Лабораторна робота №6 Потоки. Керування

Завдання №1: Атрибути потоку

Передаємо дані між потоками, керуємо атрибутами потоків, такими як "від'єднаний" або "звичайний", і синхронізуємо доступ до спільного глобального масиву. Окремий потік забезпечує постійний моніторинг стану масиву, виводячи його вміст у режимі реального часу.

```
mint@asus-mint:~/git/op-course/lab6/task1$ ./bin/main 5
Current state of global array: -1 -1 -1 -1 -1
Thread 0 generated number 4
Thread 1 generated number 7
Current state of global array: 4 7 -1 -1 -1
Thread 2 generated number 8
Current state of global array: 4 7 8 -1 -1
Thread 3 generated number 6
Current state of global array: 4 7 8 6 -1
Thread 0 finished.
Thread 4 generated number 4
Current state of global array: 4 7 8 6 4
Display thread terminating as global array is fully populated.
Thread 1 finished.
Thread 4 finished.
Thread 3 finished.
Thread 2 finished.
All calculation threads have finished.
mint@asus-mint:~/git/op-course/lab6/task1$
```

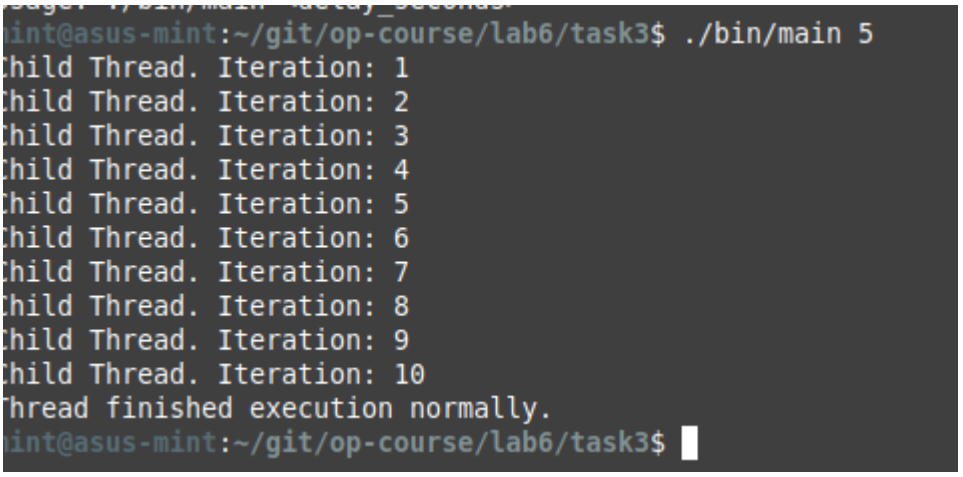
Завдання №2: Потік, що скасовується асинхронно

Встановлюємо режим асинхронного скасування для потоку, що дозволяє завершувати його роботу у будь-який момент. Аналізуємо статус завершення, щоб зрозуміти, чи було потік скасовано примусово, чи він завершився самостійно.

```
mint@asus-mint:~/git/op-course/lab6/task2$ ./bin/main 3
Child Thread. Iteration: 0
Child Thread. Iteration: 1
Child Thread. Iteration: 2
Thread was canceled.
mint@asus-mint:~/git/op-course/lab6/task2$ task2
```

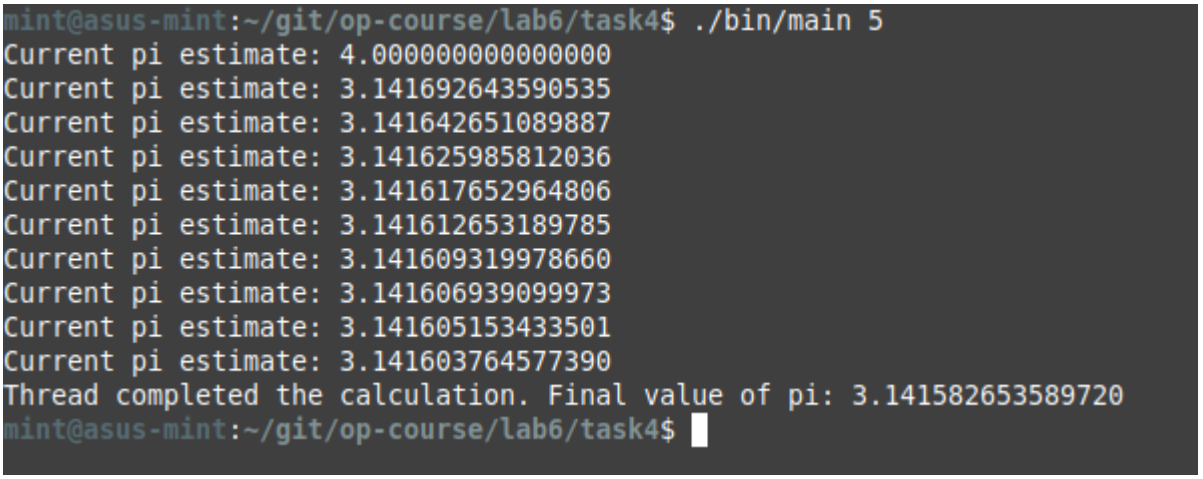
Завдання №3: Потік, що не скасовується

Забезпечуємо безпеку виконання завдань, роблячи потік неможливим до скасування. Це дозволяє виконати критично важливу роботу до кінця, навіть якщо основний потік намагається його завершити.



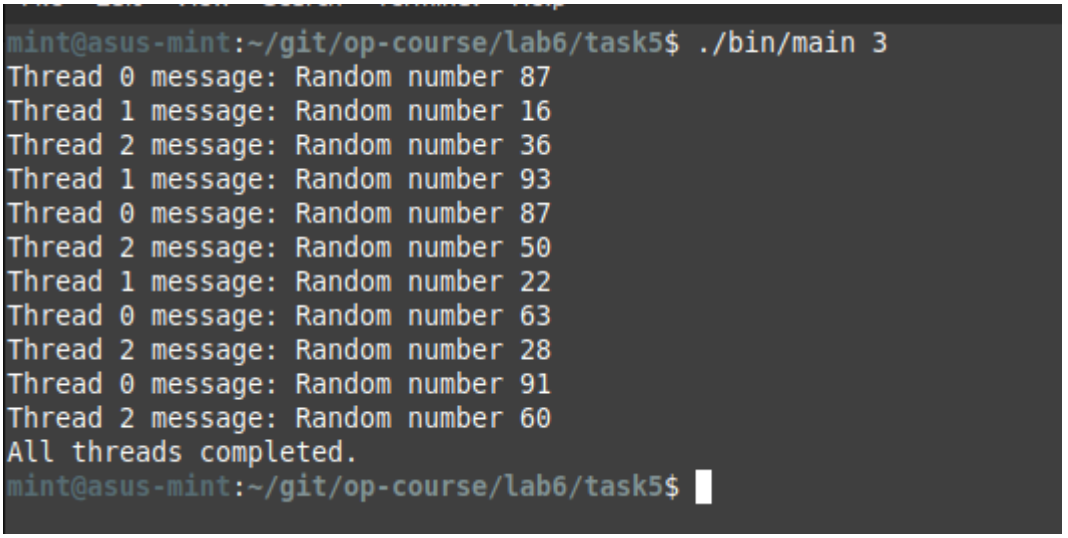
Завдання №4: Потік, що скасовується синхронно

Налаштовуємо потік на скасування тільки у контрольованих точках виходу. Це дає змогу забезпечити безпечне виконання коду, навіть якщо обчислення довготривалі. Виводимо поточні результати обчислення числа π за формулою Лейбніца та аналізуємо причину завершення потоку.



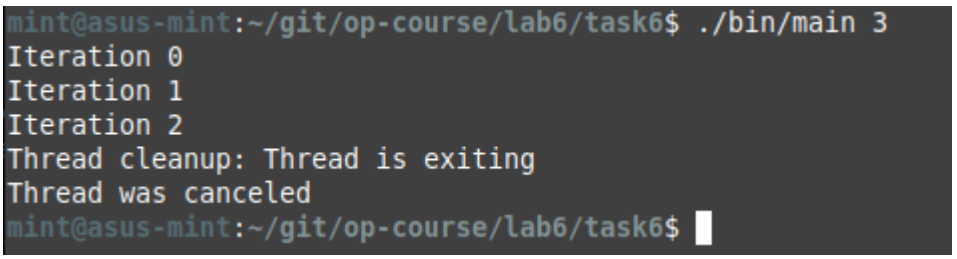
Завдання №5: Потокові дані

Використовуємо механізм потокових даних для ізоляції змінних кожного потоку, що гарантує незалежність їхньої роботи. Це допомагає уникнути конфліктів доступу до спільних ресурсів і дає змогу автоматично керувати пам'яттю через спеціальні функції очищення.



Завдання №6: Обробники очищення

Додаємо обробники очищення для забезпечення коректного завершення роботи потоку-нащадка. Потік гарантує виконання важливих дій, таких як виведення повідомлення з даними, навіть у випадку примусового скасування. Це вчить використовувати pthread_cleanup_push та pthread_cleanup_pop для забезпечення безпечного управління ресурсами та виконання необхідних дій перед завершенням роботи.



Висновок

Ми ознайомилися з основними принципами багатопоточності в C: створення потоків, налаштування їхніх атрибутів, методи скасування, синхронізація роботи і використання локальних потокових даних. Ці знання дозволяють ефективно працювати з паралельними обчисленнями, оптимізувати програми для багатоядерних процесорів і забезпечувати їхню стабільність у багатопоточному середовищі.