signal.h>
#include <stdio.h>

pid_t pid = 8700;

int main(void) {
    if(!kill(pid,SIGUSR2))
        printf("Process of Grebenik send signal\n");
    else
        fprintf(stderr, "Error!\n");
    return 0;
}

1Help 2Save 3Mark 4Replac 5Copy 6Move 7Search 8Delete 9PullDn 10Quit

[grebenik_anzhelika@vpsj3IeQ ~]$ gcc send_signal.c -o send_signal
[grebenik_anzhelika@vpsj3IeQ ~]$ ./send_signal
Process of Grebenik send signal
[grebenik_anzhelika@vpsj3IeQ ~]$
```

в свою чергу в першому терміналі ми отримуємо номер сигналу:

```
[grebenik_anzhelika@vpsj3IeQ ~]$ gcc get_signal.c -o get_signal
[grebenik_anzhelika@vpsj3IeQ ~]$ ./get_signal
pid=8700
Process of Grebenik got signal 12
```

#### Завдання 4 Створення процесу-сироти

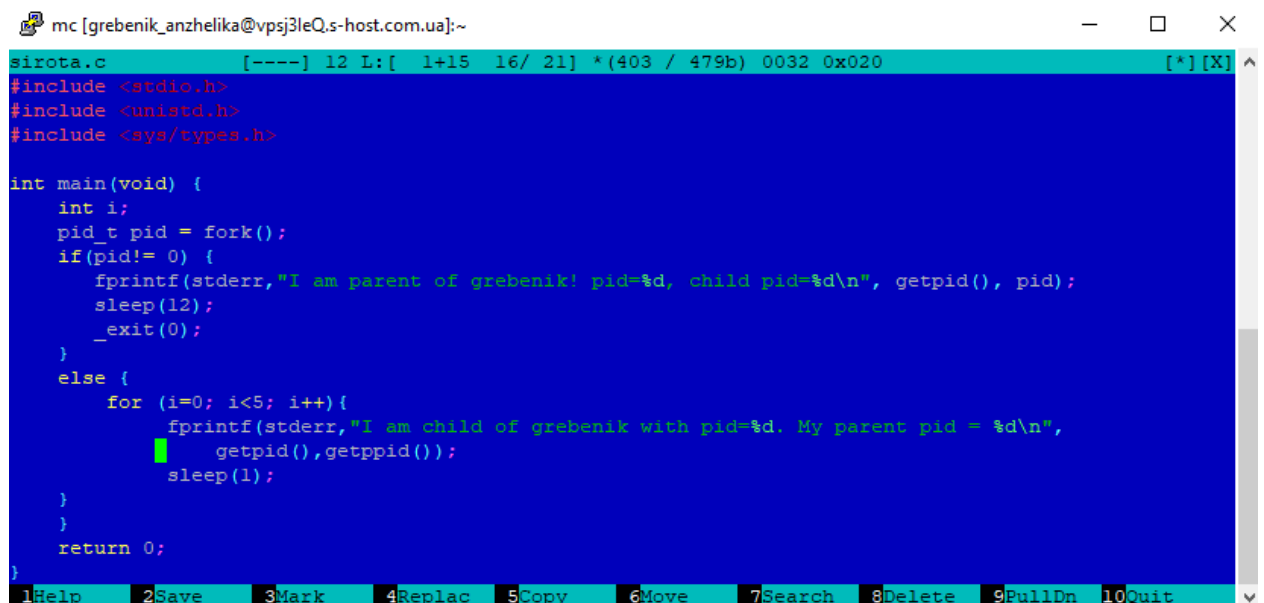
Створіть С-програму, в якій процес-батько несподівано завершується раніше процесу-нащадку. Процес-батько повинен очікувати завершення  $n+1$  секунд. Процес-нащадок повинен в циклі  $(2*n+1)$  раз із затримкою в 1 секунду виводити повідомлення, наприклад, «Parent of Ivanov», за шаблоном як в попередньому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

$$2*(4+1+1)=12$$

The screenshot shows a code editor window titled 'mc [grebenik\_anzhelika@vpsj3IeQ.s-host.com.ua]:~'. The editor contains the source code for 'sirota.c'. The code includes standard headers and a main function. The parent process forks a child process, prints its own PID and the child's PID, sleeps for 12 seconds, and then exits. The child process enters a loop that runs 5 times (i=0 to i=4), printing its own PID, its parent's PID (PPID), and sleeping for 1 second each time. The editor has a status bar at the bottom with menu items: 1Help, 2Save, 3Mark, 4Replac, 5Copy, 6Move, 7Search, 8Delete, 9PullDn, 10Quit.

```
[grebenik_anzhelika@vpsj3IeQ ~]$ gcc sirota.c -o sirota
[grebenik_anzhelika@vpsj3IeQ ~]$ ./sirota
I am parent of grebenik! pid=10608, child pid=10609
I am child of grebenik with pid=10609. My parent pid = 10608
I am child of grebenik with pid=10609. My parent pid = 10608
I am child of grebenik with pid=10609. My parent pid = 10608
I am child of grebenik with pid=10609. My parent pid = 10608
I am child of grebenik with pid=10609. My parent pid = 10608
[grebenik_anzhelika@vpsj3IeQ ~]$
```

**Висновок:** під час лабораторної роботи ми отримали навички в управлінні процесами в ОС Unix на рівні мови програмування C.