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 АІ-203
Грищенко О.Р.

Одеса 2021

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

**Завдання до 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.

*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.

*Завдання 3 Обмін сигналами між процесами*

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.

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.

*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.

*Завдання 5 Створення процесу-зомбі*

Створіть С-програму, в якій процес-нащадок несподівано завершується раніше процесу-батька, перетворюється на зомбі, виводячи в результаті повідомлення, наприклад,
«I am Zombie-process of Ivanov», за шаблоном як в попередньому завданні.

Запустіть програму у фоновому режимі, а в окремому терміналі вивчіть вміст таблиці процесів і зробіть відповідні висновки.

*Завдання 6 Попередження створення процесу-зомбі*

Створіть С-програму, в якій процес-нащадок завершується раніше процесу-батька, але ця подія контролюється процесом-батьком.

Процес-нащадок повинен виводити повідомлення, наприклад, «Child of Ivanov is finished», за шаблоном як в попередньому завданні.

Процес-батько повинен очікувати (3*n) секунд.

Значення n – номер команди студента + номер студента в команді.
Запустіть програму у фоновому режимі, а в окремому терміналі вивчіть вміст таблиці процесів і зробіть відповідні висновки.

**Результати виконання завдань:**

**1.** Перегляд інформації про процес:

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

int main(void){
    printf("My pid = %d\n", getpid());
    printf("My ppid = %d\n", getppid());
    printf("My uid = %d\n", getuid());
    printf("My gid = %d\n", getgid());
    printf("My pgrp = %d\n", getpgrp());
    printf("My sid = %d\n", getsid(0));
....
    return 0;
}
```

```
[grishenko_oleksandra@vpsj3IeQ ~]$ gcc ex_1.c -o ex_1
[grishenko_oleksandra@vpsj3IeQ ~]$ ./ex_1
My pid = 9201
My ppid = 8130
My uid = 54366
My gid = 54372
My pgrp = 9201
My sid = 8130
```

**2.** Стандартне створення процесу:

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

extern char **environ;

int main(void){
    char *echo_args[] = {"echo", "Child of Grishenko\n", NULL};
    pid_t pid = fork();
    if (pid == 0){
------>execve("/bin/echo", echo_args, environ);
------>fprintf(stderr, "Error!\n");
------>return 1;
------>}
    else
------>printf("Parent of Grishenko\n");
    return 0;
}
```

```
[grishenko_oleksandra@vpsj3IeQ ~]$ gcc ex_2.c -o ex_2
[grishenko_oleksandra@vpsj3IeQ ~]$ ./ex_2
Parent of Grishenko
[grishenko_oleksandra@vpsj3IeQ ~]$ Child of Grishenko
```

**3.** Обмін сигналами між процесами:

```
ex_32.c             [----] 17 L:[  1+
#include <stdio.h>
#include <signal.h>

pid_t pid = 17859;

int main(void){
    if (!kill(pid, SIGUSR2))
------>printf("Send signal\n");
    else
------>fprintf(stderr, "Error!\n");
    return 0;
}
```

```
ex_3.c              [----]  1 L:[  1+13   14/ 14] *(26
#include <stdio.h>
#include <signal.h>

static void sig_usr(int signo){
    if (signo == SIGUSR2)
------>printf("Process of Grishenko got signal\n");
}

int main(void){
    if(signal(SIGUSR2, sig_usr) == SIG_ERR)
------>fprintf(stderr, "Error!\n");
    for( ; ; )
------>pause();
}
```

```
[grishenko_oleksandra@vpsj3IeQ ~]$ gcc ex_32.c -o ex_32
[grishenko_oleksandra@vpsj3IeQ ~]$ ./ex_32
Send signal
[grishenko_oleksandra@vpsj3IeQ ~]$ ./ex_3
Process of Grishenko got signal
```

**4.** Створення процесу-сироти:

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

int main(void){
    int i;
    pid_t pid = fork();
    if(pid != 0){
------>sleep(5);
------>_exit(0);
    }
    else{
------>for(i=0; i<9; i++){
------>    printf("Parent of Grishenko; ppid = %d\n", getppid());
------>    sleep(1);
------>}
    }
    return 0;
}
```

```
[grishenko_oleksandra@vpsj3IeQ ~]$ gcc ex_4.c -o ex_4
[grishenko_oleksandra@vpsj3IeQ ~]$ ./ex_4
Parent of Grishenko; ppid = 17630
Parent of Grishenko; ppid = 17630
Parent of Grishenko; ppid = 17630
Parent of Grishenko; ppid = 17630
Parent of Grishenko; ppid = 17630
[grishenko_oleksandra@vpsj3IeQ ~]$ Parent of Grishenko; ppid = 1
Parent of Grishenko; ppid = 1
Parent of Grishenko; ppid = 1
Parent of Grishenko; ppid = 1
```