**Лабораторна робота №2**

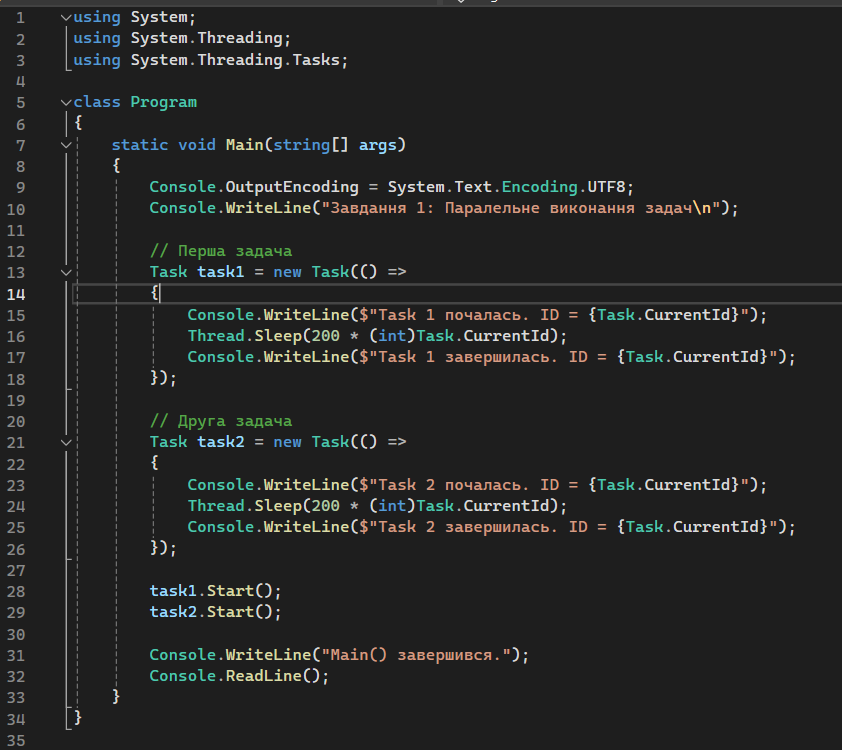
Тема 2: ПАРАЛЕЛІЗМ ДАНИХ ТА ПАРАЛЕЛІЗМ ЗАДАЧ (Ч.1)

Завдання:

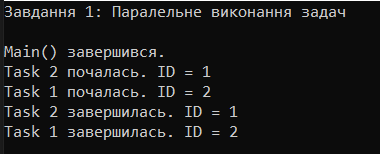
1. Створити програму, що створює дві задачі, які виконуються паралельно. Затримку методом Sleep() організувати на величину 200 мс та пропорційно ідентифікатору задачі.
2. Організувати очікування виконання задач методом WaitAll().
3. Визначити задачу для виконання у вигляді лямбда-виразу.
4. Створити програму паралельних обчислень за допомогою виклику методу Invoke(), де в якості аргументів застосовуються лямбда-вирази.

Хід роботи:

1. У ході виконання першого завдання було створено дві задачі, які виконуються паралельно. Затримку реалізовано за допомогою Thread.Sleep() із тривалістю, пропорційною ідентифікатору кожної задачі.

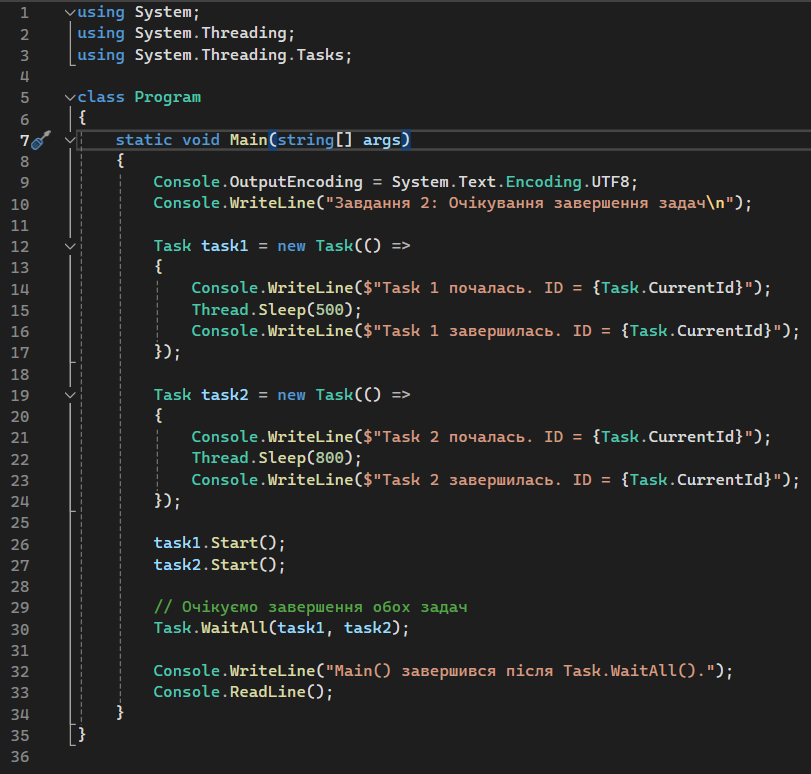


Отримуємо такий результат:

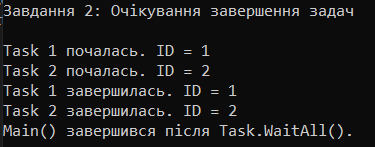


Паралельне виконання відбулося правильно — обидві задачі працювали одночасно, а результат залежав від часу затримки, встановленого за ID задачі. Це підтверджує, що паралельність реалізована коректно.

1. У другому завданні було реалізовано дві паралельні задачі з різним часом виконання. Метод Task.WaitAll() використано для синхронізації — головний потік очікує завершення обох задач перед продовженням своєї роботи.

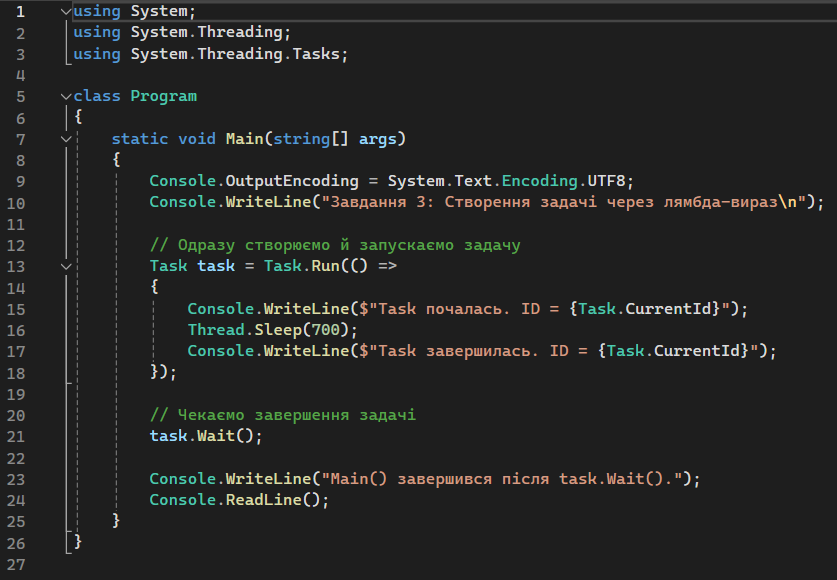


У результаті:

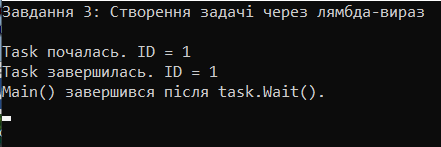


У цьому завданні було використано метод Task.WaitAll(), який забезпечив очікування завершення всіх задач перед продовженням виконання головного потоку. На відміну від попереднього прикладу, де Main() завершувався одразу після запуску задач, тут Main() завершується лише після повного виконання task1 і task2, що дозволяє синхронізувати роботу задач у багатопотоковому середовищі.

1. У цьому завданні задача створюється за допомогою лямбда-виразу – анонімної функції, яку безпосередньо передають у конструктор Task. Це дозволяє компактно визначати логіку задачі без необхідності окремого методу. Для синхронізації виконання використовується метод Wait(), що очікує завершення задачі перед завершенням Main().

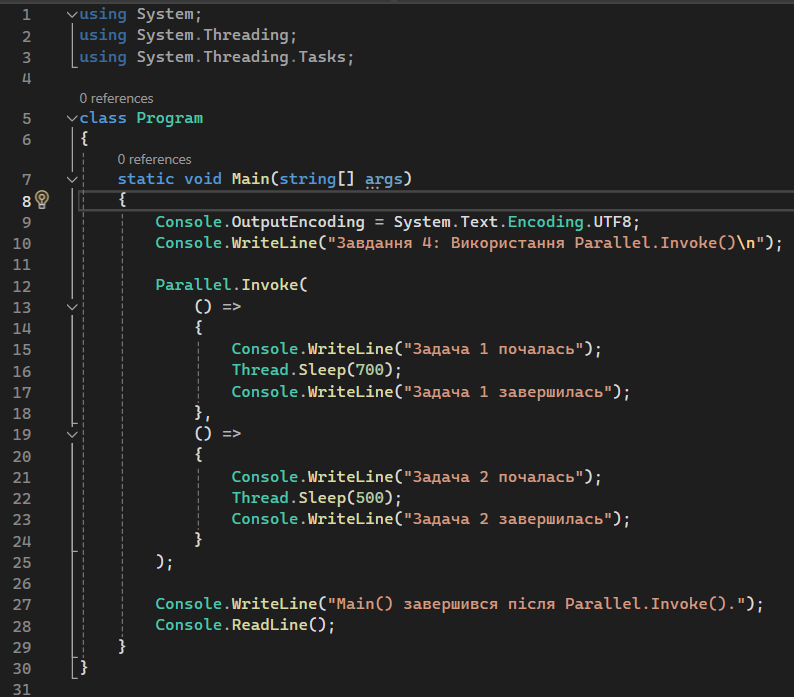


У результаті:

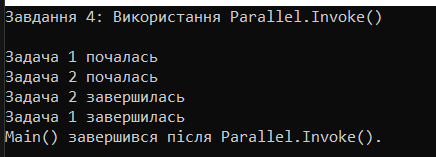


Висновок: Задача, створена через лямбда-вираз, виконується аналогічно задачі, створеній через метод, але має більш лаконічний вигляд. Використання task.Wait() забезпечує завершення Main() лише після завершення задачі, що дозволяє контролювати послідовність виконання коду.

1. У цьому завданні реалізовано паралельне виконання двох задач за допомогою методу Parallel.Invoke(), який приймає лямбда-вирази як аргументи. Метод автоматично запускає всі задачі паралельно та очікує їх завершення, не потребуючи додаткового виклику Wait().



У результаті:



Метод Parallel.Invoke() забезпечує зручний спосіб паралельного запуску декількох задач і автоматично синхронізує завершення їх виконання. Завдяки використанню лямбда-виразів код стає компактним, а виконання — ефективним.

Висновок: У цій лабораторній роботі було розглянуто створення та виконання паралельних задач у C# за допомогою бібліотеки TPL. Було використано запуск задач із різними затримками, очікування їх завершення методом WaitAll(), створення задач через лямбда-вирази та паралельне виконання методом Parallel.Invoke(). Робота показала основні підходи до реалізації паралелізму задач і їх синхронізації в багатопотоковому середовищі.