

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ
В.Н.КАРАЗІНА
КАФЕДРА ШТУЧНОГО ІНТЕЛЕКТУ ТА ПРОГРАМНОГО
ЗАБЕЗПЕЧЕННЯ

ЛАБОРАТОРНА РОБОТА № 4
З ДИСЦИПЛІНИ «ОПЕРАЦІЙНІ СИСТЕМИ»

ТЕМА «ПРОЦЕСИ»

Виконав студент 3курсу, групи
КС31

спеціальності

122 – Комп'ютерні науки

Касьяненко Микита Михайлович

Прийняв:

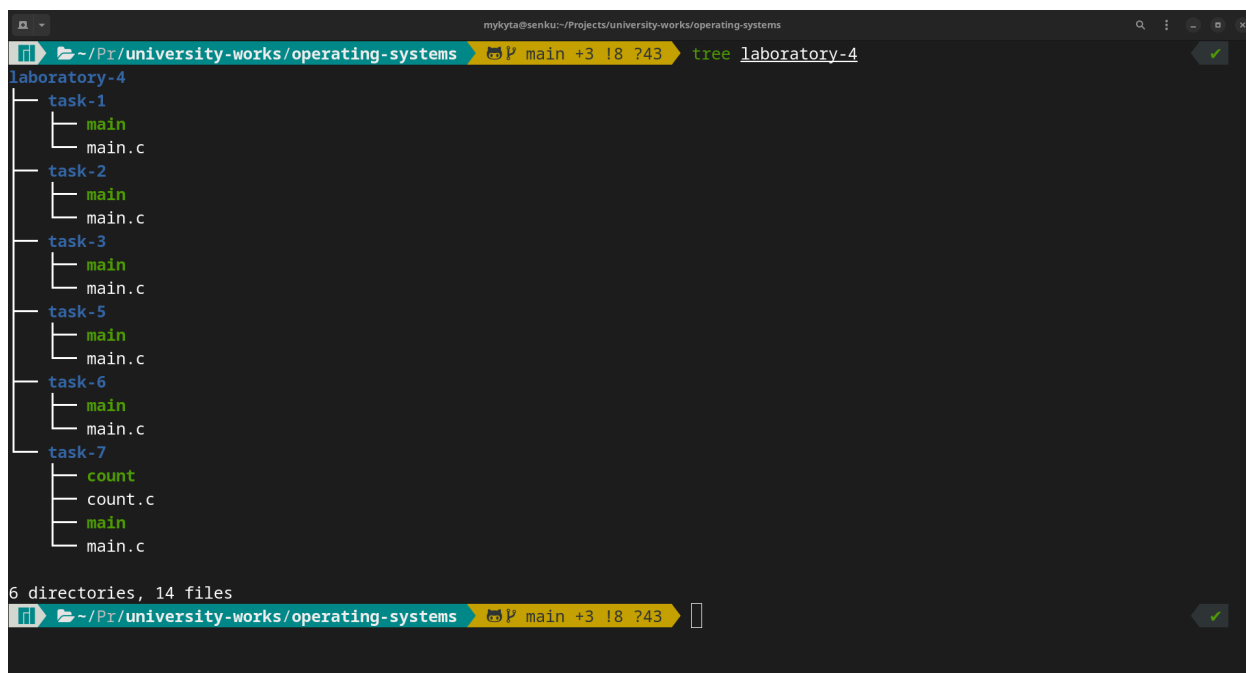
доцен кафедри шт. ін.

і прогр. забезп. к.н.т.

О.Є. Споров ____

Харків 2022

Файли проекту лабораторної роботи



```
mykyta@senku:~/Projects/university-works/operating-systems
tree laboratory-4
laboratory-4
├── task-1
│   ├── main
│   └── main.c
├── task-2
│   ├── main
│   └── main.c
├── task-3
│   ├── main
│   └── main.c
├── task-5
│   ├── main
│   └── main.c
├── task-6
│   ├── main
│   └── main.c
└── task-7
    ├── count
    ├── count.c
    ├── main
    └── main.c

6 directories, 14 files
```

Завдання 1

Написати функцію, яка виводить в стандартний потік виведення інформацію про процес. Додаткове завдання: передбачити аргумент, за допомогою якого можна управляти тією інформацією, що виводиться в стандартний потік виведення (мається на увазі — вказувати лише ті ідентифікатори, що потрібні).

Відповідь

```

mykyta@senku:~/Projects/university-works/operating-systems/laboratory-4/task-1
~/Pr/university-works/o/laboratory-4/task-1 P main +3 !8 743 gcc main.c -o main ✓
~/Pr/university-works/o/laboratory-4/task-1 P main +3 !8 743 ./main ✓

no arg...
arg list:
-p 0 - process info.
-p 1 - id current process and parent.
-p 2 - id real owner.
-p 3 - id real owner.
-p 4 - id group.
~/Pr/university-works/o/laboratory-4/task-1 P main +3 !8 743 ./main -p 0 ✓
id current: 11944
id parent: 11831
id group: 11944
id real owner: 1000
eUID: 1000
id group real: 1000
eGID: 1000
~/Pr/university-works/o/laboratory-4/task-1 P main +3 !8 743 ./main -p 1 ✓
id current: 11950
id parent: 11831
~/Pr/university-works/o/laboratory-4/task-1 P main +3 !8 743 ./main -p 2 ✓
id real owner: 1000
id group real: 1000
~/Pr/university-works/o/laboratory-4/task-1 P main +3 !8 743 ./main -p 3 ✓
eUID: 1000
id group real: 1000
~/Pr/university-works/o/laboratory-4/task-1 P main +3 !8 743 ./main -p 4 ✓

```

```

mykyta@senku:~/Projects/university-works/operating-systems/laboratory-4/task-1
no arg...
arg list:
-p 0 - process info.
-p 1 - id current process and parent.
-p 2 - id real owner.
-p 3 - id real owner.
-p 4 - id group.
~/Pr/university-works/o/laboratory-4/task-1 P main +3 !8 743 ./main -p 0 ✓
id current: 11944
id parent: 11831
id group: 11944
id real owner: 1000
eUID: 1000
id group real: 1000
eGID: 1000
~/Pr/university-works/o/laboratory-4/task-1 P main +3 !8 743 ./main -p 1 ✓
id current: 11950
id parent: 11831
~/Pr/university-works/o/laboratory-4/task-1 P main +3 !8 743 ./main -p 2 ✓
id real owner: 1000
id group real: 1000
~/Pr/university-works/o/laboratory-4/task-1 P main +3 !8 743 ./main -p 3 ✓
eUID: 1000
id group real: 1000
~/Pr/university-works/o/laboratory-4/task-1 P main +3 !8 743 ./main -p 4 ✓
id group: 11969
~/Pr/university-works/o/laboratory-4/task-1 P main +3 !8 743 [ ] ✓

```

Завдання 2

Створити один процес — нащадок. Продемонструвати «непередбачуваність» алгоритму перемикання процесів. За допомогою функції `time()` (обчислення кількості секунд, що пройшли з початку ери UNIX, файл заголовку `time.h`, виклик: `time_t now = time(NULL);`) змусити опрацювати програму задану кількість секунд (2, 3, 5, ... можна вказати або за допомогою макровизначення, або за допомогою опції командного рядка) і обчислити, скільки разів в кожному процесі виконається тіло циклу — збільшення лічильника.

Відповідь



```
mykyta@senku:~/Projects/university-works/operating-systems/laboratory-4/task-2
~/Pr/university-works/o/laboratory-4/task-2 P main +3 !8 743 gcc main.c -o main
~/Pr/university-works/o/laboratory-4/task-2 P main +3 !8 743 ./main
parent process complete = 683114825 times.
child process complete = 678620840 times.
~/Pr/university-works/o/laboratory-4/task-2 P main +3 !8 743
```

Завдання 3

а). У програмі функція `fork` може бути викликана більш ніж один раз. Її можна викликати як з батьківського процесу, так і з процесів — нащадків. Батьківський процес містить локальну змінну. Напишіть

програму, що створює два під-процеса, а кожен з них, в свою чергу, свої два під-процеса. Після кожного виклику `fork` в новому процесі збільшити значення локальної змінної, вивести на екран за допомогою функції `printf` значення цієї локальної змінної, її адресу та ідентифікатори батьківського і дочірнього процесів. Крім того, кожен батьківський процес повинен вивести ідентифікатори своїх дочірніх процесів. Звернути увагу на черговість створення процесів при кожному запуску програми.

б). Програма приймає при запуску в командному рядку натуральне число, що означає кількість процесів-нащадків, які необхідно створити. У кожному дочірньому процесі після запуску виводиться повідомлення і процес (завершується або) йде в нескінченний цикл. Основний процес, закінчивши цикл створення процесів-нащадків виводить на екран список працюючих процесів і (за допомогою послідовності викликів команди `ps` виводить інформацію про кожного з них або) видаляє їх (`kill`). Після роботи програми перевірте роботу процесів. З зауваження: для того, щоб під час роботи програми виконати команди командного процесора, можна скористатися функцією `system()`.

Відповідь

```

mykyta@senku:~/Projects/university-works/operating-systems/laboratory-4/task-3
~/Pr/university-works/o/laboratory-4/task-3  P main +3 !8 743  gcc main.c -o main
~/Pr/university-works/o/laboratory-4/task-3  P main +3 !8 743  ./main

0.parent process id: 12672.
child process id: 12573.
local variable: 0.
local variable address: 0x7ffcb73c120c.
-----
1.parent process id: 12672.
child process id: 12573.
local variable: 1.
local variable address: 0x7ffcb73c120c.
-----
2.parent process id: 12672.
child process id: 12573.
local variable: 2.
local variable address: 0x7ffcb73c120c.
-----
1.parent process id: 12673.
child process id: 12672.
local variable: 1.
local variable address: 0x7ffcb73c120c.
-----
2.parent process id: 12674.
child process id: 12672.
local variable: 2.
local variable address: 0x7ffcb73c120c.
-----
3.parent process id: 12673.
child process id: 1188.
local variable: 2.
local variable address: 0x7ffcb73c120c.
-----
3.parent process id: 12675.

```

```

local variable address: 0x7ffcb73c120c.
-----
2.parent process id: 12674.
child process id: 12672.
local variable: 2.
local variable address: 0x7ffcb73c120c.
-----
3.parent process id: 12673.
child process id: 1188.
local variable: 2.
local variable address: 0x7ffcb73c120c.
-----
3.parent process id: 12675.
5.parent process id: 12674.
child process id: 12673.
child process id: 1188.
local variable: 2.
local variable: 3.
local variable address: 0x7ffcb73c120c.
local variable address: 0x7ffcb73c120c.
-----
4.parent process id: 12673.
5.parent process id: 12676.
child process id: 1188.
child process id: 12674.
local variable: 3.
local variable: 3.
local variable address: 0x7ffcb73c120c.
local variable address: 0x7ffcb73c120c.
-----
4.parent process id: 12677.

```

```

local variable: 2.
local variable: 3.
local variable address: 0x7ffcb73c120c.
local variable address: 0x7ffcb73c120c.
-----
4.parent process id: 12673.
5.parent process id: 12676.
child process id: 1188.
child process id: 12674.
local variable: 3.
local variable: 3.
local variable address: 0x7ffcb73c120c.
local variable address: 0x7ffcb73c120c.
-----
4.parent process id: 12677.
6.parent process id: 12674.
child process id: 12673.
5.parent process id: 12675.
child process id: 1188.
local variable: 3.
child process id: 12673.
local variable: 4.
local variable address: 0x7ffcb73c120c.
local variable: 3.
local variable address: 0x7ffcb73c120c.
local variable address: 0x7ffcb73c120c.
-----
5.parent process id: 12673.
child process id: 1188.

```

```

-----
5.parent process id: 12673.
child process id: 1188.
6.parent process id: 12678.
local variable: 4.
child process id: 12674.
local variable address: 0x7ffcb73c120c.
local variable: 4.
-----
local variable address: 0x7ffcb73c120c.
-----
6.parent process id: 12675.
child process id: 12673.
local variable: 4.
5.parent process id: 12677.
local variable address: 0x7ffcb73c120c.
child process id: 12673.
-----
local variable: 4.
local variable address: 0x7ffcb73c120c.
-----
6.parent process id: 12673.
child process id: 1188.
local variable: 5.
local variable address: 0x7ffcb73c120c.
-----
5.parent process id: 12682.
child process id: 12677.
local variable: 4.
6.parent process id: 12677.
local variable address: 0x7ffcb73c120c.

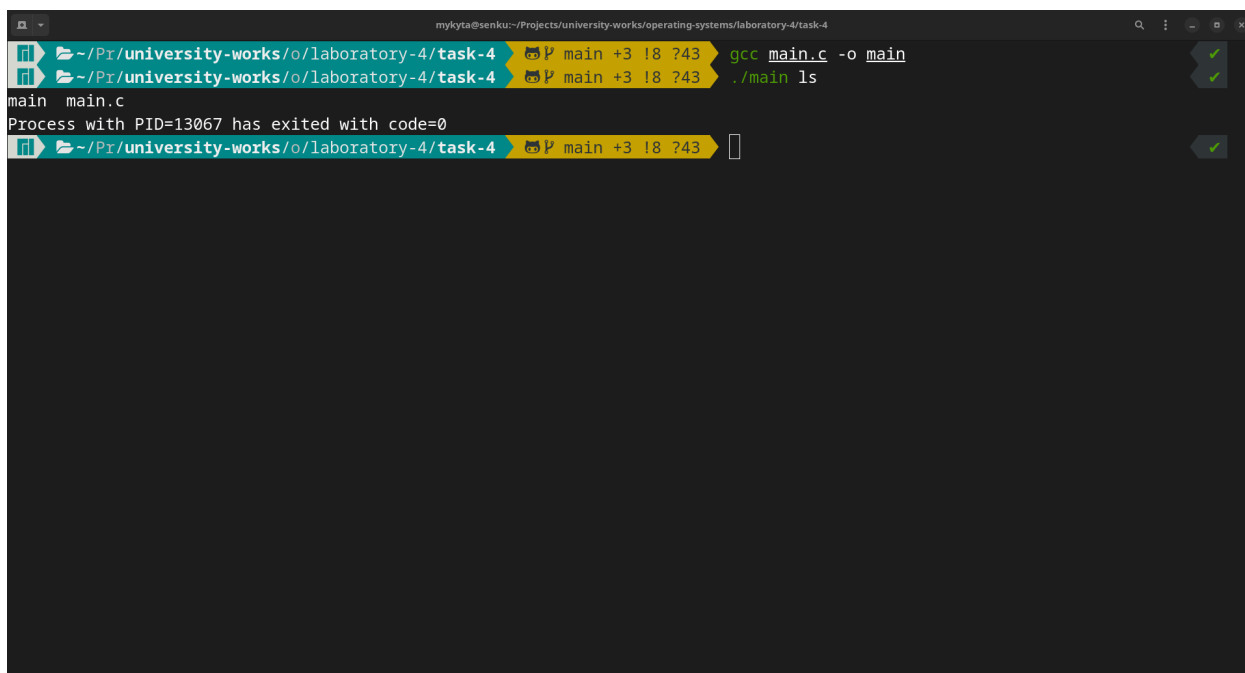
```

```
mykyta@senku:~/Projects/university-works/operating-systems/laboratory-4/task-3
local variable address: 0x7ffcb73c120c.
child process id: 12673.
-----
local variable: 5.
local variable address: 0x7ffcb73c120c.
-----
6.parent process id: 12684.
child process id: 12677.
local variable: 5.
local variable address: 0x7ffcb73c120c.
-----
5.parent process id: 12679.
child process id: 12675.
local variable: 3.
local variable address: 0x7ffcb73c120c.
-----
6.parent process id: 12683.
child process id: 12673.
local variable: 5.
local variable address: 0x7ffcb73c120c.
-----
5.parent process id: 12680.
child process id: 12673.
local variable: 4.
local variable address: 0x7ffcb73c120c.
-----
6.parent process id: 12681.
child process id: 12675.
local variable: 4.
local variable address: 0x7ffcb73c120c.
-----
```

Завдання 5

За допомогою функцій `fork`, `exec`, `wait` створіть власну функцію — спрощений аналог функції `system()` з попереднього заняття (без обробки сигналів — з цим ми розберемось пізніше).

Відповідь



```
mykyta@senku:~/Projects/university-works/operating-systems/laboratory-4/task-4
~/P/ university-works/o/laboratory-4/task-4 P main +3 !8 743 gcc main.c -o main
~/P/ university-works/o/laboratory-4/task-4 P main +3 !8 743 ./main ls
main main.c
Process with PID=13067 has exited with code=0
~/P/ university-works/o/laboratory-4/task-4 P main +3 !8 743
```

Завдання 6

Напишіть програму, яка створює процес — зомбі і показує його наявність в системі, а також його зникнення після виклику функції `wait()`. Для виклику команди `ps` з необхідними можна скористатись функцією `system()`.

Відповідь

```

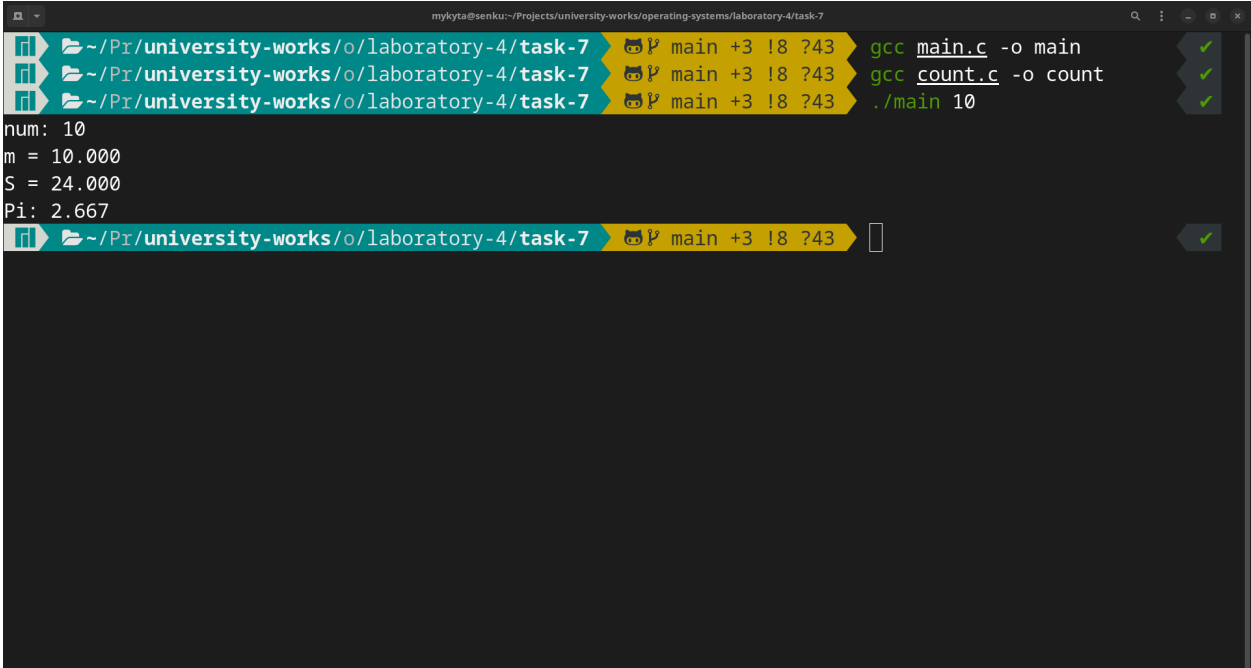
mykyta@senku:~/Projects/university-works/operating-systems/laboratory-4/task-6
~/Pr/university-works/o/laboratory-4/task-6 P main +3 !8 743 gcc main.c -o main
~/Pr/university-works/o/laboratory-4/task-6 P main +3 !8 743 ./main
main main.c
PID TTY TIME CMD
14328 pts/0 00:00:00 zsh
14333 pts/0 00:00:00 zsh
14373 pts/0 00:00:00 zsh
14374 pts/0 00:00:00 zsh
14376 pts/0 00:00:00 gitstatusd
14463 pts/0 00:00:00 main
14464 pts/0 00:00:00 ls <defunct>
14465 pts/0 00:00:00 ps
Code=0
PID TTY TIME CMD
14328 pts/0 00:00:00 zsh
14333 pts/0 00:00:00 zsh
14373 pts/0 00:00:00 zsh
14374 pts/0 00:00:00 zsh
14376 pts/0 00:00:00 gitstatusd
14463 pts/0 00:00:00 main
14470 pts/0 00:00:00 ps
~/Pr/university-works/o/laboratory-4/task-6 P main +3 !8 743

```

Завдання 7

Розглянемо найпростіший спосіб, за допомогою якого можна організувати взаємодію батьківського процесу та процесів — нащадків. Батьківський процес може передавати своєму нащадкові деякі дані за допомогою аргументів однієї з функцій сімейства `exec()`. У свою чергу процес — нащадок при нормальному завершенні передає батьківському процесу свій код повернення. Розглянемо задачу, в якій спробуємо здійснити таку взаємодію.

Відповідь



A terminal window titled "mykyta@senku:~/Projects/university-works/operating-systems/laboratory-4/task-7" displays the following commands and output:

```
~/Pr/university-works/o/laboratory-4/task-7 main +3 !8 ?43 gcc main.c -o main ✓
~/Pr/university-works/o/laboratory-4/task-7 main +3 !8 ?43 gcc count.c -o count ✓
~/Pr/university-works/o/laboratory-4/task-7 main +3 !8 ?43 ./main 10 ✓
```

The output of the program is:

```
num: 10
m = 10.000
S = 24.000
Pi: 2.667
```

The terminal also shows a prompt for the next command:

```
~/Pr/university-works/o/laboratory-4/task-7 main +3 !8 ?43 █ ✓
```