**What is threading in C#?**

**One Line Definition :** Threading means parallel code execution.

Threading helps us to executing one or more works in a parallel way or asynchronous way.

**Asynchronous :** Asynchronous means doing one of more tasks simultaneously. Here second work doesn't need to wait for first work to get complete they can work simultaneusly.

**Synchronous :** Synchronous means executing one or more work one after the other. Here second work have to wait till the first work gets completed.

The most common example of threading is available in our personal computer i.e. working on multiple applications simultaneously i.e To do multiple things on our personal computer like working on Microsoft Powerpoint + Adobe Photoshop + Windows Media Player to create business presentation and executing multiple task asynchronously. So for each application a new thread is been defined by processor as a execution path of a program. So the process to execute more than one task at a time is called as Threading.

Threads are lightweight processes and C# supports parallel execution of code through multithreading environment. Each and every thread have an independent execution path to run applications simultaneosly. That's why on your personal desktop when you open up multiple applications you are actually creating a multithread environment to run applications simultaneously.

C# supports threading : C# Console application, C# WPF, C# Windows Forms are always starts in a single thread environment created by Common Language Runtime (CLR). So when you creates a new thread in a console application it means you are actually creating a multithread environment.

## Create Threading Step by Step

In c-sharp to implementing threading in our application first we need to import threading namespace i.e. "using System.Threading;" so by using thread objects we can use thread objects to create threads in applications.

After importing the namespace next step to create Thread objects as shown in below snippet.

using System;

using System.Threading;

class Program

{

public static void Main()

{

//Creates thread objects

Thread objthread = new Thread();

}

}

So after creating thread object let's invoke thread objects. To invoke thread objects let's demonstrate a small example. In this example first we will create multiple functions to check the output. We want to test wheather normally how both functions are executed. So let's test that.!

class Program

{

static void Main(string[] args)

{

Method1();

Method2();

}

static void Method1()

{

for (int i = 0; i <= 10; i++)

{

Console.WriteLine("Method One Executed " + i.ToString());

}

}

static void Method2()

{

for (int i = 0; i <= 10; i++)

{

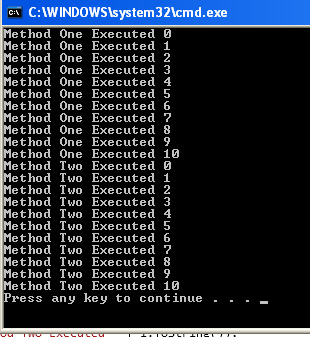
Console.WriteLine("Method Two Executed " + i.ToString());

}

}

}

**Display Output**



as you saw in our above output that "Method1()" executed first then followed by "Method2()" executed in a synchronous way or sequential way. But what we want is that both of these function should execute simultaneously. Here were threading mechanism comes in handy. So for the same application let's add threading mechanism.

**Step 1 : Import Threading Namespace**

using System;

using System.Threading;

**Step 2 : Create Thread Class and Thread Objects**

using System;

using System.Threading;

class Program

{

public static void Main()

{

//Creates thread objects

Thread objthread1 = new Thread(Method1);

Thread objthread2 = new Thread(Method2);

}

}

**Step 3 : Invoke Thread Objects**

using System;

using System.Threading;

class Program

{

public static void Main()

{

//Creates thread objects

Thread objthread1 = new Thread(Method1);

Thread objthread2 = new Thread(Method2);

thread1.Start();

thread2.Start();

}

static void Method1()

{

for (int i = 0; i <= 10; i++)

{

Console.WriteLine("Method One Executed " + i.ToString());

Thread.Sleep(4000); //Sleep for 4 seconds

}

}

static void Method2()

{

for (int i = 0; i <= 10; i++)

{

Console.WriteLine("Method Two Executed " + i.ToString());

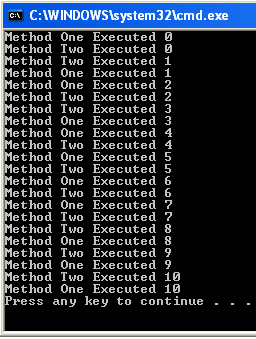
Thread.Sleep(4000); //Sleep for 4 seconds

}

}

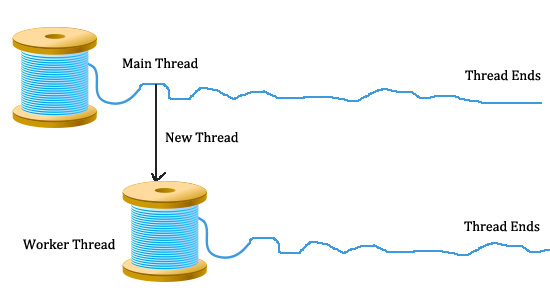
}

**Display Output**



In our above code example of a Console Application we have created two functions (Method1 and Method2 respectively). Inside both method bodys we have created the loop of 10 times and after each we have waited for 4 seconds. Kindly note for waiting we have used "Thread" class function i.e. "Sleep()". Now inside the main function body we have created the thread objects from "Thread" class and using those objects we called "Start()" function and using that "Start()" function we started execution of both function simultaneously.

C# supports threading and on dotnet platform CLR automatically starts a console application with a single thread or Main thread. If we add a new thread in our main program then main program creates a new thread (worker thread) and both of these threads starts their work in a multithread environment simultaneously. An example of multithread is shown below.



using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading;

namespace Threading

{

class Program

{

static void Main(string[] args)

{

Thread objworkerthread = new Thread(WorkerThread);

objworkerthread.Start();

for (int k = 0; k <= 10; k++)

{

Console.WriteLine("main thread");

Thread.Sleep(4000); //Sleep for 4 seconds

}

}

static void WorkerThread()

{

for (int i = 0; i <= 10; i++)

{

Console.WriteLine("worker thread");

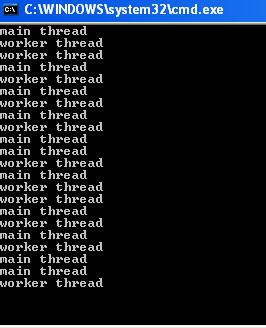
Thread.Sleep(4000); //Sleep for 4 seconds

}

}

}

}



as per both examples it is been proved that using threading we can execute mutiple task in a parallel way.

## Understand Thread.Join(), Thread.Sleep() and Thread.Abort() Methods

Join(), Sleep() and Abort() these are three main methods of a thread class which is widely used in a multithread implementation.

### Thread.Join() Method with example

Join waits for a thread to end. Join method when attached to any thread it makes that thread to finish its execution first or to end first and halts other processes. In simple words we can wait for another thread to end by calling its Join method. We can include TimeSpan or milliseconds with Join method.

**Example**

using System.Threading;

namespace Threading\_Demo1

{

class Demo4\_Join

{

static void Main(string[] args)

{

Thread objThread = new Thread(ProcessJoin);

objThread.Start();

// objThread.Join();

Console.WriteLine("Work Completed");

Console.ReadKey();

}

static void ProcessJoin()

{

for (int i = 1; i < 10; i++)

{

Console.WriteLine("Work in Progress");

}

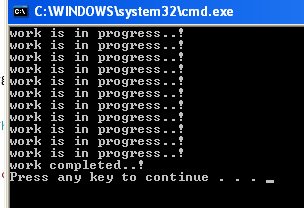
}

}

}

Above code is a simple example of Join method. In this program we have created a new thread for method "ProcessJoin" and waited "ProcessJoin" to finish its work by adding Join method. Above example first prints "work is in progress..!" 10 times and exit then prints "work completed..!" and main method exit.

**Display Output**



### Thread.Sleep() Method with example

Sleep method is used to suspend the current thread or pauses the current thread for specific time. The time can be specified in milliseconds or TimeSpan and in Sleep mode thread does not consume and CPU resources which indirectly saves the memory for other processes.

using System.Threading;

using System.Diagnostics;

namespace Threading\_Demo1

{

class Demo5\_Sleep

{

static void Main(string[] args)

{

Stopwatch stopwatch = new Stopwatch();

stopwatch.Start();

Thread objthread = new Thread(ProcessJoin);

Thread objthread2 = new Thread(ProcessJoin2);

objthread.Start();

objthread2.Start();

//objthread.Join();

objthread.Abort();

stopwatch.Stop();

TimeSpan ts = stopwatch.Elapsed;

string elapsedTime = string.Format("{0:00}:{1:00}:{2:00}", ts.Hours, ts.Minutes, ts.Seconds);

Console.WriteLine(" Total Time Elapsed " +elapsedTime);

Console.WriteLine(" Work Completed..");

Console.ReadLine();

}

public static void ProcessJoin()

{

for (int i = 1; i <= 5; i++)

{

Console.WriteLine("Work in Progress !");

Thread.Sleep(4000);

}

}

public static void ProcessJoin2()

{

for (int i = 1; i <= 5; i++)

{

Console.WriteLine("Work in Progress 2 !");

Thread.Sleep(4000);

}

}

}

}

In the above code example "ProcessJoin" makes a loop for 5 times and for every loop it sleeps for 4 seconds. We have calculated total sleep time using using System.Diagnostics namespace "stopwatch".

### Thread.Abort() Method with example

Thread.Abort method helps to terminate or to end thread. Abort method raises ThreadAbortException in the thread to do process of termination and throws ThreadAbortException in the thread to abort it. This exception can be caught in the application code. The complete termination can be done by calling Join method after Abort method.

static void Main(string[] args)

{

Stopwatch stopwatch = new Stopwatch();

stopwatch.Start();

Thread objthread = new Thread(ProcessJoin);

Thread objthread2 = new Thread(ProcessJoin2);

objthread.Start();

objthread2.Start();

//objthread.Join();

objthread.Abort();

stopwatch.Stop();

TimeSpan ts = stopwatch.Elapsed;

string elapsedTime = string.Format("{0:00}:{1:00}:{2:00}", ts.Hours, ts.Minutes, ts.Seconds);

Console.WriteLine(" Total Time Elapsed " +elapsedTime);

Console.WriteLine(" Work Completed..");

Console.ReadLine();

}

public static void ProcessJoin()

{

for (int i = 1; i <= 5; i++)

{

Console.WriteLine("Work in Progress !");

Thread.Sleep(4000);

}

}

public static void ProcessJoin2()

{

for (int i = 1; i <= 5; i++)

{

Console.WriteLine("Work in Progress 2 !");

Thread.Sleep(4000);

}

## }

## Types of Threads in C#

There two types of threads in c#.

1. Foreground Thread

2. Background Thread

### Foreground Thread

Foreground threads are those threads which keeps on running to complete its work even if the main thread quits. In simple words worker thread will keeps on running (to complete work) even if the main thread has ended the session. Here lifespan of workerthread is not dependent on the main thread. Worker thread can be alive without main thread.

using System.Threading;

namespace Threading\_Demo1

{

class Demo6\_Foreground

{

static void Main(string[] args)

{

Thread objworkerthread = new Thread(WorkerThread);

objworkerthread.Start();

Console.WriteLine( " Main Thread Quits");

//objworkerthread.Abort();

Console.ReadLine();

}

static void WorkerThread()

{

for (int i = 1; i < 5; i++)

{

Console.WriteLine("Worker Thread in Progress " );

Thread.Sleep(2000);

}

Console.WriteLine(" Worker thread Completed" );

}

}

}

### Background Thread

Background threads are those threads which quits if the main application method quits. Here lifespan of worker thread is dependent on the main thread. Worker thread quits if the main application thread quits. To use the background thread in an application we need to set a property called "IsBackground" to true.

using System.Threading;

namespace Threading\_Demo1

{

class Demp7\_Background

{

static void Main(string[] args)

{

Thread objthread = new Thread(Workerthread);

objthread.Start();

objthread.IsBackground = true;

Console.WriteLine(" Main thread Quits" );

}

public static void Workerthread()

{

for (int i = 1; i < 5; i++)

{

Console.WriteLine( "Worker thread in Progress");

Thread.Sleep(2000);

}

Console.WriteLine(" Worker Thread Completed" );

}

}

}