Многонишково програмиране в C

Съвременните приложения - настолни и мобилни са проектирани така, че да изпълняват различни задачи асинхронно. Това означава, че резултатът от изпълнението на задачите няма да се визуализира веднага след генериране на събитие от страна на потребителя. Такива задачи обикновено изискват повече време, за да се изпълнят, и към тях може да отнесем следните видове - достъп до отдалечен ресурс - изчакване на резултат от сървър, достъп до файлове, обработка на изображения и др. При всеки от изброените примери времето за получаване на резултат може да отнеме известно количество време (до няколко секунди).

Ако такъв вид достъп се имплементира в последователен режим в дадена програма, това би създало впечатлението, че създадената програма е "забила" и нищо не изпълнява.

Преминаване от еднонишково програмиране в многонишково програмиране

Поведение на еднонишкова програма при продължително изпълнение на метод

- Създайте нов WPF проект във Visual Studio. Кръстете го `AsyncTasks`.
- Ново създадения проект в MainWindow.xaml добавете `Button` и `TextBlock`. Кръстете Button с име `btnStartCount` и съдържание (content) `Start counting`; За текстовото поле задайте име `txtCountResult`.
- В MainWindow.xaml.cs създайте частен член-метод, който да генерира числата числата от 0 до 100 и да ги задава като текст на `txtCountResult`.
- В тялото на цикъла добавете:

```
Thread.Sleep(100); //Внася забавяне от 100ms при всяка итерация на цикъла
(преброяване от 0 до 100 ще отнеме около 10s)
```

- Добавете и namespace-а към cs файла:

```
using System.Threading;
```

- Създайте Click метод за бутона и извикайте метода за генериране на числата.
- Запишете, компилирайте и стартирайте проекта.

Това, което ще наблюдавате като резултат от изпълнението на програмата, е, че ще забележите след натискане на бутона, програмата вече не реагира на потребителски вход, а резултатът от изпълнението на метода не се актуализира. Причината за това "странно" поведение на програмата се крие в това, че целият програмен код се изпълнява от главната нишка на програмата. Както знаем от Първо и Второ упражнение по ПС, главната нишка в процеса се занимава с обновяването на потребителския интерфейс. Следователно, докато не се приключи действието на метода за броене, няма да се актуализира и нищо по потребителския интерфейс.

Този пример демонстрира поведението на програмата, ако тя работи само с една нишка - главната нишка.

Въвеждане на многонишковост в приложението

В C# съществуват няколко начина за разпаралеляване на програмния код и изпълнението от няколко нишки

Използване на нишки

Нишките се създават чрез създаване на обекти от класа `Thread` и поставяне на името на метода, който ще се изпълнява във функцията. Стартирането на нишката се става, чрез извикване на метода Start на обекта.

```
Thread thread = new Thread(Counter);
thread.Start();
```

- Променете съдържанието на Click метода на бутона да създава и стартира нишката по оказания по-горе пример.
- Компилирайте проекта и го стартирайте.

При натискане върху бутона в средата ще се появи exception с текст `An unhandled exception of type 'System.InvalidOperationException' occurred in WindowsBase.dll`. За да проверите причината за възникването на тази изключителна ситуация, оградете цикъла в метода за броене с `try/catch` клаузи и повторете изпълнението на програмата.

Методът за броене трябва да наподобява този пример:

```
private void Counter()
{
   try
   {
       for (int i = 0; i <= 100; i++)
       {
           txtCounter.Text = i.ToString();
           Thread.Sleep(100);
       }
   }
   catch (Exception e)
   {
       MessageBox.Show(e.Message, "Exception", MessageBoxButton.OK,
MessageBoxImage.Error);
   }
}
```

- Компилирайте и стартирайте проекта.

При натискане на бутона ще се появи следното съобщение за грешка: `The calling thread cannot access this object because a different thread owns it.` Това показва, че обновяването на информацията на графичните обекти от потребителския интерфейс може да се осъществи ЕДИНСТВЕНО през главната

нишка. Следователно за обновяване на състоянието ще трябва да се извърши чрез специална заявка към главната нишка.

Свойството Dispatcher

В текущия прозорец съществува свойство, към което могат да се подават заявки за опресняване на потребителския интерфейс. Тези заявки се изпълняват от главната нишка по възможност.

Заявката се извършва чрез извикване на метода Invoke и подаване на Action, който да бъде изпълнен. Едно такова действие може да бъде зададено и по следния модел с lambda израз:

```
Dispatcher.Invoke((Action)(() =>
{
    // Тяло, в което се записва кода, който да се изпълни от главната нишка
при възможност.
}));
```

- Заградете в кода реда, който отговаря за промяната на състоянието на текстовото поле, с кода за извикването на Dispatcher-а.

```
Dispatcher.Invoke((Action)(() =>
{
    txtCounter.Text = i.ToString();
}));
```

- Компилирайте и стартирайте приложението.

Вече работата на приложението трябва да е нормална.

Работа с ThreadPool

ThreadPool е клас, който дава възможност за асинхронното изпълнение на задачи като се грижи за управлението на стартираните нишки. Той може да се използва, когато не се налага създаването на методи за изпълнение, както и изричното създаване на нишка.

Пример за такова изпълнение в ThreadPool може да се приложи и в текущия пример:

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    ThreadPool.QueueUserWorkItem(o =>
    {
        try
        {
            for (int i = 0; i <= 100; i++)
            {
              Dispatcher.Invoke((Action)(() =>
              {
                txtCounter.Text = i.ToString();
              }));
              Thread.Sleep(100);
```

```
            }
        }
        catch (Exception ex)
        {
           MessageBox.Show(ex.Message, "Exception", MessageBoxButton.OK,
MessageBoxImage.Error);
        }
    });
}
```

- Приложете горния пример за съдържание на Click метода на бутона, компилирайте и стартирайте програмата.

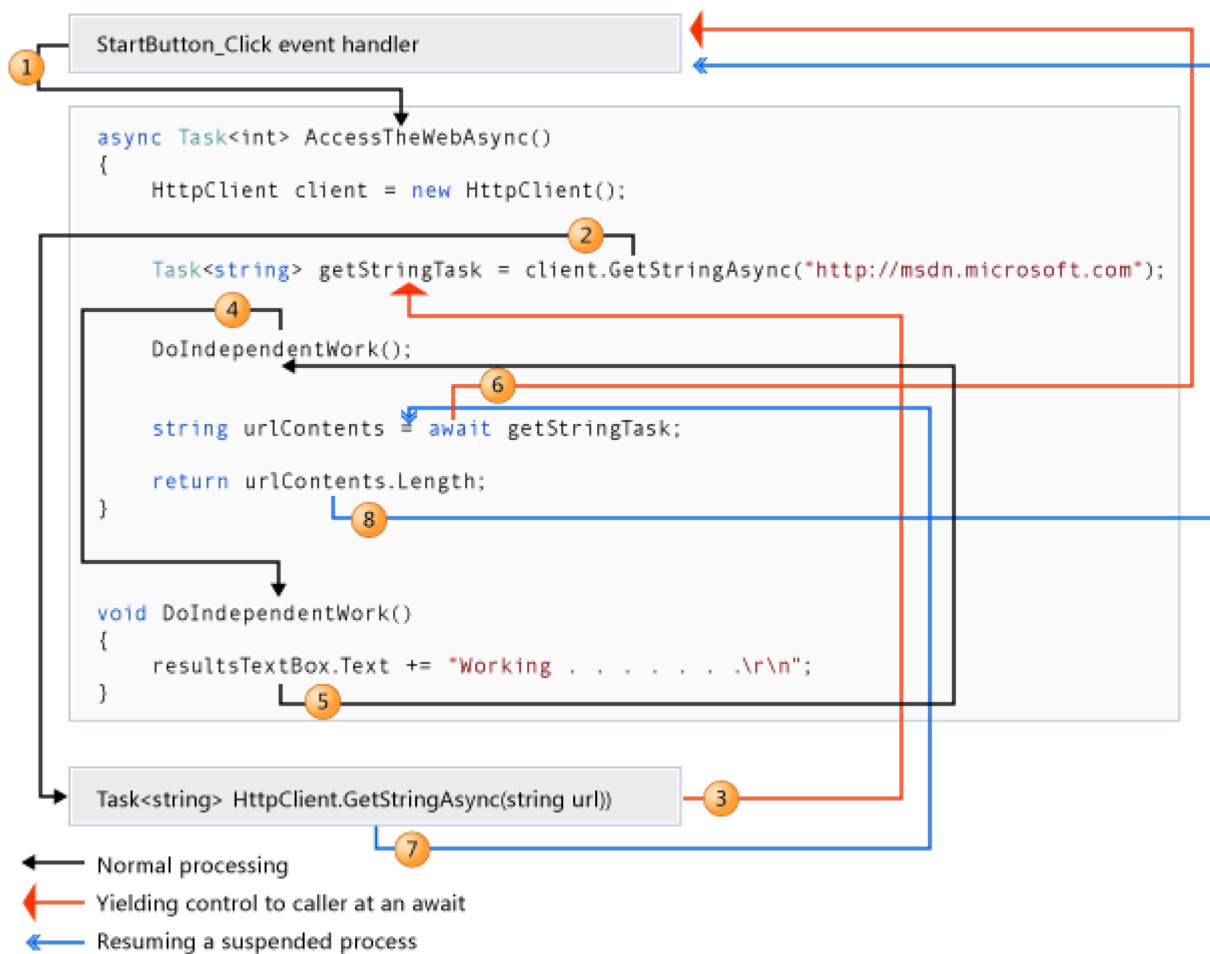
ASYNC и AWAIT ключови думи

Ключовите думи async и await в C# са сърцето на асинхронното програмиране. Чрез използването на тези ключови думи, могат да се изпозват ресурсите на .NET Framework или Windows Runtime, за да се създадат асинхронни методи с приблизителната сложност на синхронните методи. Асинхронните методи се дефинират с async, а те се извикват с await. Пример от Asynchronous Programming with Async and Await

Един асинхронен метод се отличава от синхронния по следните характеристики:

- Притежава `async` модификатор пред името дефиницията си;
- Името на метода завършва с Async суфикс по конвенция;
- Връща като резултат някое от следните:
  - `Task<TResult>` - ако методът връща състояние, където операндът има тип `TResult`.
  - `Task` - ако методът не връща резултат.
  - `void` - Ако се създава асинхронен event handler.
- Методът обикновено включва поне едно `await` извикване, което маркира точката, през която не може да се продължи докато не се изчака готов резултат. Тогава, методът се прекъсва и изчаква асинхронната операция да приключи.

Какво се случва в async метод?



1. Влиза се в асинхронния метод `AccessTheWebAsync`;
2. Стартира се асинхронната задача;
3. Връщаме се в изпълнението на метод, докато се изпълнява асинхронната задача;
4. Изпълнява се метод, който не е свързан с резултата от задачата;
5. Изпълнение на съдържанието на метода;
6. Момент, при който се налага получаването на резултата, изпълнението на текущия метод спира, докато не се получат резултатите. В същото време в точката от която сме влезнали в `AccessTheWebAsync` си продължава работата (докато не се стигне до момент, където резултата от изпълнението на `AccessTheWebAsync` е необходим;
7. Получава на резултат от заявката;
8. Продължение на кода, където се блокирала програмата до получаването на резултата от `AccessTheWebAsync`.

- Добавете `Button` и `TextBlock`. Кръстете Button с име `btnGetWeb` и съдържание (content) `Access Web`; За текстовото поле задайте име `txtCountResult`.
- Генерирайте Click събитие за бутона и довавете следния код:

```
        private async void Button_Click_1(object sender, RoutedEventArgs e)
        {
            int contentLength = await AccessTheWebAsync();
            txtWebStatus.Text += String.Format("\r\nLength of the
downloaded string: {0}.\r\n", contentLength);
        }

        async Task<int> AccessTheWebAsync()
        {
            HttpClient client = new HttpClient();
            Task<string> getStringTask =
client.GetStringAsync("http://fksu2505.tu-sofia.bg");
            DoIndependentWork();
            string urlContents = await getStringTask;
            return urlContents.Length;
        }


        void DoIndependentWork()
        {
            txtWebStatus.Text += "Working . . . . . . .\r\n";
        }
```

- Добавете към `References` на прокета пакета `System.Net.Http`;
- Добавете към MainVindow.xaml.cs

```
using System.Net.Http;
```

- Компилирайте проекта и стартирайте приложението.

Ще забележите, че при натискане на новия бутон в текстовото поле се добавя текст `Working . . . . .` то е резултат от изпълнението на независимия от ресурса код и може да послужи на потребителите да ги информира, че в момента програмата работи и чаква отдалечения ресурс.

Асинхронни задачи в MVVM

- Създайте нов проект в рамките на текущото решение.
- Добавете към проекта нов клас с име `RelayCommand` (от упр.3).
- Поставете това като съдържание на класа:

```
public class RelayCommand : ICommand
    {
        private Action<object> execute;
        private Predicate<object> canExecute;
        private event EventHandler CanExecuteChangedInternal;
        public RelayCommand(Action<object> execute)
            : this(execute, DefaultCanExecute)
        {
        }
        public RelayCommand(Action<object> execute, Predicate<object>
canExecute)
        {
            if (execute == null)
            {
                throw new ArgumentNullException("execute");
            }
            if (canExecute == null)
            {
                throw new ArgumentNullException("canExecute");
            }
            this.execute = execute;
            this.canExecute = canExecute;
        }
        public event EventHandler CanExecuteChanged
        {
            add
            {
                CommandManager.RequerySuggested += value;
                this.CanExecuteChangedInternal += value;
            }
            remove
            {
                CommandManager.RequerySuggested -= value;
                this.CanExecuteChangedInternal -= value;
            }
        }
        public bool CanExecute(object parameter)
        {
            return this.canExecute != null && this.canExecute(parameter);
        }
        public void Execute(object parameter)
        {
            this.execute(parameter);
        }
        public void OnCanExecuteChanged()
        {
            EventHandler handler = this.CanExecuteChangedInternal;
            if (handler != null)
            {
                //DispatcherHelper.BeginInvokeOnUIThread(() =>
handler.Invoke(this, EventArgs.Empty));
                handler.Invoke(this, EventArgs.Empty);
            }
```

```
        }
        public void Destroy()
        {
            this.canExecute = _ => false;
            this.execute = _ => { return; };
        }

        private static bool DefaultCanExecute(object parameter)
        {
            return true;
        }
    }
```

- Добавете към проекта клас `ViewModel`.
- Добавете във `ViewModel` кода за генериране на събитие за `PropertyChanged`

```
        public event PropertyChangedEventHandler PropertyChanged;
        private void NotifyPropertyChanged([CallerMemberName]String 
propName = "")
        {
            if (PropertyChanged != null)
            {
                PropertyChanged(this, new 
PropertyChangedEventArgs(propName));
            }
        }
```

- Добавете имплементирането на интерфейс `INotifyPropertyChanged` към класа `ViewModel`
- Създайте свойство `CurrentTime` от тип `string`, което да генерира събитие при промяна
- Създайте свойства за команди с имена `StartTimeCommand` и `StopTimeCommand`. За тези свойства е необходимо само да ги дефинирате с `{get; set;}`
- Добавете поле от тип `Stopwatch` с име `stopWatch` към класа `ViewModel`.

```
private Stopwatch stopWatch;
```

- Добавете

```
using System.Diagnostics;
```

- Добавете следните 2 метода към класа `ViewModel`

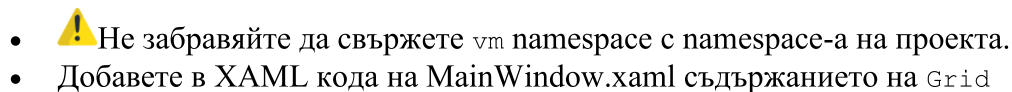
```
        private void StopTimer(object obj)
        {
            stopWatch.Stop();
        }

        private void StartTimer(object obj)
        {
            stopWatch.Restart();
            ThreadPool.QueueUserWorkItem(o =>
            {
                while (stopWatch.IsRunning)
                {
                    TimeSpan ts = stopWatch.Elapsed;
```

```
                    CurrentTime = 
String.Format("{0:00}:{1:00}:{2:00}.{3:00}",
                         ts.Hours, ts.Minutes, ts.Seconds,
                         ts.Milliseconds / 10);
                }
            });
        }
```

- Създайте конструктор на класа `ViewModel`, където:
  - задайте първоначална стойност на `CurrentTime` да е `String.Empty`
  - `stopWatch = new Stopwatch();`
  - свържете `StartTimeCommand` и `StopTimeCommand` със съответните им методи `StartTimer` и `StopTimer`, като създадете обекти от `RelayCommand`
- Добавете в XAML кода на MainWindow.xaml свързване на `DataContext` с `ViewModel`

```
    <Window.DataContext>
        <vm:ViewModel/>
    </Window.DataContext>
```

- Не забравяйте да свържете `vm` namespace с namespace-а на проекта.
- Добавете в XAML кода на MainWindow.xaml съдържанието на `Grid`

```
    <Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition/>
            <ColumnDefinition/>
        </Grid.ColumnDefinitions>
        <StackPanel VerticalAlignment="Center" Margin="30">
            <Button Content="Start" Command="{Binding 
Path=StartTimeCommand}" Margin="10,10"/>
            <Button Content="Stop" Command="{Binding Path=StopTimeCommand}" 
Margin="10,10"/>

        </StackPanel>
        <StackPanel Grid.Column="1" VerticalAlignment="Center" Margin="30">
            <TextBlock Name="txtCounter" FontSize="30" Margin="10,10" 
HorizontalAlignment="Center" Text="{Binding Path=CurrentTime}"/>

        </StackPanel>
    </Grid>
```

- Компилирайте и стартирайте програмата.