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

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

**ОК Системне програмування**

**Звіт лабораторної роботи №2**  
**Тема “**Створення програм з системним викликом fork()**”**

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

**Луцьк 2025**

Лабораторна робота №1

**Тема: Створення програм з системним викликом fork()**

**Мета роботи**

**Навчитися створювати системні програми з можливістю системного виклику fork() для створення процесу в системі.**

***Завдання 1.***

*Створення програми з використанням системного виклику fork() з однаковою роботою батьківського і новоствореного процесів.*

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

**#include <unistd.h>**

**#include <stdio.h>**

**int main()**

**{**

**pid\_t pid, ppid;**

**int a = 0;**

**fork();**

**a = a + 1;**

**pid = getpid();**

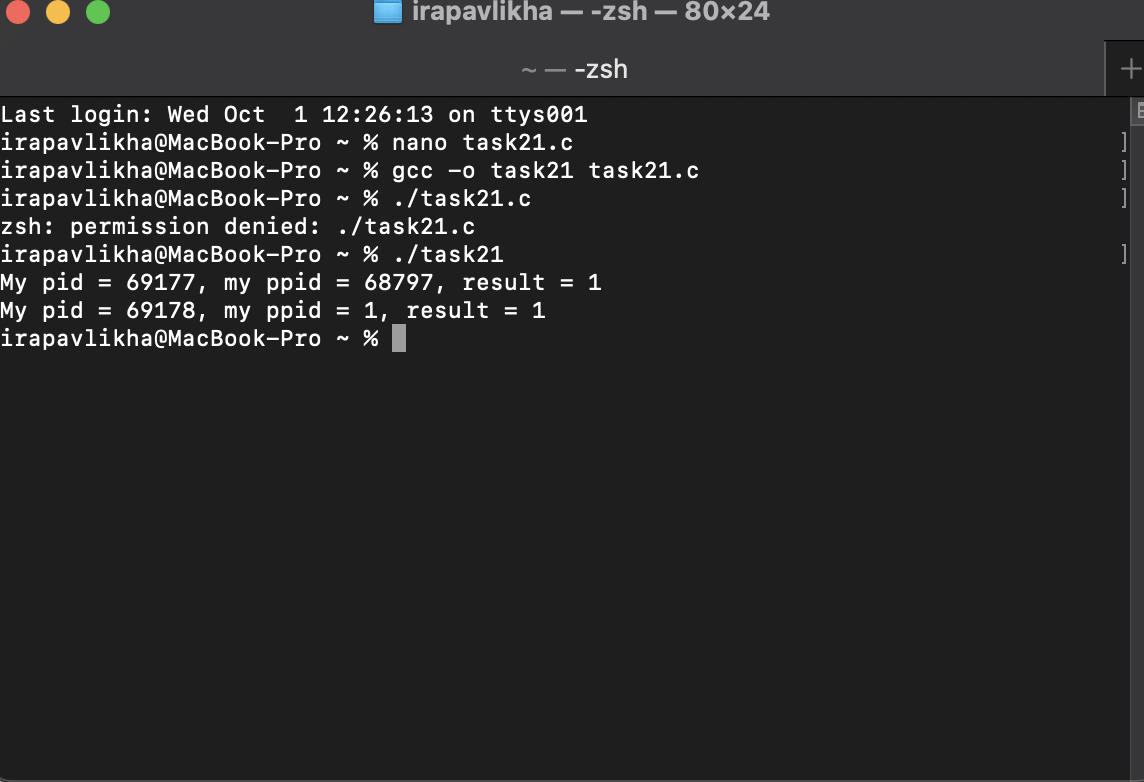
**ppid = getppid();**

**printf("My pid = %d, my ppid = %d, result = %d\n", (int)pid, (int)ppid, a);**

**return 0;**

**}**

Програма демонструє створення нового процесу за допомогою системного виклику fork(). Після виклику створюється копія батьківського процесу — **дочірній процес**, який виконує ту саму послідовність інструкцій. Змінна a копіюється, тому в обох процесах вона має власне значення, яке збільшується незалежно. Функції getpid() і getppid() повертають ідентифікатори поточного та батьківського процесів, що дозволяє відрізнити їх у виводі.  
 Результатом є два рядки — по одному від кожного процесу, що підтверджує роботу fork().

Мал 1.1 - Виконання програми 1

**Завдання 2.**

*Створення програми з використанням системного виклику fork() з різною роботою батьківського і новоствореного процесів.*

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

**#include <unistd.h>**

**#include <stdio.h>**

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

**#include <stdlib.h>**

**int main()**

**{**

**pid\_t pid;**

**pid = fork();**

**if(pid < 0) return 1;**

**if(pid == 0)**

**{**

**int sum = 0;**

**for(int i = 1; i <= 10; ++i) sum += i;**

**printf("Child: sum 1..10 = %d\n", sum);**

**\_exit(0);**

**}**

**else**

**{**

**long prod = 1;**

**for(int i = 1; i <= 10; ++i) prod \*= i;**

**int status;**

**waitpid(pid, &status, 0);**

**printf("Parent: product 1..10 = %ld\n", prod);**

**}**

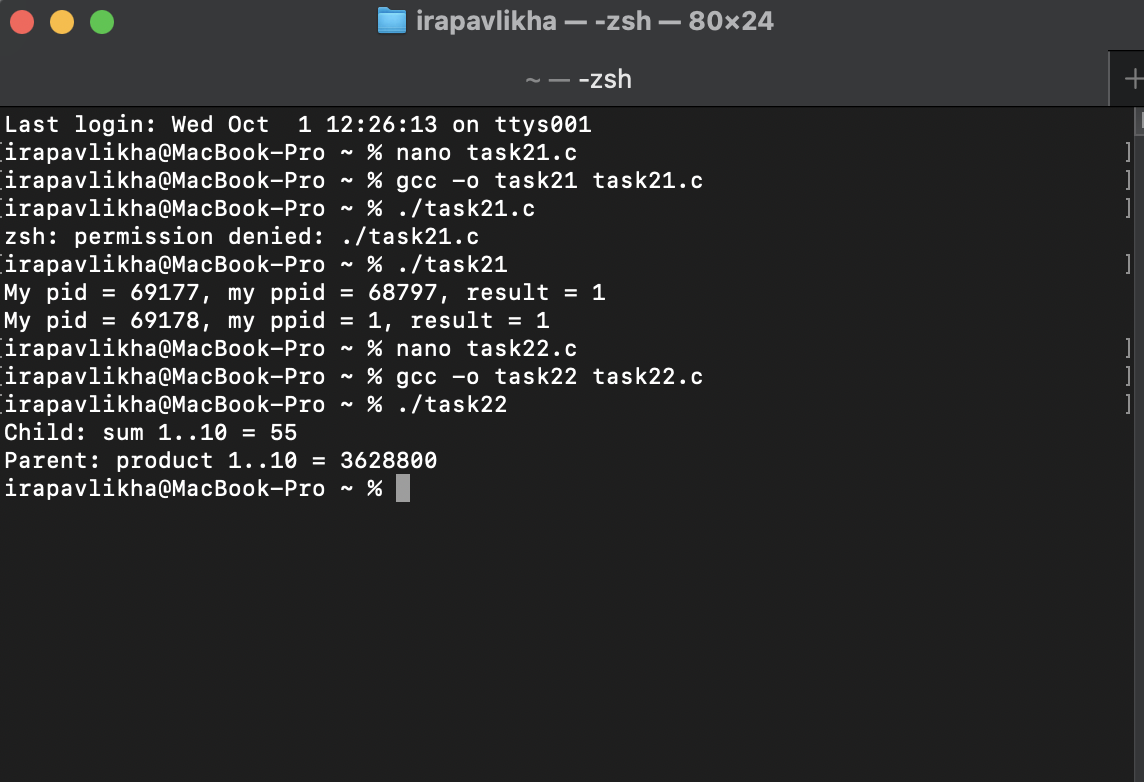
**return 0;**

**}**

У цій програмі показано роботу двох процесів із різними завданнями:

* **Дочірній процес** обчислює суму чисел від 1 до 10.
* **Батьківський процес** після завершення дочірнього обчислює добуток чисел від 1 до 10.

Функція waitpid() змушує батьківський процес дочекатися завершення дитини, щоб уникнути одночасного виводу.  
 Результати підтверджують, що після виклику fork() обидва процеси працюють незалежно та можуть виконувати різні операції.

Мал 1.2 - Виконання програми 2

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

*Використання системного виклику vfork() та функції execlp() для передачі керування іншій програмі.*

**#define \_POSIX\_C\_SOURCE 200112L**

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

**#include <unistd.h>**

**#include <stdio.h>**

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

**#include <stdlib.h>**

**int main()**

**{**

**pid\_t pid;**

**int status;**

**pid = vfork();**

**if(pid < 0) return 1;**

**if(pid == 0)**

**{**

**execlp("echo", "echo", "vfork child: hello from exec", NULL);**

**\_exit(127);**

**}**

**else**

**{**

**waitpid(pid, &status, 0);**

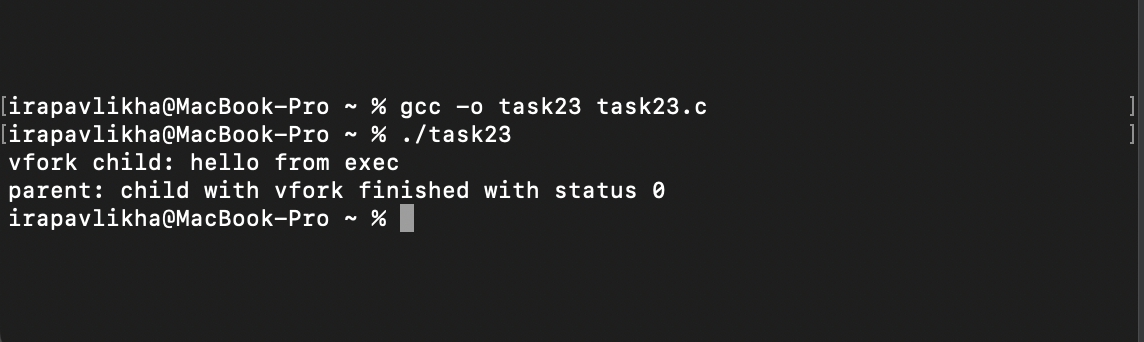
**printf("parent: child with vfork finished with status %d\n", WEXITSTATUS(status));**

**}**

**return 0;**

**}**

У цьому прикладі застосовується системний виклик vfork(), який створює процес без копіювання пам’яті. Дочірній процес виконує команду echo через execlp() — тобто передає управління іншій програмі. Батьківський процес чекає завершення дочірнього через waitpid(). Це демонструє принципову різницю між fork() і vfork() та показує, як можна використовувати системні виклики для запуску зовнішніх програм.

Мал 1.3 - Виконання програми 3

**Завдання 4. (**Творче завдання**)**

*Використання системного виклику fork() для роботи з різними типами даних: числовими, рядковими та структурними.*

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

**#include <unistd.h>**

**#include <stdio.h>**

**#include <string.h>**

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

**#include <stdlib.h>**

**struct Data { int id; double value; char note[64]; };**

**int main()**

**{**

**pid\_t pid;**

**int number = 7;**

**char text[64];**

**struct Data d;**

**strcpy(text, "orig");**

**d.id = 1;**

**d.value = 3.14;**

**strcpy(d.note, "initial");**

**pid = fork();**

**if(pid < 0) return 1;**

**if(pid == 0)**

**{**

**number += 5;**

**strcat(text, "\_child");**

**d.id = 2;**

**d.value \*= 2;**

**strcpy(d.note, "child\_changed");**

**printf("child: number=%d text=%s data={%d,%.2f,%s}\n", number, text, d.id, d.value, d.note);**

**\_exit(0);**

**}**

**else**

**{**

**number -= 3;**

**strcat(text, "\_parent");**

**d.id = 99;**

**d.value += 1.86;**

**strcpy(d.note, "parent\_changed");**

**int status;**

**waitpid(pid, &status, 0);**

**printf("parent: number=%d text=%s data={%d,%.2f,%s}\n", number, text, d.id, d.value, d.note);**

**}**

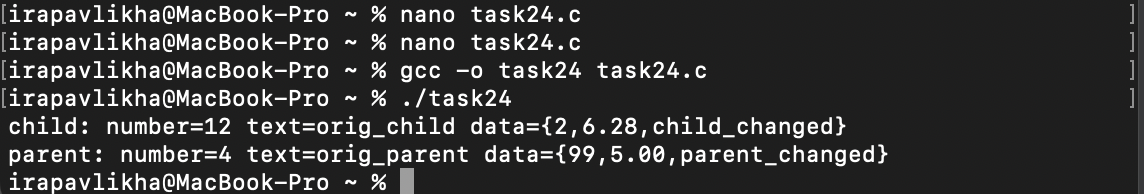
**return 0;**

**}**

Ця програма демонструє роботу fork() з різними типами даних.  
 У кожному процесі створюються власні копії змінних, рядків і структур.

* Дочірній процес змінює значення змінних, додає до рядка \_child, оновлює структуру Data.
* Батьківський процес виконує свої зміни з суфіксом \_parent.

Результати показують, що пам’ять процесів після fork() не спільна — кожен має власні копії даних.  
 Це підтверджує принцип **незалежності процесів**, що є ключовим у системному програмуванні.

Мал 1.4 - Виконання програми 4

**Пояснення до питань самоконтролю**

**1. У чому перевага виклику fork()?**  
 Дозволяє створювати копію процесу для паралельного виконання. Це основа багатозадачності в Unix-системах.

**2. Чи всі ОС дозволяють створювати процеси?**  
 Ні. У Unix/Linux — через fork(), у Windows — через CreateProcess(). Механізм залежить від системи.

**3. Для чого потрібен власний процес?**  
 Для незалежного виконання частин програми, підвищення надійності та паралельної обробки задач.

**4. Навіщо приведення типів у програмі 1?**  
 Щоб коректно вивести pid\_t як ціле число у printf(), бо pid\_t може мати інший тип, ніж int.

**Висновок**

У ході виконання лабораторної роботи було досліджено механізм створення процесів за допомогою fork() і vfork() у середовищі Linux; продемонстровано можливість паралельного виконання процесів і різні варіанти їх взаємодії; вивчено роботу функцій getpid(), getppid(), waitpid(), execlp(); показано незалежність пам’яті між процесами після виклику fork(); реалізовано роботу з числовими, рядковими та структурними типами даних у різних процесах.  
Робота виконана повністю: **основні, додаткові та творчі завдання реалізовано**. Отримано практичні навички системного програмування з використанням процесів у Linux.