ЛАБОРАТОРНАЯ РАБОТА 10

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

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

Время выполнения: 4 часа.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

В основе библиотеки TPL лежит концепция задач, каждая из которых описывает отдельную продолжительную операцию. В библиотеке классов .NET задача представлена специальным классом - классом Task, который находится в пространстве имен System.Threading.Tasks. Данный класс описывает отдельную задачу, которая запускается асинхронно в одном из потоков из пула потоков. Хотя ее также можно запускать синхронно в текущем потоке.

Для определения и запуска задачи можно использовать различные способы. Первый способ создание объекта Task и вызов у него метода Start:

```
1	Task task = new Task(() => Console.WriteLine("Hello Task!"));
2	task.Start();
```

В качестве параметра объект Task принимает делегат Action, то есть мы можем передать любое действие, которое соответствует данному делегату, например, лямбда-выражение, как в данном случае, или ссылку на какой-либо метод. То есть в данном случае при выполнении задачи на консоль будет выводиться строка "Hello Task!".

А метод Start() собственно запускает задачу.

Второй способ заключается в использовании статического метода Task.Factory.StartNew(). Этот метод также в качестве параметра принимает делегат Action, который указывает, какое действие будет выполняться. При этом этот метод сразу же запускает задачу:

```
Task task = Task.Factory.StartNew(() => Console.WriteLine("Hello Task!"));
```

В качестве результата метод возвращает запущенную задачу.

Третий способ определения и запуска задач представляет использование статического метода Task.Run():

```
Task task = Task.Run(() => Console.WriteLine("Hello Task!"));
```

Метод Task.Run() также в качестве параметра может принимать делегат Action - выполняемое действие и возвращает объект Task.

Определим небольшую программу, где используем все эти способы:

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

```
namespace HelloApp
{
   class Program
   {
      static void Main(string[] args)
      {
         Task task1 = new Task(() => Console.WriteLine("Task1 is executed"));
         task1.Start();

         Task task2 = Task.Factory.StartNew(() => Console.WriteLine("Task2 is executed"));

         Task task3 = Task.Run(() => Console.WriteLine("Task3 is executed"));

         Console.ReadLine();
      }
   }
}
```

Важно понимать, что задачи не выполняются последовательно. Первая запущенная задача может завершить свое выполнение после последней задачи.

Или рассмотрим еще один пример:

```
using System;
using System.Threading;

namespace TaskApp
{
   class Program
   {
      static void Main(string[] args)
      {
         Task task = new Task(Display);
         task.Start();

         Console.WriteLine("Завершение метода Main");

         Console.ReadLine();
      }

      static void Display()
      {
         Console.WriteLine("Начало работы метода Display");

         Console.WriteLine("Завершение работы метода Display");
      }
   }
}
```

Класс Task в качестве параметра принимает метод Display, который соответствует делегату Action. Далее чтобы запустить задачу, вызываем

метод Start: task.Start(), и после этого метод Display начнет выполняться во вторичном потоке. В конце метода Main выводит некоторый маркер-строку, что метод Main завершился.

Однако в данном случае консольный вывод может выглядеть следующим образом:

Завершение метода Main

Начало работы метода Display

Завершение работы метода Display

То есть мы видим, что даже когда основной код в методе Main уже отработал, запущенная ранее задача еще не завершилась.

Чтобы указать, что метод Main должен подождать до конца выполнения задачи, нам надо использовать метод Wait:

```
static void Main(string[] args)
{
   Task task = new Task(Display);
   task.Start();
   task.Wait();
   Console.WriteLine("Завершение метода Main");
   Console.ReadLine();
}
```

Свойства класса Task

Класс Task имеет ряд свойств, с помощью которых мы можем получить информацию об объекте. Некоторые из них:

AsyncState: возвращает объект состояния задачи

CurrentId: возвращает идентификатор текущей задачи

Exception: возвращает объект исключения, возникшего при выполнении задачи

Status: возвращает статус задачи.

Вложенные задачи

Одна задача может запускать другую - вложенную задачу. При этом эти задачи выполняются независимо друг от друга. Например:

```
static void Main(string[] args)
{
    var outer = Task.Factory.StartNew(() =>      // внешняя задач
    {
        Console.WriteLine("Outer task starting...");
        var inner = Task.Factory.StartNew(() =>  // вложенная зад
        {
            Console.WriteLine("Inner task starting...");
            Thread.Sleep(2000);
            Console.WriteLine("Inner task finished.");
        });
    });
    outer.Wait(); // ожидаем выполнения внешней задачи
    Console.WriteLine("End of Main");
```

```
    Console.ReadLine();}
```

Несмотря на то, что здесь мы ожидаем выполнения внешней задачи, но вложенная задача может завершить выполнение даже после завершения метода Main:

Outer task starting...

End of Main

Inner task starting...

Inner task finished.

Если необходимо, чтобы вложенная задача выполнялась вместе с внешней, В данном случае необходимо использовать значение TaskCreationOptions.AttachedToParent:

```
static void Main(string[] args)
{
    var outer = Task.Factory.StartNew(() =>      // внешняя задач
    {
        Console.WriteLine("Outer task starting...");
        var inner = Task.Factory.StartNew(() =>  // вложенная зад
        {
            Console.WriteLine("Inner task starting...");
            Thread.Sleep(2000);
            Console.WriteLine("Inner task finished.");
        }, TaskCreationOptions.AttachedToParent);
    });
    outer.Wait(); // ожидаем выполнения внешней задачи
    Console.WriteLine("End of Main");

    Console.ReadLine();
}
```

Консольный вывод:

Outer task starting...

Inner task starting...

Inner task finished.

End of Main

Массив задач

Также как и с потоками, мы можем создать и запустить массив задач. Можно определить все задачи в массиве непосредственно через объект Task:

```
Task[] tasks1 = new Task[3]
{
    new Task(() => Console.WriteLine("First Task")),
    new Task(() => Console.WriteLine("Second Task")),
    new Task(() => Console.WriteLine("Third Task"))
};
// запуск задач в массиве
foreach (var t in tasks1)
    t.Start();
```

Либо также можно использовать методы Task.Factory.StartNew или Task.Run и сразу запускать все задачи:

```
Task[] tasks2 = new Task[3];
int j = 1;
for (int i = 0; i < tasks2.Length; i++)
    tasks2[i] = Task.Factory.StartNew(() => Console.WriteLine($"T
```

Но в любом случае мы опять же можем столкнуться с тем, что все задачи из массива могут завершиться после того, как отработает метод Main, в котором запускаются эти задачи:

```
static void Main(string[] args)
{
    Task[] tasks1 = new Task[3]
    {
        new Task(() => Console.WriteLine("First Task")),
        new Task(() => Console.WriteLine("Second Task")),
        new Task(() => Console.WriteLine("Third Task"))
    };
    foreach (var t in tasks1)
        t.Start();

    Task[] tasks2 = new Task[3];
    int j = 1;
    for (int i = 0; i < tasks2.Length; i++)
        tasks2[i] = Task.Factory.StartNew(() => Console.WriteLine

    Console.WriteLine("Завершение метода Main");

    Console.ReadLine();
}
```

Один из возможных консольных выводов программы:

Second Task

Task 1

Завершение метода Main

Third Task

Task 3

First Task

Task 2

Если необходимо выполнять некоторый код лишь после того, как все задачи из массива завершатся, то применяется метод Task.WaitAll(tasks):

```
static void Main(string[] args)
{
    Task[] tasks1 = new Task[3]
```

```
    {
        new Task(() => Console.WriteLine("First Task")),
        new Task(() => Console.WriteLine("Second Task")),
        new Task(() => Console.WriteLine("Third Task"))
    };
    foreach (var t in tasks1)
        t.Start();
    Task.WaitAll(tasks1); // ожидаем завершения задач

    Console.WriteLine("Завершение метода Main");

    Console.ReadLine();
}
```

В этом случае сначала завершатся все задачи, и лишь только потом будет выполняться последующий код из метода Main:

Second Task

Third Task

First Task

Завершение метода Main

В то же время порядок выполнения самих задач в массиве также недетерминирован.

Также мы можем применять метод Task.WaitAny(tasks). Он ждет, пока завершится хотя бы одна из массива задач.

Возвращение результатов из задач

Задачи могут не только выполняться как процедуры, но и возвращать определенные результаты:

```
class Program
{
    static void Main(string[] args)
    {
        Task<int> task1 = new Task<int>(()=>Factorial(5));
        task1.Start();

        Console.WriteLine($"Факториал числа 5 равен {task1.Result

        Task<Book> task2 = new Task<Book>(() =>
        {
            return new Book { Title = "Война и мир", Author = "Л.
        });
        task2.Start();

        Book b = task2.Result;  // ожидаем получение результата
        Console.WriteLine($"Название книги: {b.Title}, автор: {b.
```

```
            Console.ReadLine();
        }


        static int Factorial(int x)
        {
            int result = 1;

            for (int i = 1; i <= x; i++)
            {
                result *= i;
            }

            return result;
        }
    }

    public class Book
    {
        public string Title { get; set; }
        public string Author { get; set; }
    }
```

Во-первых, чтобы задать возвращаемый из задачи тип объекта, мы должны типизировать Task. Например, Task<int> - в данном случае задача будет возвращать объект int.

И, во-вторых, в качестве задачи должен выполняться метод, возвращающий данный тип объекта. Например, в первом случае у нас в качестве задачи выполняется функция Factorial, которая принимает числовой параметр и также на выходе возвращает число.

Возвращаемое число будет храниться в свойстве Result: task1.Result. Нам не надо его приводить к типу int, оно уже само по себе будет представлять число.

То же самое и со второй задачей task2. В этом случае в лямбда-выражении возвращается объект Book. И также мы его получаем с помощью task2.Result

При этом при обращении к свойству Result программа текущий поток останавливает выполнение и ждет, когда будет получен результат из выполняемой задачи.

Индивидуальные задания для лабораторной работы 2:

1.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)С 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 последовательности, 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: спрос 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.

ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1. Изучить теоретическую часть лабораторной работы.

2. Реализовать индивидуальное задание по вариантам, представленные в теоретических сведениях, сделать скриншоты работающих программ. Написать комментарии.

3. Написать отчет, содержащий:

1. Титульный лист, на котором указывается:

а) полное наименование министерства образование и название учебного заведения;

б) название дисциплины;

в) номер практического занятия;

г) фамилия преподавателя, ведущего занятие;
д) фамилия, имя и номер группы студента;
е) год выполнения лабораторной работы.