Відокремлений структурний підрозділ «**Волинський фаховий коледж Національного університету харчових технологій**»

Освітньо-професійна програма: **Інженерія програмного забезпечення**

**ОК Системне програмування**

**Звіт лабораторної роботи №3**  
**Тема “**Програмне застосування системних викликів сімейства exec()**”**

Студентки 4-того курсу   
Групи ІПЗ-41  
**Павліхи І.В**

**Луцьк 2025**

Лабораторна робота №3

**Тема:** Програмне застосування системних викликів сімейства exec()

**Мета роботи**

Навчитися параметризувати системні виклики сімейства exec і використовувати їх у програмі. Дослідити процес зміни контексту користувача при виконанні викликів цього сімейства.

***Завдання 1.***

Створити програму з використанням системного виклику execle().

#include <sys/types.h>

#include <unistd.h>

#include <stdio.h>

#include <stdlib.h>

int main(int argc, char \*argv[], char \*envp[]) {

(void) execle("/bin/cat", "cat", "03-2.c", NULL, envp);

printf("Error on program start\n");

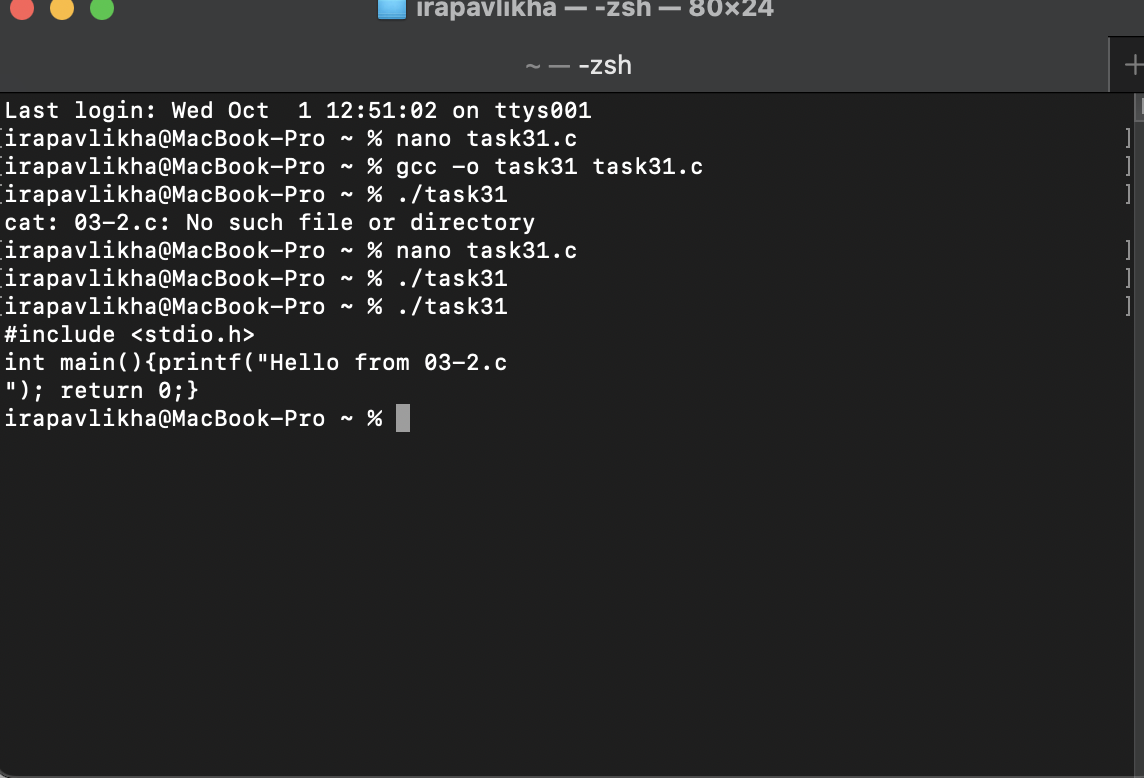
exit(-1);

}

Програма виконує системний виклик execle(), який замінює поточний процес на новий — у цьому випадку команду cat, що виводить вміст файлу 03-2.c.  
 Параметри функції:

* перший аргумент — шлях до виконуваного файлу /bin/cat;
* наступні аргументи — імена команди та файлу, які виконуються;
* NULL позначає кінець списку аргументів;
* envp — масив середовищних змінних, який передається новому процесу.

Якщо execle() виконується успішно, то наступний рядок з printf **ніколи не виконується**, оскільки контекст процесу повністю змінюється.  
 Якщо ж виникла помилка — виводиться повідомлення "Error on program start". execle() дозволяє запускати іншу програму, змінюючи середовище виконання. При цьому поточний процес заміщується новим.

Мал 1.1 - Виконання програми 1

**Завдання 2. (**Додаткове завдання**)**

Створити програму з використанням системного виклику execv(path, argv). Використовуючи програму завдання 1, переробити її для даного системного виклику, враховуючи організацію даних та його необхідні аргументи.

**#include <sys/types.h>**

**#include <unistd.h>**

**#include <stdio.h>**

**#include <stdlib.h>**

**int main() {**

**char \*args[] = {"cat", "03-2.c", NULL};**

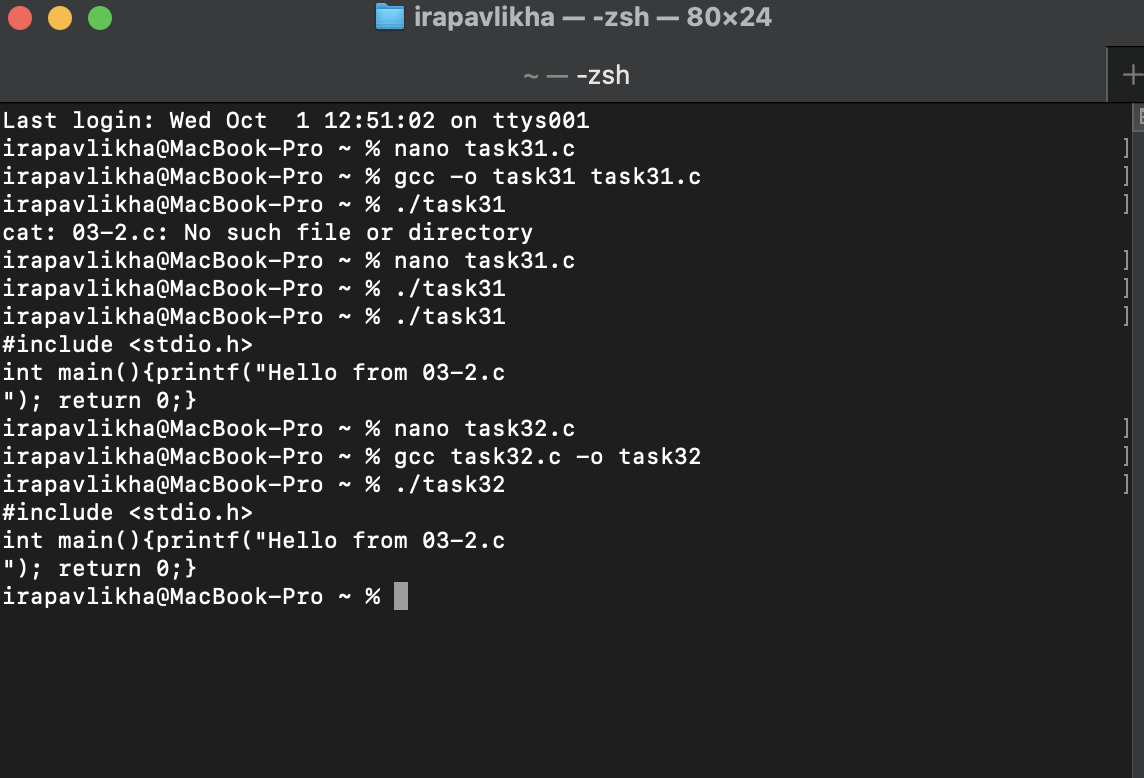
**execv("/bin/cat", args);**

**printf("Error on program start\n");**

**exit(-1);**

**}**

Ця програма робить те саме, що і попередня, але замість execle() використовує execv(). Різниця полягає в тому, що execv() приймає **масив аргументів** (argv[]), але не приймає середовище (envp), тобто використовує **поточне середовище** батьківського процесу. execv() простіший у використанні, коли не потрібно змінювати середовище процесу.

Мал 1.2 - Виконання програми 2

**Завдання 3. (**Додаткове завдання**)**

Створити програму з використанням системного виклику fork() для виводу аргументів командного рядка та параметрів середовища. В якості прикладу написати програму, яка самостійно виводить параметри командного рядка і параметри навколишнього програмного середовища для поточного процесу, використовуючи аргументи int argc, char \*argv[], char \*envp[] для функції main().

#include <sys/types.h>

#include <unistd.h>

#include <stdio.h>

#include <stdlib.h>

int main(int argc, char \*argv[], char \*envp[]) {

pid\_t pid = fork();

if (pid == 0) {

printf("Child process:\n");

for (int i = 0; i < argc; i++) printf("argv[%d] = %s\n", i, argv[i]);

for (int i = 0; envp[i] != NULL; i++) printf("envp[%d] = %s\n", i, envp[i]); } else if (pid > 0) {

wait(NULL);

printf("Parent process finished\n");

} else {

printf("Fork failed\n");

exit(-1);

}

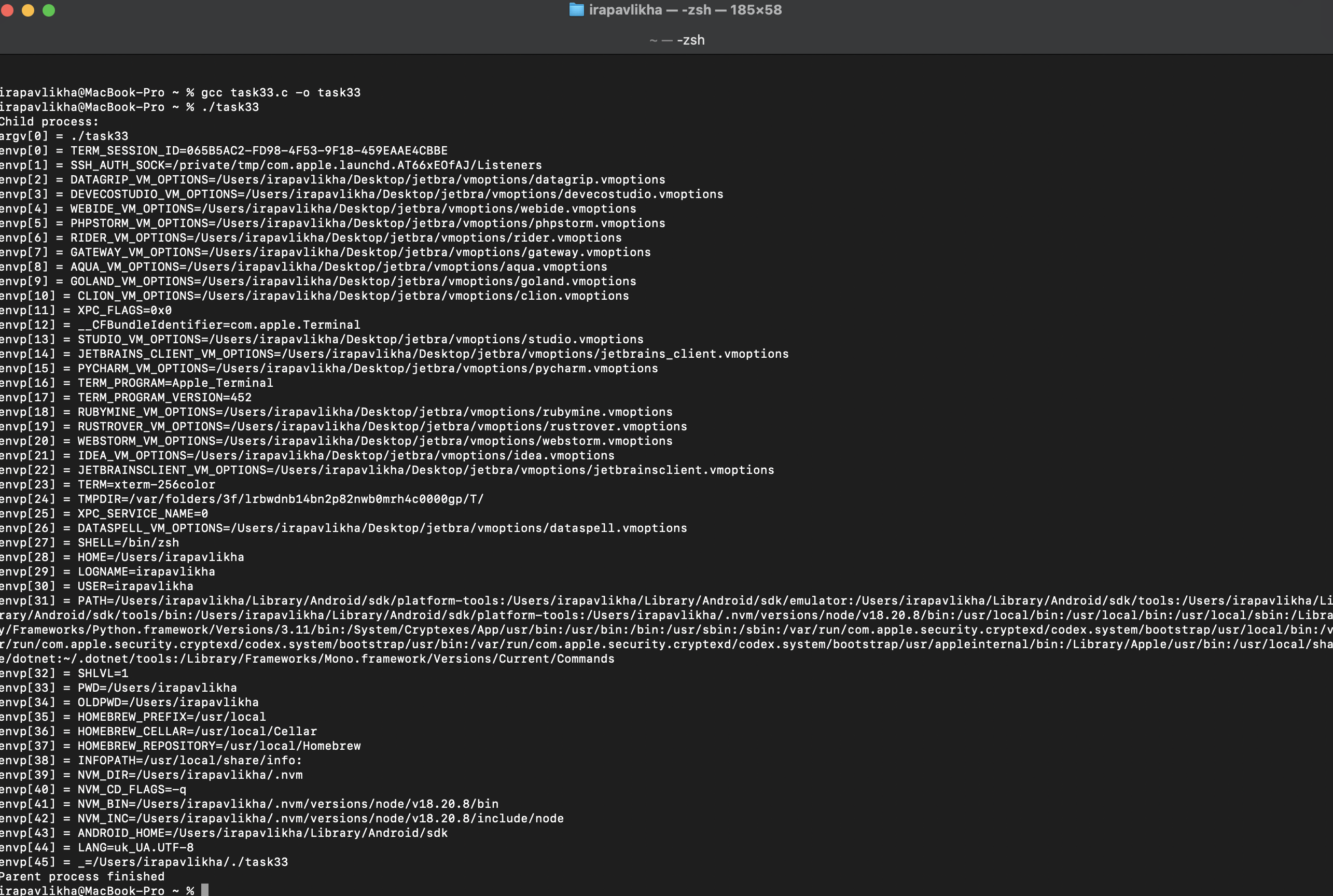
return 0;

}

Програма створює новий процес за допомогою fork().

* Якщо pid == 0, виконується код дочірнього процесу, який виводить усі аргументи командного рядка (argv[]) та змінні середовища (envp[]).
* Якщо pid > 0, виконується батьківський процес, який очікує завершення дочірнього (wait(NULL)) і виводить повідомлення про завершення.
* Якщо fork() не вдався — виводиться повідомлення про помилку.

Дана програма демонструє створення процесів і передачу параметрів командного рядка та середовища.

Мал 1.3 - Виконання програми 3

**Завдання 4. (**Творче завдання**)**

Створити програму з використанням системного виклику execve(path, arg, envp). Використовуючи програму завдання 1, переробити її для даного системного виклику.

#include <sys/types.h>

#include <unistd.h>

#include <stdio.h>

#include <stdlib.h>

int main(int argc, char \*argv[], char \*envp[]) {

char \*args[] = {"cat", "03-2.c", NULL};

execve("/bin/cat", args, envp);

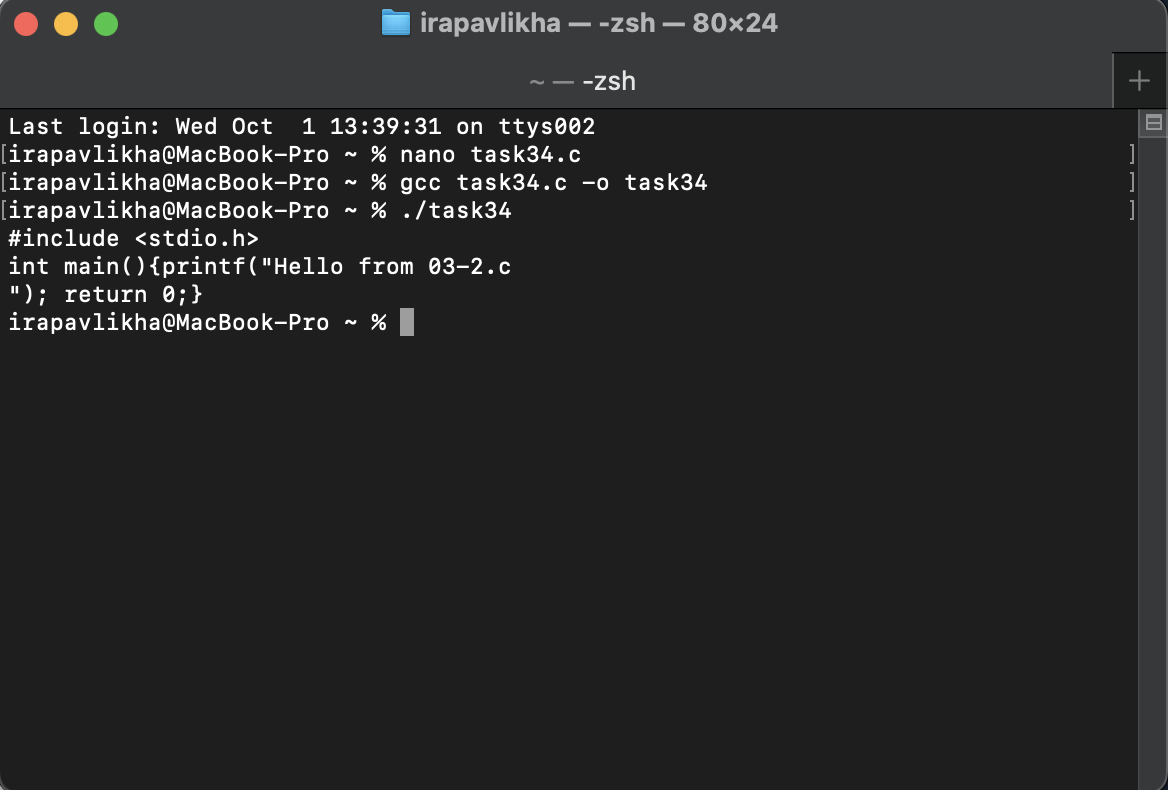
printf("Error on program start\n");

exit(-1);  
}

execve() — це базова форма всіх викликів сімейства exec.  
 Вона приймає як аргументи:

* шлях до програми,
* масив аргументів,
* масив змінних середовища.

Програма виконує команду cat 03-2.c, передаючи поточне середовище envp. execve() — найгнучкіший системний виклик, який дозволяє явно задавати як аргументи, так і середовище процесу.

Мал 1.4 - Виконання програми 4

**Завдання 5. (**Творче завдання**)**

Створити програму з використанням системного виклику fork() і execle (path, arg, envp) з різною поведінкою батьківського і новонародженого процесів так, щоб новостворений процес запускав на виконання нову (будь-яку найпростішу) програму. Виконуючи дане завдання використовуйте програми завдання 1 лабораторних робіт 2 і 3.

#include <sys/types.h>

#include <unistd.h>

#include <stdio.h>

#include <stdlib.h>

int main(int argc, char \*argv[], char \*envp[]) {

pid\_t pid = fork();

if (pid == 0) {

execle("/bin/ls", "ls", "-l", NULL, envp);

printf("Error starting child process\n");

exit(-1);

} else if (pid > 0) {

printf("Parent process running\n");

wait(NULL);

printf("Parent process finished\n");

} else {

printf("Fork failed\n");

exit(-1);

}

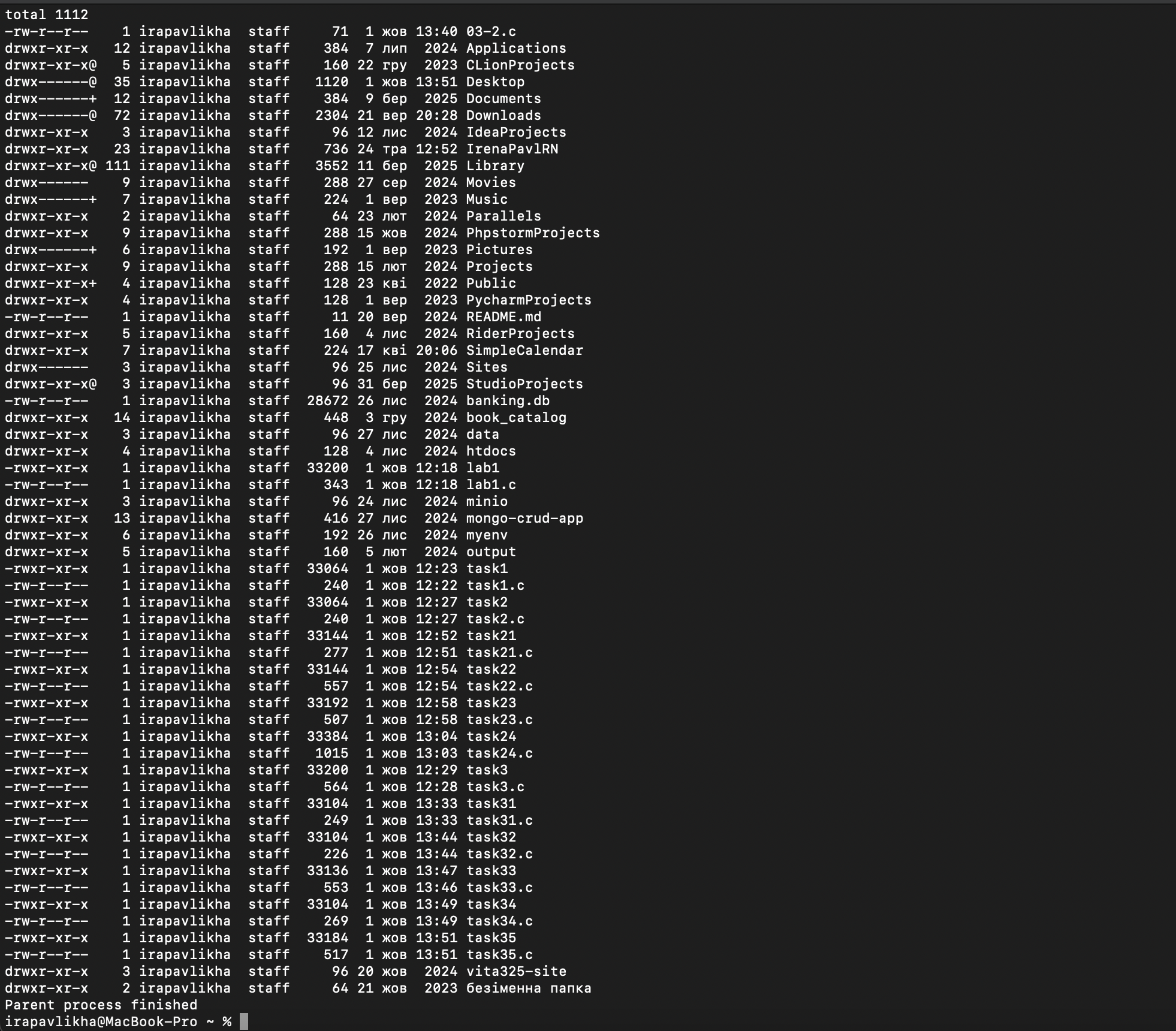
return 0;

}

У цій програмі використовується комбінація системних викликів fork() і execle().

* Батьківський процес створює дочірній.
* Дочірній процес виконує команду ls -l через execle(), тобто показує список файлів у поточній директорії.
* Батьківський процес очікує завершення дочірнього і після цього виводить повідомлення про завершення.

Таке поєднання дозволяє запускати нові процеси, які виконують інші програми, не перериваючи основний процес. Це основа для створення багатопроцесних системних програм.

 Мал 1.5 - Виконання програми 5

**Порівняння системних викликів execle, execv, execve**

|  |  |  |  |
| --- | --- | --- | --- |
| **Виклик** | **Як передаються аргументи** | **Як передається середовище** | **Особливості** |
| execle(path, arg0, arg1, ..., NULL, envp) | Списком (через кому) | Передається явно (envp) | Дозволяє задавати нове середовище |
| execv(path, argv[]) | Масивом (argv[]) | Використовується поточне середовище | Простий у використанні |
| execve(path, argv[], envp[]) | Масивом (argv[]) | Передається явно (envp[]) | Найгнучкіший, базовий системний виклик |

Сімейство системних викликів exec() дозволяє **змінювати контекст процесу**, запускаючи нову програму замість поточної. Виклики execv, execle, execve відрізняються способом передачі аргументів та середовища. Поєднання fork() і exec() дозволяє створювати нові процеси, які виконують інші програми — це основа роботи операційних систем.

Виконані завдання показали різницю між способами передачі параметрів та змін середовища, а також механізм взаємодії між батьківським і дочірнім процесами.

**Пояснення до питань самоконтролю**

**1. Зміна контексту користувача при exec():**  
 Поточний процес замінюється новим — код, дані і стек стають як у нової програми, PID лишається той самий. Якщо exec() не вдалось — виконується наступний код.

**2. Від чого залежать параметри exec():**  
 Від шляху до програми, масиву аргументів (argv[]) та масиву середовища (envp[]), якщо він передається.

**3. Аргументи командного рядка:**  
 Це дані, які користувач передає програмі при запуску для керування її роботою. Роль — передати параметри виконання.

**4. argc:**  
 Кількість аргументів командного рядка (включно з ім’ям програми).

**5. argv[]:**  
 Масив рядків, що містить всі аргументи командного рядка, argv[0] — ім’я програми.

**6. envp[]:**  
 Масив рядків середовища процесу, формуються операційною системою і передаються процесу при запуску.

**7. Компіляція з аргументами:**

gcc program.c -o a.out  
./a.out arg1 arg2

**8. Тривіальний вихід з програми:**  
 Завершення програми без помилок, реалізується return 0; або exit(0);.

**Висновок**

Виконана лабораторна робота показала, як системні виклики сімейства exec() змінюють контекст процесу, дозволяючи запускати інші програми у межах існуючого процесу. Було досліджено різницю між execle, execv та execve у способі передачі аргументів та середовища процесу. Поєднання fork() і exec() продемонструвало створення нових процесів та організацію їх взаємодії з батьківським процесом. Практична робота допомогла зрозуміти роль аргументів командного рядка (argc, argv[]) та змін середовища (envp[]) для налаштування виконання програм. Отримані знання є основою для розробки системних програм та управління процесами в ОС Linux.