

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.

```
mint@asus-mint:~/git/op-course/lab6/task3$ ./bin/main 5
Child Thread. Iteration: 1
Child Thread. Iteration: 2
Child Thread. Iteration: 3
Child Thread. Iteration: 4
Child Thread. Iteration: 5
Child Thread. Iteration: 6
Child Thread. Iteration: 7
Child Thread. Iteration: 8
Child Thread. Iteration: 9
Child Thread. Iteration: 10
Thread finished execution normally.
mint@asus-mint:~/git/op-course/lab6/task3$
```

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.

```
mint@asus-mint:~/git/op-course/lab6/task4$ ./bin/main 5
Current pi estimate: 4.0000000000000000
Current pi estimate: 3.141692643590535
Current pi estimate: 3.141642651089887
Current pi estimate: 3.141625985812036
Current pi estimate: 3.141617652964806
Current pi estimate: 3.141612653189785
Current pi estimate: 3.141609319978660
Current pi estimate: 3.141606939099973
Current pi estimate: 3.141605153433501
Current pi estimate: 3.141603764577390
Thread completed the calculation. Final value of pi: 3.141582653589720
mint@asus-mint:~/git/op-course/lab6/task4$
```

Завдання №5: Потокові дані

Використовуємо механізм поточкових 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.

```
mint@asus-mint:~/git/op-course/lab6/task5$ ./bin/main 3
Thread 0 message: Random number 87
Thread 1 message: Random number 16
Thread 2 message: Random number 36
Thread 1 message: Random number 93
Thread 0 message: Random number 87
Thread 2 message: Random number 50
Thread 1 message: Random number 22
Thread 0 message: Random number 63
Thread 2 message: Random number 28
Thread 0 message: Random number 91
Thread 2 message: Random number 60
All threads completed.
mint@asus-mint:~/git/op-course/lab6/task5$
```

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.

```
mint@asus-mint:~/git/op-course/lab6/task6$ ./bin/main 3
Iteration 0
Iteration 1
Iteration 2
Thread cleanup: Thread is exiting
Thread was canceled
mint@asus-mint:~/git/op-course/lab6/task6$
```

Висновок

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