

Forum:[Hack Forums](#)

Thread: [how to implement multithreading in c# ?](#)

fallag - January 5, 2021, 1:42 am

or maybe how should approach if u have any information just post it here . how to deal with corss heading variable ex. any information u have could be useful :)

xHeaven - January 5, 2021, 4:03 am

A very basic approach would be something like this.

PHP Code

```
<?php
using System;
using System.Threading;

public static void Main()
{
    Thread first = new Thread(firstThread);
    Thread second = new Thread(secondThread);

    first.Start();
    second.Start();
}

public static void firstThread()
{
    for (int i = 0; i < 10; i++) {
        Console.WriteLine($"{i} from firstThread");
    }
}

public static void secondThread()
{
    for (int i = 0; i < 10; i++) {
        Console.WriteLine($"{i} from secondThread");
    }
}
```

guardian - January 5, 2021, 4:09 am

If I am not mistaken you can use System.Threading for that
Then you can use it to manage your threads

Shadow_Cloud - January 5, 2021, 5:01 am

Can you explain what you are trying to achieve? It almost sounds like you are interested in synchronized access to specific resources from multiple threads? Doing things like accessing a static dictionary from multiple threads can have dire consequences if you aren't synchronizing the writes to that dictionary across threads.

This is very common in web application where different requests to the same application is actually executing on different threads with no explicit code written to access the information from separate threads.

Rmdy - January 5, 2021, 6:42 am

What do you mean by corss heading?

You mean CORS (Cross-Origin Resource Sharing), because if that's the cause, I don't see any correlation with your "current multi-threading" problem. I might be wrong so correction is very welcome.

Mind sharing your ideas or code if possible so we can understand your problem.

Regards,
Rmdy

riu41776 - January 5, 2021, 7:51 am

in .NET, you have maybe 6 ways to spawn a thread. Despite that, it all depends on your use-case in terms of code design.

Some cases might need a BackgroundWorker, some could just simply use the Thread class and of course the new TPL which abstracts a thread with useful features.

Back in the early days of .NET, you had BeginInvoke in 1.1 until 2.0 came out with other features.

neon_light - January 6, 2021, 1:53 pm

Spawning a new thread could certainly be a solution, but its not a very good one if you do a lot of async stuff because creating a new thread is fairly expensive.

You should probably use async await, it is easy to use and efficient. It uses a thread pool behind the scenes, so you are not spawning a new thread for every task.

<https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/async/>

StarZ - January 6, 2021, 4:56 pm

(Jan 5, 2021 07:42 AM)Rmdy Wrote: [\[CITE\]](#) What do you mean by corss heading?

You mean CORS (Cross-Origin Resource Sharing), because if that's the cause, I don't see any correlation with your "current multi-threading" problem. I might be wrong so correction is very welcome.

Mind sharing your ideas or code if possible so we can understand your problem.

Regards,
Rmdy
Probably meant cross-thread referencing of objects.

An example would be trying to access UI elements from a thread other than the UI thread. In which case, I'd lean towards using Events.

Aziz Shavershian - February 13, 2021, 7:47 pm

I think that Tasks are much more relevant when it comes to parallel programming than threads.

- Tasks do not create own their own OS thread, but are executed by a task scheduler
- Tasks allow you to find out when they have finished

-can be used in sync/async programming i.e. continue with Y, when X is finished

In conclusion tasks are much more efficient and have a stronger API in comparison to threads.

uint32 - February 13, 2021, 9:44 pm

you should much rather use tpl pattern (async/await)

here's an example:

Code public void Main()

```
{
    //now this method will execute asynchronously
    //it will run in the background so the program will go straight to the other work while the
function finishes
    var task = DoWork();
    Console.WriteLine("started processing...");

    //do some other work
    Task.Delay(5000).Wait();

    //now we get the return value of our task
    var result = task.Result;

    Console.WriteLine(result);
}
public async Task<string> DoWork()
{
    //here is where your slow logic/code goes
    await Task.Delay(1000);

    return "completed work!";
}
```

this snippet will finish executing after 5000ms, and not 6000ms if the code was synchronous

there are many advantages to using tasks over threads:

- tasks are managed by the threadpool which avoids the slow process creating/destroying threads which is also inefficient at the os level
- code ends up being much cleaner and functional

references:

task class: <https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.task?view=net-5.0>

async/await: <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/async/>