

Forum:[Hack Forums](#)

Thread: [\[C#\] Simple Multithreading \[TUT\]](#)

jhouns - October 4, 2010, 8:05 am

Simple Multithreading

If you want to leech or whatever it's all good just credit me and stuff.

What is Multithreading? [\(Click to View\)](#)

Every program runs in a thread created by the OS. If the program stops responding it's thread is closed. Ever noticed how when doing something that takes a while your program freezes? Well run it in a new thread and your program will work fine as the main thread is left to keep redrawing the interface as a new thread does the work!

EXAMPLE:

Windows form is doing something interesting like playing music. This takes up the main thread. You want to copy a file at the same time (via code on the form) but you can't because it would pause what everything else is doing to copy the file. Create a new thread that will run at the same time as the main one to copy the file and let the work be done in that instead leaving your music to play uninterrupted!

Simple thread creation in a command line environment [\(Click to View\)](#)

First of all open up a new command line project and create a class called 'Busy.cs'. This will have our function in to write out some numbers. It could be in the same class as Main() but hey ho whatever.

Here is the code for the Busy class:

```
Code using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace MultiThreading
{
    class Busy
    {
        public void DoMemoryIntensiveTask()
        {
            for (int i = 0; i < 100; i++)
            {
                Console.WriteLine(i);
            }

            Console.Read();
        }
    }
}
```

}Very simple. If you can't understand it you really shouldn't be learning about this yet...

Right now we have our extremely memory intensive task written ? we need to call it in our main

function! Here is how it's done:

```
Code static void Main(string[] args)
```

```
{
    Console.WriteLine("Creaing new instance of Busy.cs (Class).");
    Busy Job1 = new Busy();
    Console.WriteLine("Creating new thread.");

    Thread Job1Thread = new Thread(new ThreadStart(Job1.DoMemoryIntensiveTask));
    Console.WriteLine("Starting new thread.");
    Job1Thread.Start();
    for (int i = 0; i < 99; i++)
    {
        Console.WriteLine("DOING SOMETHING IN ORIGINAL THREAD");
    }
    Console.Read();
}
```

}Looks confusing? Possibly. Difficult? NO! Lets tear it apart shall we ;3 First we tell our viewer what's going on and then create a new instance of our 'Busy' class called 'Job1', doesn't matter what it's called could even be 'IHateXThreading'.

We then create a new thread called 'Job1Thread' (again whatever you want i just named it that so i understand what it does) that points to our function. But lets delve into it a little bit more:

```
Code Thread Job1Thread = new Thread(new ThreadStart(Job1.DoMemoryIntensiveTask));
```

The first part ('Thread Job1Thread = new Thread') is self evident however the next part is only somewhat. We create a new 'ThreadStart' with the parameter 'our function we want run in a new thread'. As far as i can tell it gets its memory address but we don't really need to know about that; just that in order to create a new thread we use

```
Code Thread <thread name> = new Thread(new ThreadStart(<function name>));
```

We then start the thread using '<thread name>.Start()' or in our case 'Job1Thread.Start()' which does exactly as it says before saying it's doing something 99 times.

If we run this program we will see that the processor is actually executing the 2 threads simultaneously (it's actually giving them an allotted space in which it's executed and cycles through those spaces but for the sake of this tutorial it's at the same time).

SO LETS RECAP! We wrote a function that will print 0 - 99 and then created a thread for it. We called that thread and printed something in the main thread. Instead of it being printed 0-99 THEN what we wanted in the main thread it does them at the same time and they are spliced together! (sort of simultaneously).

Multithreading in a Windows Form ([Click to View](#))

Coming as soon as i can be bothered to write it up.

Comments appreciated. Post suggestions/points needing clarification below.

Remember to say thanks and if you really love me [Click Here](#)(goes to google)

Slayer9x9 - October 4, 2010, 12:24 pm

Not a bad tut.

It is a little confusing - there are simpler ways to use threads.

I'm looking forward to your explanation on modifying form controls within other threads.

jhouns - *October 4, 2010, 1:10 pm*

I try to go into too much detail on everything >.< that's why i like people to point out sections which are too confusing so i can rewrite them. Trying to put ALL of your knowledge in one thread gets hard XD It'll be up when i get home tomorrow so not long to wait :)