Министерство образования Республики Беларусь

Учреждение образования

«Белорусский государственный университет

информатики и радиоэлектроники»

Кафедра инженерной психологии и эргономики

­­­­­

Конструирование программного обеспечения

Отчет по лабораторной работе №10

«Разработка, отладка и испытание приложения параллельной обработки данных, применение мьютексов и семафоров»

Выполнил: Усов А.М.

Студент группы 310901

Преподаватель: Давыдович К. И.

Минск 2024

Цель: : Сформировать умения и навыки разработки программ с использованием классов Task и Parallel.

Задание: Используя TPL создайте длительную по времени задачу (на основе Task) на выбор:

✓поиск простых чисел (желательно взять «решето Эратосфена»),

✓перемножение матриц,

✓умножение вектора размера 100000 на число,

✓создание множества Мандельброта

✓или другой алгоритм.

1)Выведите идентификатор текущей задачи, проверьте во время выполнения – завершена ли задача и выведите ее статус.

2)Оцените производительность выполнения используя объект Stopwatch на нескольких прогонах.

*Дополнительно:*

*Для сравнения реализуйте последовательный алгоритм.*

2.Реализуйте второй вариант этой же задачи с токеном отмены CancellationToken и отмените задачу.

3.Создайте три задачи с возвратом результата и используйте их для выполнения четвертой задачи. Например, расчет по формуле.

4.Создайте задачу продолжения (continuation task) в двух вариантах:

1)C ContinueWith - планировка на основе завершения множества предшествующих задач

2) На основе объекта ожидания и методов GetAwaiter(),GetResult();

5.Используя Класс Parallel распараллельте вычисления циклов For(), ForEach(). Например, обработку (преобразования)последовательности, генерация нескольких массивов по 1000000 элементов, быстрая сортировка последователности, обработка текстов. Оцените производительность по сравнению с обычными циклами

6.Используя Parallel.Invoke() распараллельте выполнение блока операторов.

7.Используя Класс BlockingCollection реализуйте следующую задачу:

Есть 5 поставщиков бытовой техники, они завозят уникальные товары на склад (каждый по одному) и 10 покупателей – покупают все подряд, если товара нет - уходят. В вашей задаче: cпрос превышает предложение. Изначально склад пустой. У каждого поставщика своя скорость завоза товара. Каждый раз при изменении состоянии склада выводите наименования товаров на складе.

8.Используя async и await организуйте асинхронное выполнение метода.

Код Программы:

using System.Collections.Concurrent;

using System.Diagnostics;

namespace LT10;

public class Program

{

public static void FindPrimes(int n,CancellationToken cancellationToken)

{

var primes = new bool[n + 1];

for (int i = 0; i <= n; i++) primes[i] = true;

primes[0] = primes[1] = false;

for (int p = 2; p \* p <= n; p++)

{

if (cancellationToken.IsCancellationRequested) return;

if (primes[p])

{

for (int i = p \* p; i <= n; i += p)

primes[i] = false;

}

}

for (int i = 2; i <= n; i++)

{

if (cancellationToken.IsCancellationRequested) return;

if (primes[i]) Console.WriteLine(i);

}

}

public static async Task Main()

{

var cts = new CancellationTokenSource();

var token = cts.Token;

var task = Task.Run(() => FindPrimes(1000000, token));

cts.Cancel();

Console.WriteLine($"Task ID: {task.Id}");

Console.WriteLine($"Task Status: {task.Status}");

var stopwatch = Stopwatch.StartNew();

try

{

task.Wait();

}

catch (AggregateException ex)

{

ex.Handle(e => e is TaskCanceledException);

Console.WriteLine("Task canceled");

}

stopwatch.Stop();

Console.WriteLine($"Execution Time: {stopwatch.ElapsedMilliseconds} ms");

var task1 = Task.Run(() => 5);

var task2 = Task.Run(() => 10);

var task3 = Task.Run(() => 15);

var task4 = Task.WhenAll(task1, task2, task3).ContinueWith(t =>

{

int sum = t.Result.Sum();

Console.WriteLine($"Sum: {sum}");

});

task4.Wait();

var task5 = Task.Run(() => 42);

var awaiter = task5.GetAwaiter();

awaiter.OnCompleted(() =>

{

var result = awaiter.GetResult();

Console.WriteLine($"Result: {result}");

});

task5.Wait();

var arr1 = new List<int>(10000000);

var arr2 = new List<int>(10000000);

var task6 = Task.Run(() =>

{

for (int i = 0; i < 10000000; i++)

{

arr1.Add(i);

arr2.Add(i);

}

});

task6.Wait();

var stopwatch1 = Stopwatch.StartNew();

Parallel.For(0, 10000000, i =>

{

arr1[i] = arr1[i] \* 2731;

});

stopwatch1.Stop();

Console.WriteLine($"Parallel Execution Time: {stopwatch1.ElapsedMilliseconds} ms");

var stopwatch2 = Stopwatch.StartNew();

for (int i = 0; i < 10000000; i++)

{

arr2[i] = arr2[i] \* 2731;

}

stopwatch2.Stop();

Console.WriteLine($"Sequential Execution Time: {stopwatch2.ElapsedMilliseconds} ms");

Parallel.Invoke(

() => Console.WriteLine("Task 1"),

() => Console.WriteLine("Task 2"),

() => Console.WriteLine("Task 3")

);

int result = await CalculateAsync();

Console.WriteLine($"Result: {result}");

var stock = new BlockingCollection<string>();

Task.Run(() =>

{

while (true)

{

if (stock.TryTake(out var item)) Console.WriteLine($"Customer bought: {item}");

}

});

var suppliers = new Task[5];

for (int i = 0; i < 5; i++)

{

suppliers[i] = Task.Run(() =>

{

while (true)

{

var item = $"Item {Guid.NewGuid()}";

stock.Add(item);

Console.WriteLine($"Supplier added: {item}");

Thread.Sleep(300\*i);

}

});

}

Task.WaitAll(suppliers);

}

public static async Task<int> CalculateAsync()

{

await Task.Delay(1000);

return 42;

}

}

Результат выполнения программы представлен на рисунке 1.

# 

Рисунок 1 – Результат выполнения программы