Лабораторная работа №4

Цель работы

Приобретение навыков разработки многопоточных приложений в среде .Net.

В ходе работы требуется реализовать приложение для мониторинга погодных условий в различных городах России с помощью сайта **www.gismeteo.ru.** При этом загрузку страниц следует организовать параллельно.

Упражнение 1. Создание простого многопоточного приложения

При реализации приложений часто возникают задачи, на выполнение которых может уходить достаточно много времени. Одним из способов ускорения работы программы является применение многопоточности. Обычное C# приложение запускается в единственном потоке, создание, запуск и остановка отдельных потоков, параллельных основному, может быть реализована стандартными средствами платформы .Net.

Загрузка интернет страницы обычно занимает значительное время, поэтому такая задача может и должна запускаться в отдельном потоке. Реализуем приложение, загружающее веб – страницу в отдельном потоке.

```
using System;
using System.Net;
using System.Threading;
using System.Diagnostics;

namespace weather
{
    class Program
    {
        static void Main(string[] args)
        {
            Stopwatch stopwatch = new Stopwatch();
            Thread getWeatherThread = new Thread(GetWebPage);
            getWeatherThread.Start();
            do
            {
                if (getWeatherThread.IsAlive)
                {
                    if (!stopwatch.IsRunning)
                    {
                        stopwatch.Start();
                    }
                }
                else
                {
                    stopwatch.Stop();
                    break;
                }
            } while (true);
            Console.WriteLine("Page downloaded for " + (Convert.ToDouble(stopwatch.ElapsedMilliseconds) / 1000) + " sec.");
            Console.ReadKey();
        }

        static void GetWebPage()
        {
            WebClient client = new WebClient();
            client.DownloadFile(new Uri("https://www.gismeteo.ru/city/daily/4368/"), "Moscow.txt");
        }
    }
}
```

Приведенный выше код реализует пример загрузки страницы в отдельном потоке с измерением времени выполнения. Загруженная страница по - умолчанию создается в директории с исполняемым файлом (**Moscow.txt**).

Для того чтобы создать отдельный поток необходимо инстанцировать класс **Thread** и передать в его конструктор метод, который будет выполняться в отдельном потоке (**GetWebPage).** Для измерения времени выполнения кода используется класс **Stopwatch** пространства имен **System.Diagnostics**.

Упражнение 2. Передача параметров в метод, выполняемый в отдельном потоке

Для реализации логики в метод, запускаемый в отдельном потоке, неизбежно потребуется передать некоторые параметры. Один из способов заключается в создании отдельного класса параметров, его инициализации и последующей передаче в метод с помощью класса **ParameterizedThreadStart.** Удобнее воспользоваться лямбда – выражением. Реализуем многопоточную загрузку веб-страницы с передачей в метод параметров. Также создадим структуру, в которой будут храниться пары значений: город и его идентификатор на сайте, с которого загружаем данные о погоде.

```
namespace weather
{

    public struct Cities
    {
        public static KeyValuePair<string, string> Moscow = new KeyValuePair<string, string>("Moscow", "4368");
        public static KeyValuePair<string, string> StPetersburg = new KeyValuePair<string, string>("StPetersburg", "4079");
        public static KeyValuePair<string, string> Novosibirsk = new KeyValuePair<string, string>("Novosibirsk", "4690");
        public static KeyValuePair<string, string> Ekaterinburg = new KeyValuePair<string, string>("Ekaterinburg", "4517");
        public static KeyValuePair<string, string> NNovgorod = new KeyValuePair<string, string>("NNovgorod", "4355");
        public static KeyValuePair<string, string> Kazan = new KeyValuePair<string, string>("Kazan", "4364");
        public static KeyValuePair<string, string> Chelyabinsk = new KeyValuePair<string, string>("Chelyabinsk", "4565");
        public static KeyValuePair<string, string> Omsk = new KeyValuePair<string, string>("Omsk", "4578");
        public static KeyValuePair<string, string> Samara = new KeyValuePair<string, string>("Samara", "4618");
        public static KeyValuePair<string, string> RostovNaDonu = new KeyValuePair<string, string>("RostovNaDonu", "5110");
    }
    class Program
    {
        static void Main(string[] args)
        {
            string uri = String.Format("https://www.gismeteo.ru/city/daily/{0}/", Cities.Moscow.Value);

            Stopwatch stopwatch = new Stopwatch();
            Thread getWeatherThread = new Thread(() => GetWebPage(uri, Cities.Moscow.Key));
            getWeatherThread.Start();
            do
            {
                if (getWeatherThread.IsAlive)
                {
                    if (!stopwatch.IsRunning)
                    {
                        stopwatch.Start();
                    }
                }
                else
                {
                    stopwatch.Stop();
                    break;
                }
            } while (true);
            Console.WriteLine("Page downloaded for " + (Convert.ToDouble(stopwatch.ElapsedMilliseconds) / 1000) + " sec.");
            Console.ReadKey();
        }

        static void GetWebPage(string uri, string localFileName)
        {
            WebClient client = new WebClient();
            client.DownloadFile(new Uri(uri), localFileName + ".txt");
        }
    }
}
```

Не забудьте подключить пространство имен **System.Collections.Generic.**

Упражнение 3. Работа с пулом потоков с помощью Task Parallel Library

Пул потоков — это коллекция потоков, которые могут использоваться для выполнения нескольких задач в фоновом режиме. Это позволяет разгрузить главный поток для асинхронного выполнения других задач.

Пулы потоков часто используются в серверных приложениях. Каждый входящий запрос назначается потоку из пула, таким образом, запрос может обрабатываться асинхронно без задействования главного потока и задержки обработки последующих запросов.

Когда поток в пуле завершает выполнение задачи, он возвращается в очередь ожидания, в которой может быть повторно использован. Повторное использование позволяет приложениям избежать дополнительных затрат на создание новых потоков для каждой задачи.

Обычно пулы имеют максимальное количество потоков. Если все потоки заняты, дополнительные задачи помещаются в очередь, где хранятся до тех пор, пока не появятся свободные потоки.

Одним из основных средств, внедренных в версию 4.0 среды .NET Framework, является **библиотека распараллеливания задач (TPL)**. Эта библиотека усовершенствует многопоточное программирование двумя основными способами. Во-первых, она упрощает создание и применение многих потоков. Во-вторых, она позволяет автоматически использовать несколько процессоров. Библиотека TPL определена в пространстве имен **System.Threading.Tasks**

Задействуем в нашем приложении метод класса **Task**. Также реализуем дополнительные методы для получения данных с сайта погоды. В результате получим приложение, выводящее в реальном времени данные по температуре в данный момент в 10 городах России. Для получения данных с загруженной страницы воспользуемся библиотекой **HtmlAgilityPack**, ссылку на которую необходимо подключить к проекту в **References**.

```
using System;
using System.Collections.Generic;
using System.Net;
using System.Threading;
using System.Threading.Tasks;
using HtmlAgilityPack;

namespace weather
{
    class Program
    {
        public static Dictionary<string, int> CitiesDictionary = new Dictionary<string, int>()
        {
            {"Moscow", 4368},
            {"StPetersburg", 4079},
            {"Novosibirsk", 4690},
            {"Ekaterinburg", 4517},
            {"NNovgorod", 4355},
            {"Kazan", 4364},
            {"Chelyabinsk", 4565},
            {"Omsk", 4578},
            {"Samara", 4618},
            {"RostovNaDonu", 5110}
        };
        static void Main(string[] args)
        {
            do
            {
                foreach (var c in CitiesDictionary)
                {
                    string city = c.Key;
                    int cityId = c.Value;

                    Task.Factory.StartNew(() => PrintCityTemp(city, cityId));
                }
                Thread.Sleep(5000);
                Console.Clear();
            } while (true);
        }

        static void PrintCityTemp(string city, int cityId)
        {
            double cityTemp = 0;
            string fileName = city + ".xml";

            try
            {
                string weatherUri = GetCityTempUri(cityId);
                GetWebPage(weatherUri, fileName);
                cityTemp = ParseWebPage(fileName);
            }
            catch
            {
                Console.WriteLine("Error while processing "+city);
                return;
            }

            Console.WriteLine(cityTemp + " in \t" + city);
        }

        static string GetCityTempUri(int cityId)
        {
            return String.Format("https://www.gismeteo.ru/city/daily/{0}/", cityId);
        }

        static void GetWebPage(string uri, string localFileName)
        {
            WebClient client = new WebClient();
            client.DownloadFile(new Uri(uri), localFileName);
        }

        static double ParseWebPage(string file)
        {
            double temp = 0;
            HtmlDocument htmlDoc = new HtmlDocument();

            htmlDoc.Load(file);

            if (htmlDoc.DocumentNode != null)
            {
                HtmlNode bodyNode = htmlDoc.DocumentNode.SelectSingleNode("//*[@id=\"weather\"]//div[3]//div[2]//dd[1]//text()");

                if (bodyNode != null)
                {
                    string tempString = bodyNode.InnerText;
                    tempString = tempString.Replace("&minus;", "-");
                    temp = Convert.ToDouble(tempString);
                }
            }

            return temp;
        }
```

Упражнение 4. Блокировка общих ресурсов в многопоточном приложении

При реализации многопоточного приложения возникает необходимость работы с общими для нескольких потоков ресурсами. В таком случае необходимо явно реализовать механизм доступа к такому ресурсу для каждого потока. Наиболее популярными подходами являются создание блокирующего объекта (lock), мьютекс и семафор.

Реализуем файловый вывод данных из различных потоков и продемонстрируем работу блокирующего объекта.

В первую очередь добавим метод для текстового вывода, добьемся возникновения исключения путем добавления искусственной задержки в вывод.

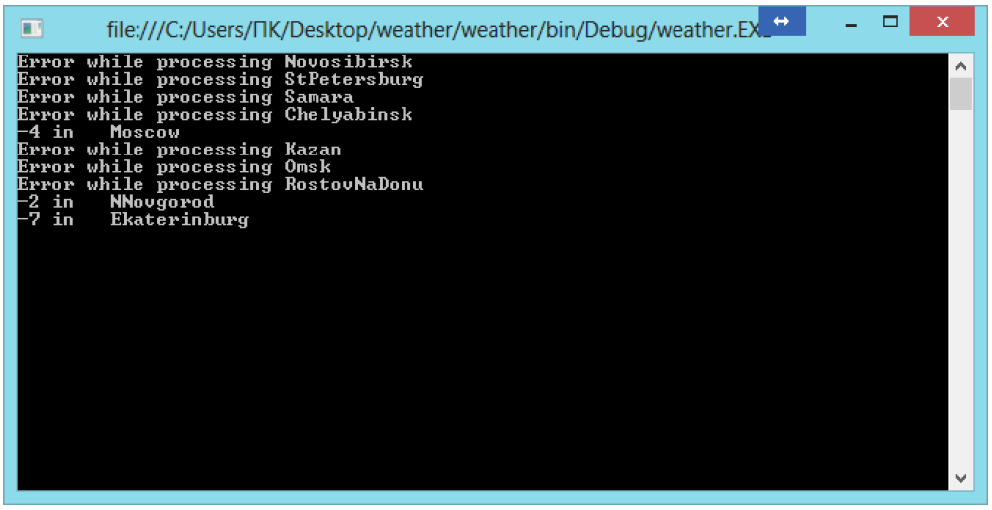
```
1 reference
static void FileOutPut(string outPut)
{

        using (FileStream fs = new FileStream("Output.txt", FileMode.OpenOrCreate))
        {
            fs.Position = fs.Length;
            using (StreamWriter sw = new StreamWriter(fs))
            {
                sw.WriteLine(outPut);
                Thread.Sleep(500);

            }
        }

}
```

```
1 reference
static void PrintCityTemp(string city, int cityId)
{
    double cityTemp = 0;
    string fileName = city + ".xml";

    try
    {
        string weatherUri = GetCityTempUri(cityId);
        GetWebPage(weatherUri, fileName);
        cityTemp = ParseWebPage(fileName);

        FileOutPut(cityTemp + " in \t" + city);
    }
    catch
    {
        Console.WriteLine("Error while processing "+city);
        return;
    }

    Console.WriteLine(cityTemp + " in \t" + city);
}
```

```
static void Main(string[] args)
{
    do
    {
        foreach (var c in CitiesDictionary)
        {
            string city = c.Key;
            int cityId = c.Value;

            Task.Factory.StartNew(() => PrintCityTemp(city, cityId));
        }
        Thread.Sleep(5000);
        Console.Clear();
        FileDelete();
    } while (true);
}
```

```
1 reference
static void FileDelete()
{
        File.Delete("Output.txt");
}
```

Убедимся в возникновении исключения при попытке потока обратиться к файлу, который занят другим потоком.



Теперь добавим оператор **lock** в код файлового вывода, убедимся, что программа работает корректно.

```
class Program
{
    static object locker = new object();
```

```
1 reference
static void FileOutPut(string outPut)
{
    lock (locker)
    {
        using (FileStream fs = new FileStream("Output.txt", FileMode.OpenOrCreate))
        {
            fs.Position = fs.Length;
            using (StreamWriter sw = new StreamWriter(fs))
            {
                sw.WriteLine(outPut);
                Thread.Sleep(500);

            }
        }
    }
}
```

Не забудьте так же изменить код удаления старого файла вывода. Убедитесь, что программа работает без ошибок доступа к файловой системе.

Упражнение 5. Разработка многопоточного приложения загрузки и вывода котировок ценных бумаг в реальном времени.

Самостоятельно реализуйте приложение, получающее текущую стоимость следующего далее списка ценных бумаг. При этом загрузка каждой интернет – страницы должна производиться в отдельном потоке. Реализуйте файловый вывод полученных данных.

1.Ссылка для загрузки страницы с текущей ценой: http://finance.yahoo.com/q?s=AAPL&fr=uh3_finance_web&uhb=uhb2

где **AAPL –** название инструмента.

2.Путь **XPath** до необходимого значения на странице:

```
HtmlNode bodyNode = htmlDoc.DocumentNode.SelectSingleNode("//*[@id=\"yfs_l84_aapl\"]");
```

где в подстроке **yfs_184_aapl**, **aapl –** название инструмента в нижнем регистре.

3.Список инструментов:

FST, ACI, SPY, AAPL, IWM, FB, TWTR, EEM, QQQ, MSFT, GOOG.