Ильяхова Алиса, Б9122-02.03.01сцт

Лабораторная работа №9. Параллельные циклы и LINQ-запросы

Вариант 9

1. Цель и задачи работы

Цель:

Научиться использовать параллельные циклы и параллельные LINQ-запросы.

Задачи:

- Изучить класс Parallel;
- Научиться использовать методы Parallel.For и Parallel.ForEach;
- Научиться использовать параллельные LINQ-запросы (PLINQ).

2. Реализация индивидуального задания

2.1. Условие варианта 9

Согласно таблице на стр. 31:

- **Тип делегата**: лямбда-выражение
- **Решаемая задача**: Метод возвращает результат шифрования строки: каждый исходный символ строки заменяется шифрованным символом, код которого на n больше кода исходного символа.
- **Входные параметры**: Два параметра — исходная строка, число сдвига n.

2.2. Метод шифрования

Реализован метод EncryptString:

- Выполняет побайтовое шифрование.
- Имитирует долгую операцию через Thread.Sleep(500).

2.3. Последовательная обработка

Для сравнения реализована последовательная обработка через обычный цикл for.

2.4. Параллельный цикл Parallel.ForEach

- Использован Parallel.ForEach для обработки массива строк.

- Применено **синхронизирующее блокирование** (lock) для корректного вывода в консоль.
- Сохранён порядок результатов через индексацию.

2.5. PLINQ (параллельный LINQ)

- Использован метод AsParallel() для преобразования запроса в параллельный.
- Применён AsOrdered() для сохранения исходного порядка элементов.
- Результаты получены через Select и ToArray().

2.6. Проверка корректности

Выполнена проверка совпадения результатов всех трёх подходов с помощью SequenceEqual.

3. Ответы на контрольные вопросы

1. **Что такое PLINQ? Как преобразовать последовательный LINQ-запрос в параллельный?**

PLINQ (Parallel LINQ) — это технология параллельной обработки данных в LINQ.

Для преобразования достаточно вызвать метод AsParallel() над источником данных.

2. **Как сохранить порядок следования элементов при использовании PLINQ?**

С помощью метода AsOrdered(), который гарантирует, что выходная последовательность будет иметь тот же порядок, что и входная.

3. **Опишите назначение методов Parallel.For и Parallel.ForEach.**
    a. Parallel.For — параллельная версия цикла for для числовых диапазонов.
    b. Parallel.ForEach — параллельная версия цикла foreach для коллекций.
4. **Как обеспечить синхронизацию доступа к общему ресурсу в параллельном цикле?**

С помощью примитивов синхронизации: lock, Monitor, Mutex, или потокобезопасных коллекций (ConcurrentBag, ConcurrentDictionary).

4. Экранные формы и листинг программы

4.1. Консольный вывод программы

```
=== Лабораторная работа №9. Вариант 9 ===
Параллельные циклы и LINQ-запросы

1. Последовательная обработка:
  "Hello" → "Khoor"
  "World" → "Zruog"
  "Secret" → "Vhfuhw"
  "Message" → "Phvvdjh"
  "Test" → "Whvw"

2. Параллельный цикл Parallel.ForEach:
  "Test" → "Whvw" (Поток 21)
  "Hello" → "Khoor" (Поток 18)
  "Secret" → "Vhfuhw" (Поток 19)
  "World" → "Zruog" (Поток 24)
  "Message" → "Phvvdjh" (Поток 20)

3. PLINQ (AsParallel):
  "Hello" → "Khoor"
  "World" → "Zruog"
  "Secret" → "Vhfuhw"
  "Message" → "Phvvdjh"
  "Test" → "Whvw"

 Результаты совпадают:
   Последовательный == Параллельный: True
   Последовательный == PLINQ: True

=== Готово ===
```

Порядок выполнения в Parallel.ForEach может отличаться, но результаты всегда совпадают.

4.2. Полный листинг программы с комментариями

```
using System;
using System.Linq;
using System.Threading;
using System.Threading.Tasks;
```

```
// Лабораторная работа №9. Параллельные циклы и LINQ-запросы
// Вариант 9

class Program
{
    // Метод шифрования строки
    static string EncryptString(string input, int shift)
    {
        if (input == null) return null;
        Thread.Sleep(500); // Имитация долгой операции

        char[] buffer = new char[input.Length];
        for (int i = 0; i < input.Length; i++)
        {
            buffer[i] = (char)(input[i] + shift);
        }
        return new string(buffer);
    }

    static void Main()
    {
        Console.WriteLine("=== Лабораторная работа №9. Вариант 9 ===");
        Console.WriteLine("Параллельные циклы и LINQ-запросы\n");

        // Тестовые данные: массив строк
        string[] inputStrings = { "Hello", "World", "Secret", "Message", "Test" };
        int shiftValue = 3;

        // === 1. Последовательная обработка ===
        Console.WriteLine("1. Последовательная обработка:");
        var sequentialResults = new string[inputStrings.Length];
        for (int i = 0; i < inputStrings.Length; i++)
        {
            sequentialResults[i] = EncryptString(inputStrings[i], shiftValue);
            Console.WriteLine($"  \"{inputStrings[i]}\" → \"{sequentialResults[i]}\"");
        }

        // === 2. Параллельный цикл ForEach ===
        Console.WriteLine("\n2. Параллельный цикл Parallel.ForEach:");
        var parallelResults = new string[inputStrings.Length];
        object lockObj = new object(); // для синхронизации вывода

        Parallel.ForEach(inputStrings.Select((s, i) => new { Index = i, Value = s }), item
=>
        {
            string encrypted = EncryptString(item.Value, shiftValue);
            lock (lockObj)
            {
                parallelResults[item.Index] = encrypted;
                Console.WriteLine($"  \"{item.Value}\" → \"{encrypted}\" (Поток
{Task.CurrentId})");
            }
```

```
        });

        // === 3. PLINQ (параллельный LINQ) ===
        Console.WriteLine("\n3. PLINQ (AsParallel):");
        var plinqResults = inputStrings
            .AsParallel()
            .AsOrdered() // сохраняем порядок
            .Select(s => EncryptString(s, shiftValue))
            .ToArray();

        for (int i = 0; i < inputStrings.Length; i++)
        {
            Console.WriteLine($"  \"{inputStrings[i]}\" → \"{plinqResults[i]}\"");
        }

        // === Проверка корректности ===
        bool seqParEqual = sequentialResults.SequenceEqual(parallelResults);
        bool seqPlinqEqual = sequentialResults.SequenceEqual(plinqResults);

        Console.WriteLine($"\n Результаты совпадают:");
        Console.WriteLine($"   Последовательный == Параллельный: {seqParEqual}");
        Console.WriteLine($"   Последовательный == PLINQ: {seqPlinqEqual}");

        Console.WriteLine("\n=== Готово ===");
    }
}
```

5. Вывод

В ходе выполнения лабораторной работы №9 были:

- Реализован метод шифрования строки согласно варианту 9;
- Продемонстрированы три подхода: последовательная обработка, Parallel.ForEach, PLINQ;
- Обеспечена синхронизация вывода в параллельном цикле;
- Сохранён порядок элементов в PLINQ;
- Подтверждена корректность всех подходов через сравнение результатов.