

Міністерство освіти і науки України
Державний університет «Одеська політехніка»
Інститут комп'ютерних систем
Кафедра інформаційних систем

Лабораторна робота №8
з дисципліни «Операційні Системи»

Тема: «Програмування керуванням процесами в ОС Unix»

Виконав:
ст. гр. AI-204
Колесник К. В.

Перевірив:
Блажко О. А.

Одеса – 2021

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

Завдання для виконання:

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.

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

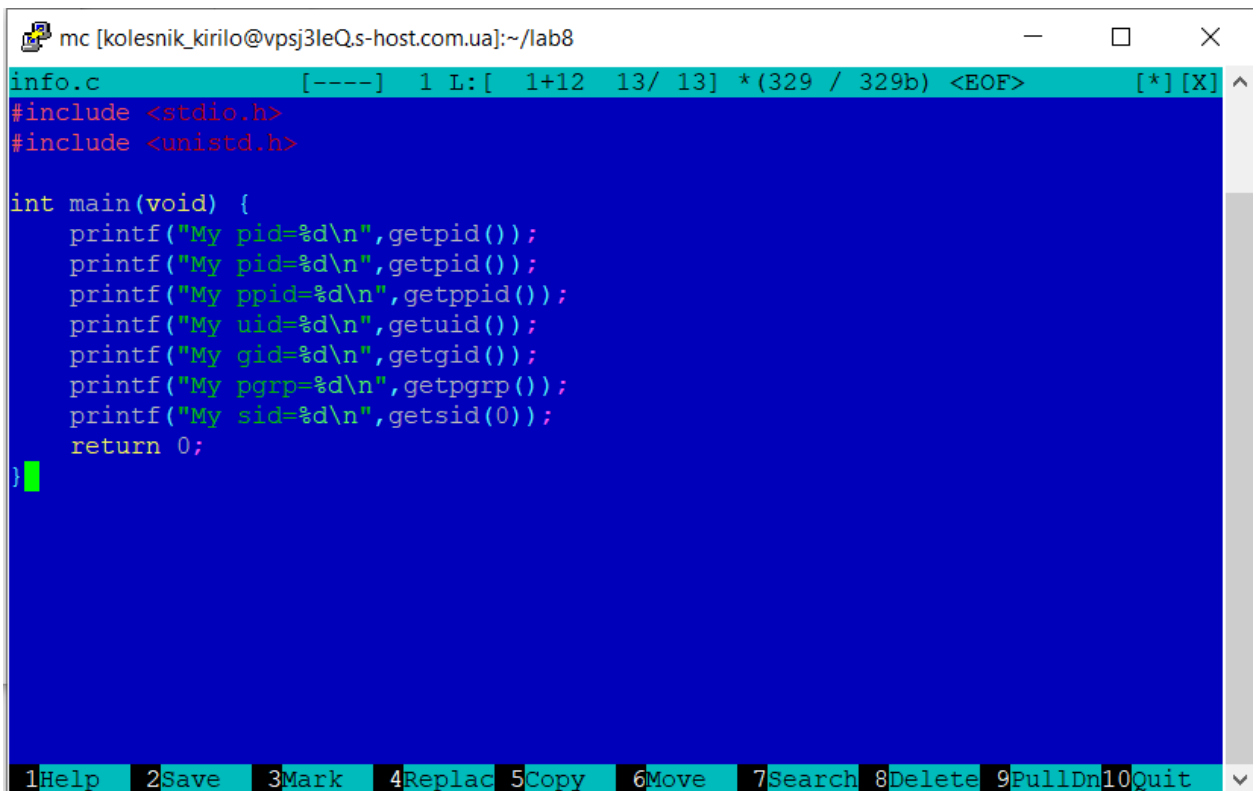
Створіть C-програму, в якій процес-батько несподівано завершується раніше процесу-нащадку. Процес-батько повинен очікувати завершення $n+1$ секунд. Процес-нащадок повинен в циклі $(2*n+1)$ раз із затримкою в 1 секунду виводити повідомлення, наприклад, «Parent of Ivanov», за шаблоном як в попередньому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

Створено C-програму, яка виводить на екран таку інформацію:

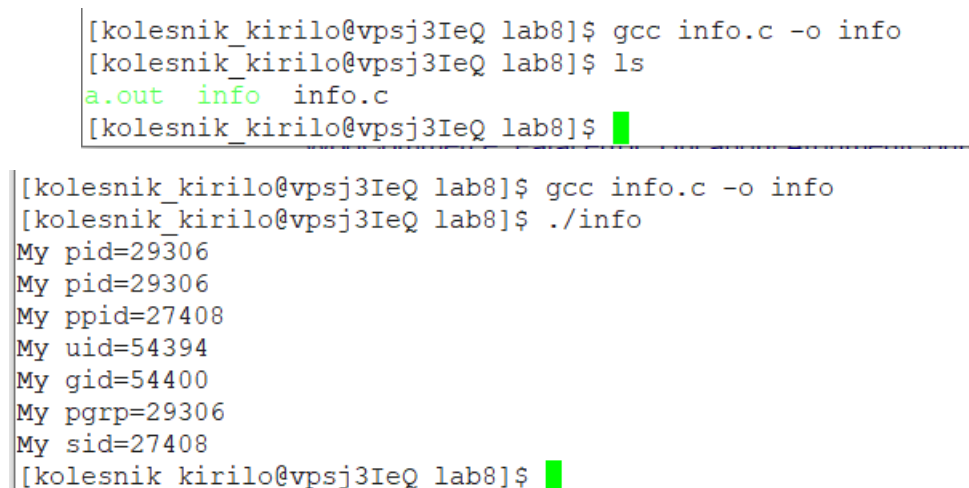
- ідентифікатор групи процесів лідера сесії;
- ідентифікатор групи процесів, до якої належить процес;
- ідентифікатор процесу, що викликав цю функцію;
- ідентифікатор батьківського процесу;
- ідентифікатор користувача процесу, який викликав цю функцію;
- ідентифікатор групи користувача процесу, який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



```
mc [kolesnik_kirilo@vpsj3IeQ.s-host.com.ua]:~/lab8
info.c [----] 1 L: [ 1+12 13/ 13] *(329 / 329b) <EOF> [*] [X] ^
#include <stdio.h>
#include <unistd.h>

int main(void) {
    printf("My pid=%d\n",getpid());
    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;
}
```

Рис. 1.1 – код програми



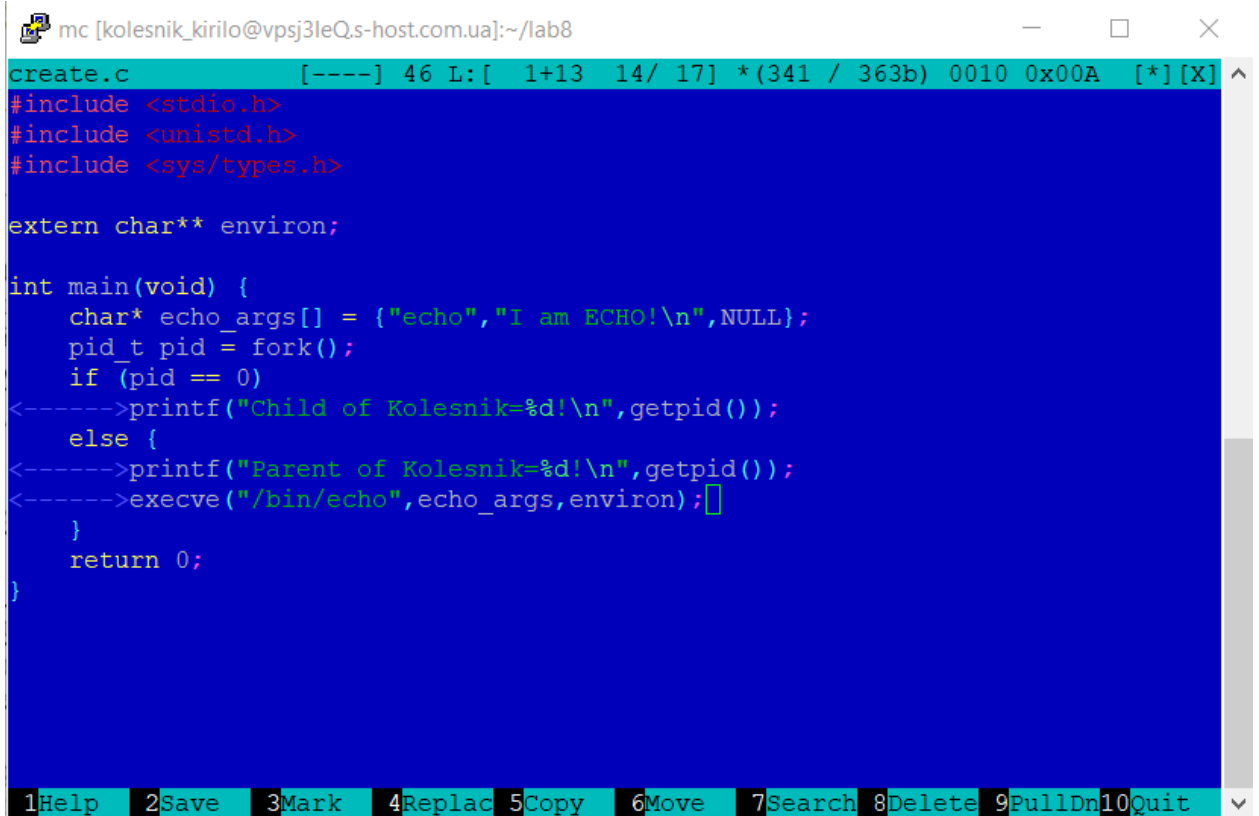
```
[kolesnik_kirilo@vpsj3IeQ lab8]$ gcc info.c -o info
[kolesnik_kirilo@vpsj3IeQ lab8]$ ls
a.out  info  info.c
[kolesnik_kirilo@vpsj3IeQ lab8]$

[kolesnik_kirilo@vpsj3IeQ lab8]$ gcc info.c -o info
[kolesnik_kirilo@vpsj3IeQ lab8]$ ./info
My pid=29306
My pid=29306
My ppid=27408
My uid=54394
My gid=54400
My pgrp=29306
My sid=27408
[kolesnik_kirilo@vpsj3IeQ lab8]$
```

Рис. 1.2 (а) – результат компіляції, (б) – результат виконання програми

Завдання 2 Стандартне створення процесу

Створено С-програму, яка створює процес-нащадок, породжуючи процес та замінюючи образ процесу. У програмі процес-батько видає повідомлення типу «Parent of Kolesnik», а процес-нащадок видає повідомлення типу «Child of Kolesnik» через виклик команди echo.



```
mc [kolesnik_kirilo@vpsj3IeQ.s-host.com.ua]:~/lab8
create.c [----] 46 L: [ 1+13 14/ 17] *(341 / 363b) 0010 0x00A [*] [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("Child of Kolesnik=%d!\n", getpid());
    else {
<----->printf("Parent of Kolesnik=%d!\n", getpid());
<----->execve("/bin/echo", echo_args, environ);[]
    }
    return 0;
}
```

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

Рис. 2.1 – код програми

```
[kolesnik_kirilo@vpsj3IeQ lab8]$ gcc create.c -o create
[kolesnik_kirilo@vpsj3IeQ lab8]$ ./create
Parent of Kolesnik=31245!
Child of Kolesnik=31246!
I am ECHO!

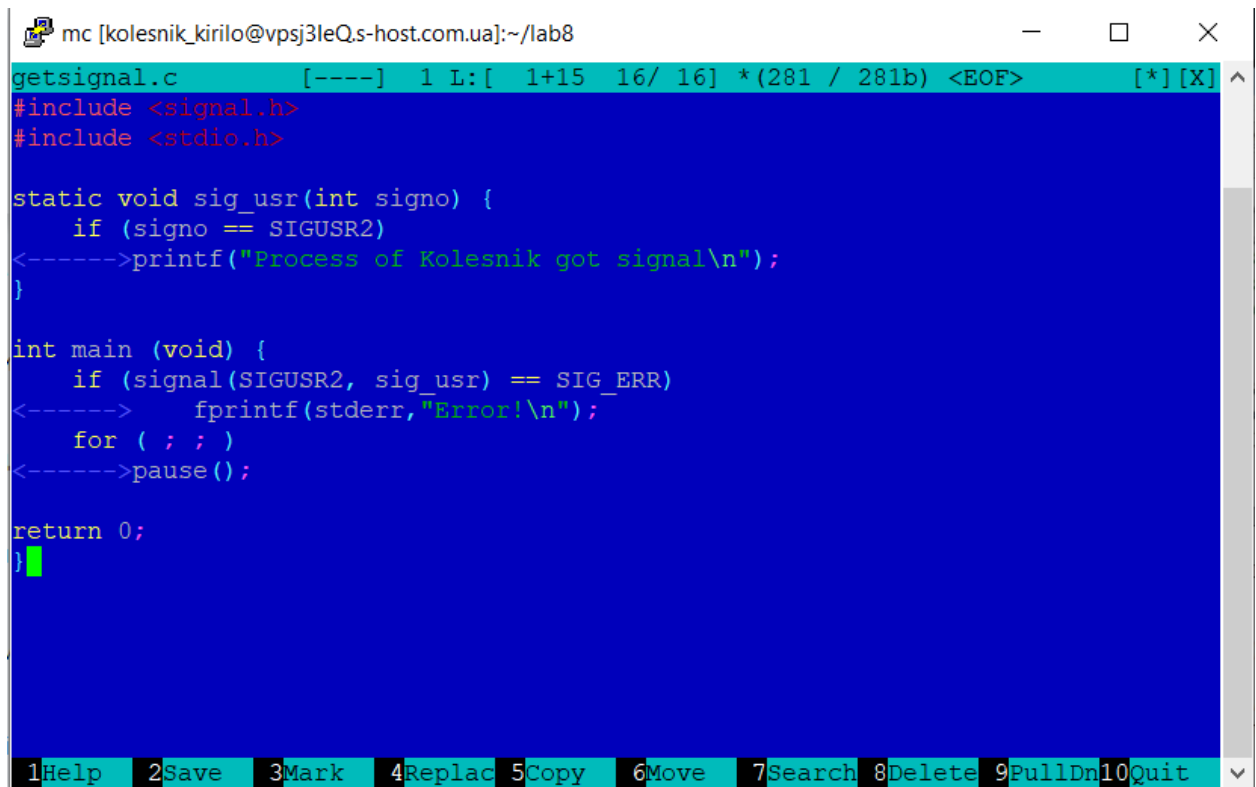
[kolesnik_kirilo@vpsj3IeQ lab8]$
```

Рис. 2.2 – результат виконання програми

Завдання 3 Обмін сигналами між процесами

3.1 Створено С-програму, в якій процес очікує отримання сигналу SIGUSR2 та виводить повідомлення типу «Process of Kolesnik got signal» після отримання сигналу.

Запущено створену С-програму.



```
mc [kolesnik_kirilo@vpsj3IeQ.s-host.com.ua]:~/lab8
getsignal.c [----] 1 L:[ 1+15 16/ 16] *(281 / 281b) <EOF> [*] [X] ^
#include <signal.h>
#include <stdio.h>

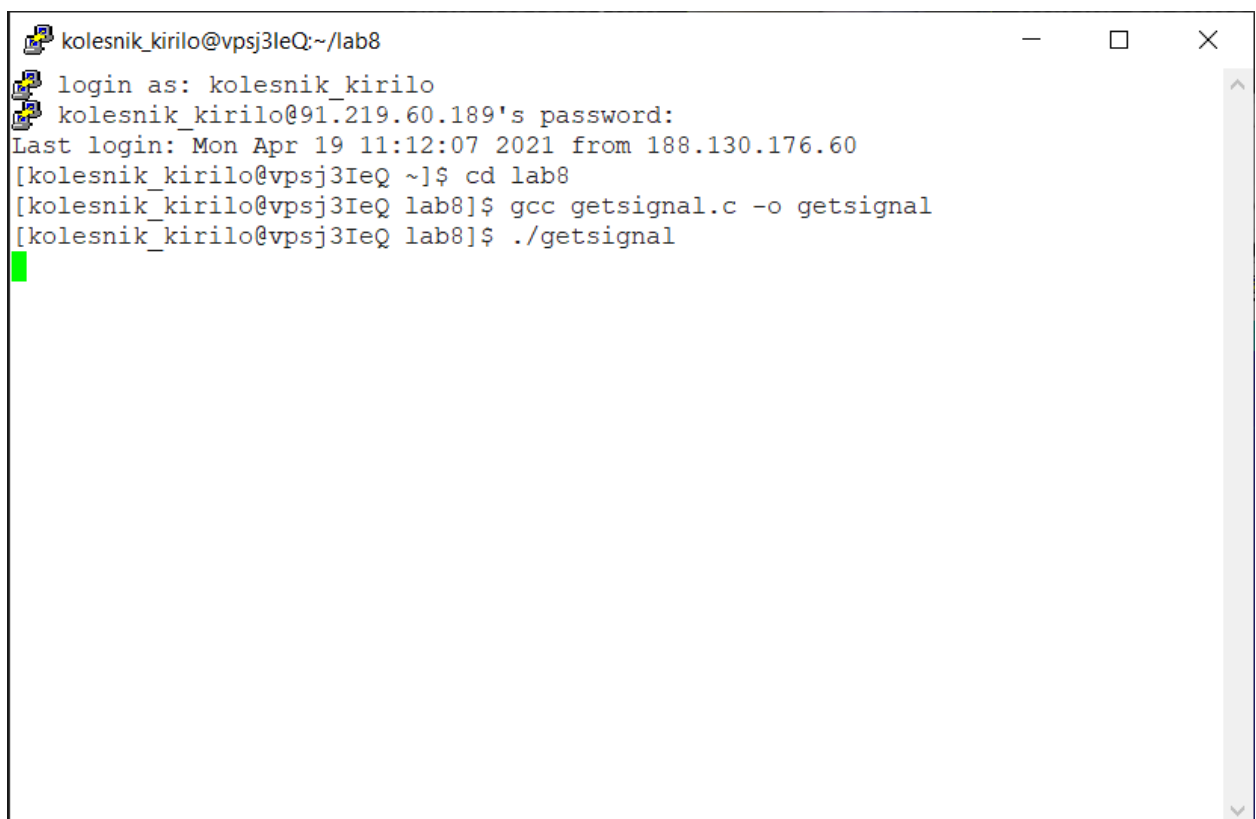
static void sig_usr(int signo) {
    if (signo == SIGUSR2)
<----->printf("Process of Kolesnik got signal\n");
}

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

return 0;
}

1Help 2Save 3Mark 4Replac 5Copy 6Move 7Search 8Delete 9PullDn10Quit v
```

Рис. 3.1.1 – код програми

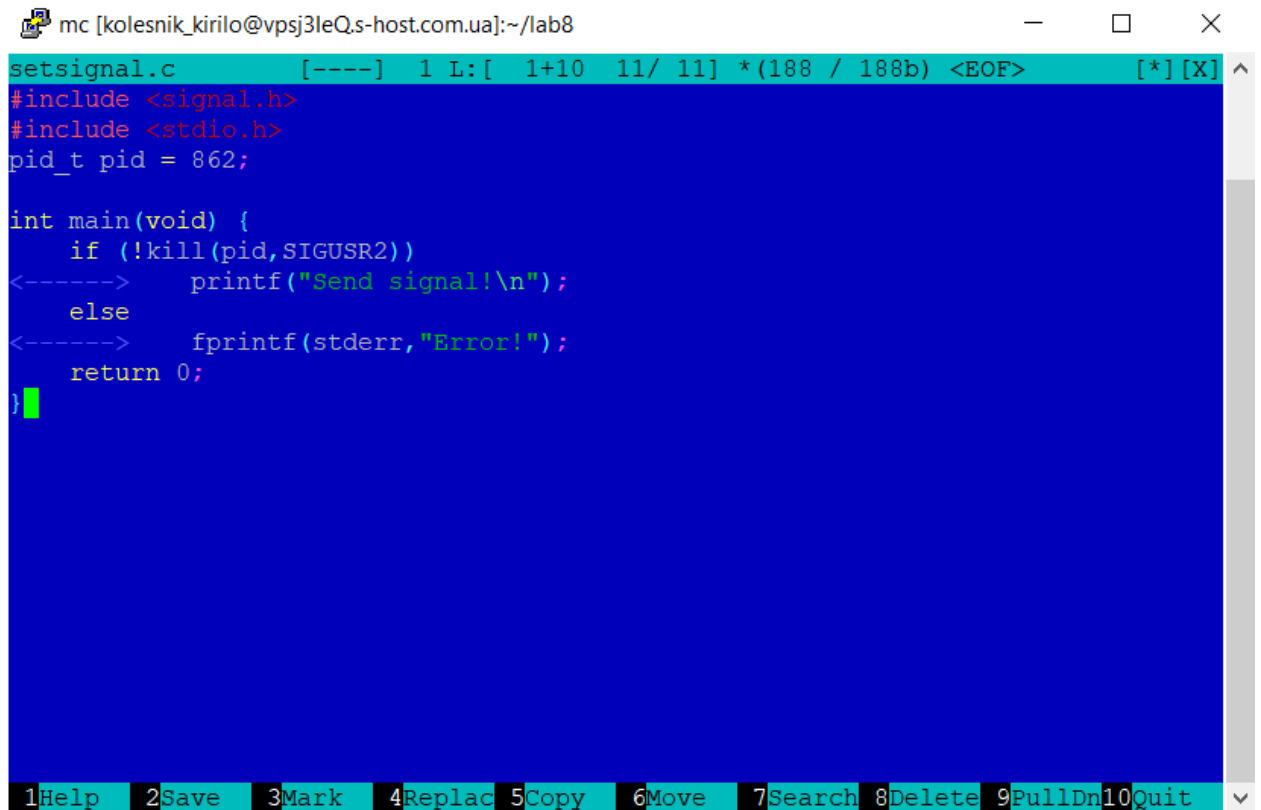


```
kolesnik_kirilo@vpsj3IeQ:~/lab8
login as: kolesnik_kirilo
kolesnik_kirilo@91.219.60.189's password:
Last login: Mon Apr 19 11:12:07 2021 from 188.130.176.60
[kolesnik_kirilo@vpsj3IeQ ~]$ cd lab8
[kolesnik_kirilo@vpsj3IeQ lab8]$ gcc getsignal.c -o getsignal
[kolesnik_kirilo@vpsj3IeQ lab8]$ ./getsignal
```

Рис. 3.1.2 – результат виконання програми

3.2 Створено С-програму, яка надсилає сигнал SIGUSR2 процесу, запущеному в попередньому пункту завдання.

Запущено створену С-програму. Бачимо, що перша програма отримала сигнал та показало відповідне повідомлення.



```
setsignal.c [----] 1 L:[ 1+10 11/ 11] *(188 / 188b) <EOF> [*] [X] ^
#include <signal.h>
#include <stdio.h>
pid_t pid = 862;

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

Рис. 3.2.1 – код програми

```
[kolesnik_kirilo@vpsj3IeQ lab8]$ gcc setsignal.c -o setsignal
[kolesnik_kirilo@vpsj3IeQ lab8]$ ./setsignal
Send signal!
[kolesnik_kirilo@vpsj3IeQ lab8]$
```

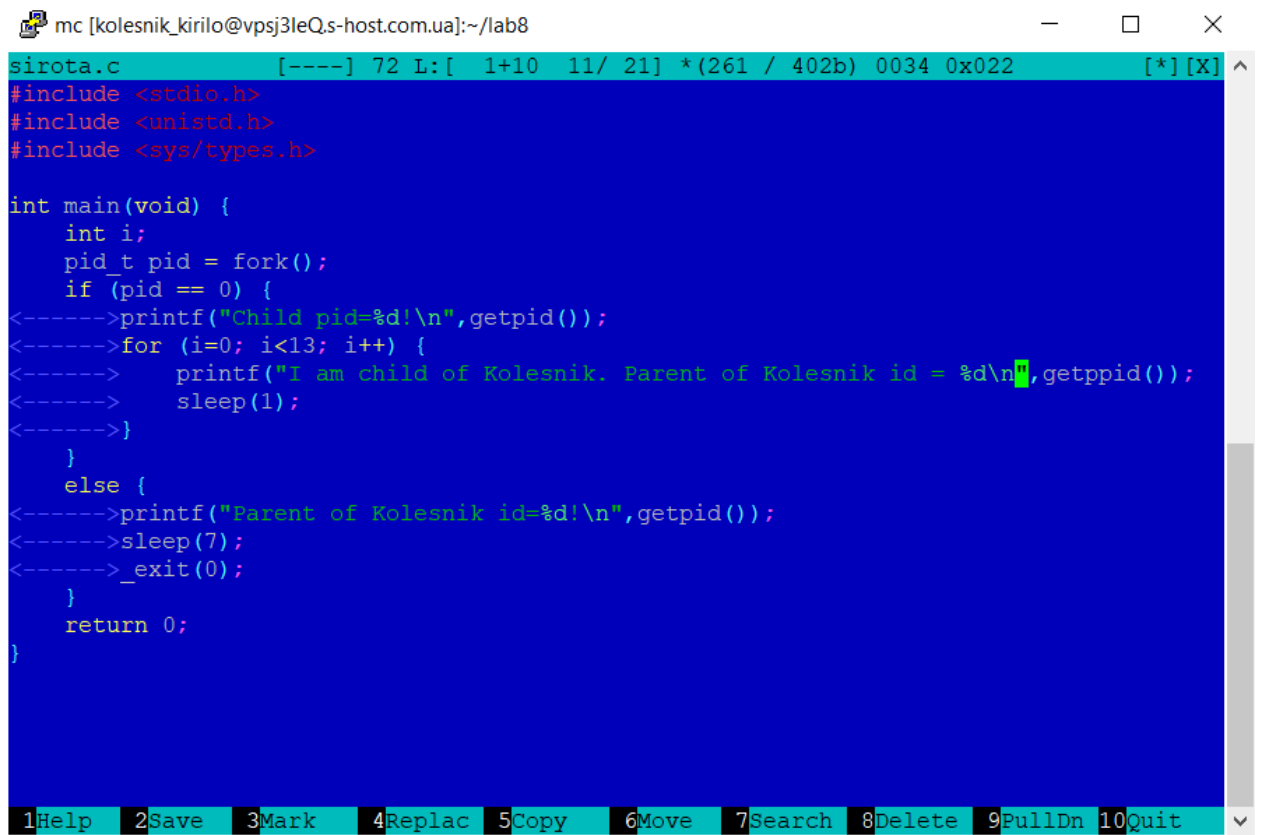
Рис. 3.2.2 – результат виконання програми

```
[kolesnik_kirilo@vpsj3IeQ lab8]$ ./getsignal
Process of Kolesnik got signal
```

Рис. 3.2.3 – повідомлення у терміналі першої програми

Завдання 4 Створення процесу-сироти

Створено С-програму, в якій процес-батько несподівано завершується раніше процесу-нащадку. Процес-батько повинен очікувати завершення $n+1$ ($6+1 = 7$) секунд. Процес-нащадок повинен в циклі $(2*n+1)$ ($2*6+1 = 13$) раз із затримкою в 1 секунду виводити повідомлення, «Parent of Kolesnik», за шаблоном як в попередньому завданні, і додатково виводити PPID процесу-батька. Значення n – номер команди студента (команда №6) + номер студента в команді (студент №1). Бачимо, що процес-сирота був успішно створений.



```
mc [kolesnik_kirilo@vpsj3leQ.s-host.com.ua]:~/lab8
sirota.c [----] 72 L: [ 1+10 11/ 21] *(261 / 402b) 0034 0x022 [*] [X] ^
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>

int main(void) {
    int i;
    pid_t pid = fork();
    if (pid == 0) {
<----->printf("Child pid=%d!\n",getpid());
<----->for (i=0; i<13; i++) {
<----->    printf("I am child of Kolesnik. Parent of Kolesnik id = %d\n",getppid());
<----->    sleep(1);
<----->}
    }
    else {
<----->printf("Parent of Kolesnik id=%d!\n",getpid());
<----->sleep(7);
<----->_exit(0);
    }
    return 0;
}
```

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

Рис. 4.1 – код програми


```
[kolesnik_kirilo@vpsj3IeQ lab8]$ gcc sirota.c -o sirota
[kolesnik_kirilo@vpsj3IeQ lab8]$ ./sirota
Parent of Kolesnik id=9966!
Child pid=9967!
I am child of Kolesnik. Parent of Kolesnik id = 9966
I am child of Kolesnik. Parent of Kolesnik id = 9966
I am child of Kolesnik. Parent of Kolesnik id = 9966
I am child of Kolesnik. Parent of Kolesnik id = 9966
I am child of Kolesnik. Parent of Kolesnik id = 9966
I am child of Kolesnik. Parent of Kolesnik id = 9966
I am child of Kolesnik. Parent of Kolesnik id = 9966
I am child of Kolesnik. Parent of Kolesnik id = 9966
[kolesnik_kirilo@vpsj3IeQ lab8]$ I am child of Kolesnik. Parent of Kolesnik id = 1
I am child of Kolesnik. Parent of Kolesnik id = 1
I am child of Kolesnik. Parent of Kolesnik id = 1
I am child of Kolesnik. Parent of Kolesnik id = 1
I am child of Kolesnik. Parent of Kolesnik id = 1
I am child of Kolesnik. Parent of Kolesnik id = 1
```

Рис. 4.2 – результат виконання програми

Висновок: Під час виконання цієї лабораторної роботи, було на практиці розглянуто принципи взаємодії типу “клієнт – сервер” із детальним дослідженням функцій та можливостей для програмування алгоритмів управління процесами у Unix-подібних ОС інтерфейсу командного рядку. Виявлено, що користувач отримує можливість здійснювати всі попередньо розглянуті операції керування процесами автоматично, за допомогою програм, створених на мові C. Таким чином спектр можливостей керування процесами у CPU розширюється та стає більш зручним.

Складностей під час виконання роботи не виникало.