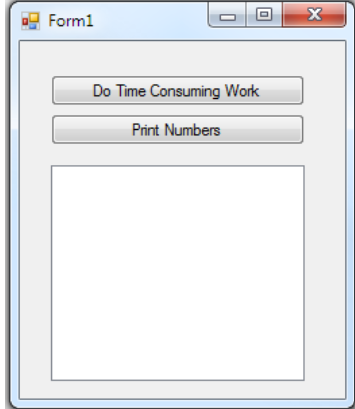
**Part 86 - Multithreading in C#**

**In this video we will discuss**  
1. What is a process and a thread  
2. Simple multithreading example

**Before we discuss multithreading, first let's understand the following terms**  
**1. Process**- Process is what the operating system uses to facilitate the execution of a program by providing the resources required. Each process has a unique process Id associated with it. You can view the process within which a program is being executed using windows task manager.

**2. Thread**- Thread is a light weight process. A process has at least one thread which is commonly called as main thread which actually executes the application code. A single process can have multiple threads.

**Please Note:**All the threading related classes are present in System.Threading namespace.  
  
**Multithreading Example:** Create a new windows forms application with 2 buttons and a listbox control as shown in the image below, and set the following properties.



**For the first button control, set**  
Name = btnTimeConsumingWork  
Text = Do Time Consuming Work  
  
**For the second button control, set**  
Name = btnPrintNumbers  
Text = Print Numbers  
  
Double click on each of the buttons to generate their respective click event handlers.  
  
**For the listbox control, set**  
Name = listBoxNumbers  
  
**Copy and paste the following code in Form1.cs file**

usingSystem**;**

usingSystem.Threading**;**

usingSystem.Windows.Forms**;**

namespaceThreadingExample

**{**

publicpartialclassForm1**:**Form

**{**

publicForm1**()**

**{**

InitializeComponent**();**

**}**

privatevoidbtnTimeConsumingWork\_Click**(**objectsender**,**EventArgse**)**

**{**

btnTimeConsumingWork.Enabled=false**;**

btnPrintNumbers.Enabled=false**;**

DoTimeConsumingWork**();**

btnTimeConsumingWork.Enabled=true**;**

btnPrintNumbers.Enabled=true**;**

**}**

privatevoidDoTimeConsumingWork**()**

**{**

// Make the thread sleep, to introduce artifical latency

Thread.Sleep**(**5000**);**

**}**

privatevoidbtnPrintNumbers\_Click**(**objectsender**,**EventArgse**)**

**{**

for**(**inti=1**;**i<=10**;**i++**)**

**{**

listBoxNumbers.Items.Add**(**i**);**

**}**

**}**

**}**

**}**

**1.**At this point if we run the program, **one thread**is automatically created. This thread is called as the **Main thread**or **UI thread.** This is the thread that is responsible for doing all the work.   
**2.** Now when you click **"Do Time Consuming Work"**, the first 2 lines of code to disable the button is executed. As a result both the buttons are disabled.

**3.** **DoTimeConsumingWork()**method is called next, and at this point the application is unresponsive as it is waiting for the method to complete. Note that the buttons are still disabled and you cannot click on any of them.  
**4.**Finally, once the **DoTimeConsumingWork()**method completes the buttons are enabled and the application is responsive.  
  
Now change the code in **btnTimeConsumingWork\_Click()**event handler method as shown below.

privatevoidbtnTimeConsumingWork\_Click**(**objectsender**,**EventArgse**)**

**{**

btnTimeConsumingWork.Enabled=false**;**

btnPrintNumbers.Enabled=false**;**

// Create another THREAD to offload the work of

// executing the time consuming method to it.

// As a result the UI thread, is free to execute the

// rest of the code and our application is more responsive.

ThreadbackGroundThread=newThread**(**DoTimeConsumingWork**);**

backGroundThread.Start**();**

//DoTimeConsumingWork();

btnTimeConsumingWork.Enabled=true**;**

btnPrintNumbers.Enabled=true**;**

**}**

So one of the benefits of **multithreaded programming**is that it makes your application more responsive. In our next video, we will discuss the rest of the advantages and disadvantages of multithreaded programming.