TutorialKart

C# Multithreading – Examples

C# Multithreading

Mutlithreading can be used to take advantage of the processing power you have with the CPU. Nowadays, most of the processes used in computers are multicore. Therefore they can support multiple processes at a time.

We have hardware powerful enough to compute multiple threads simultaneously. Do we have software that can process simultaneously? We can write one using C#.

Every C# application has a thread called Main thread which is responsible for the sequential flow of program. And multiple threads can be started from Main thread using System.Threading namespace.

When should Multithreading be considered?

Not every application needs multithreading. But you should know when to use multithreading to make your application faster. Let us dive into some of the use-cases where multithreading can be used.

Consider that you need to download all the wiki pages. For downloading each page, your program may need to do some HTTP requests and wait for the response. The network may take significant amount of time to save the page and continue with the next page. During the network latency, our thread will have to wait. But, what if we start about 10 threads that run simultaneously to download the pages. While one therad is waiting for the network response, other threads can request for other pages. Thus making the total program execution cut down to nearly 10%. Pretty amazing with the performance, right!

Now, this idea can be extended to operations that include database operations, file operations, or any methods that take significant amount of time and make the thread idle waiting for the result from other resources.

Define Thread in C#

To define a thread in C#, use the namespace `System.Threading`. From the namespace, use `Thread` class to declare a new Thread.

```
Thread thread1 = new Thread(someMethod);
```

We have defined a thread named `thread1` that can execute `someMethod` simultaneously with other thread(s) if any.

thread(s) if any.

Start Thread in C#

To start a thread, you can call `Start()` method on the thread reference

```
thread1.Start();
```

Example – C# Multithreading

In the following example, we have two methods named `method1` and `method2`. Let us define two threads: one Thread with `method1` and the other thread with `method2`. Then, we start these two threads and observe the output.

Program.cs

```
using System;
using System.Threading;

namespace CSharpExamples {

    class Program {

        public static void method1() {
            for(int i=0;i<10;i++) {
                Console.WriteLine("method1");
            }
        }

        public static void method2() {
            for(int i=0;i<10;i++) {
                Console.WriteLine("method2");
            }
        }

        static void Main(string[] args) {
            Thread thread1 = new Thread(method1);
            Thread thread2 = new Thread(method2);
            thread1.Start();
            thread2.Start();
        }
    }
}
```

Output

```
PS D:\workspace\csharp\HelloWorld> dotnet run
method1
method1
method1
method1
method2
```

```
method2
method2
method2
method2
method2
method2
method2
method2
method2
method1
method1
method1
method1
method1
method1
```

We can observe that Thread2 started right after Thread1, not waiting for Thread1 to complete. Meaning, Thread1 and Thread2 started running simultaneously.

Conclusion

In this C# Tutorial, we learned what Multithreading is, and how to implement multithreading in C#.

- C# String
- C# Array
- C# Command Line Arguments

C# Console Operations

- C# Write to Console
- C# Read from Console

C# Object Oriented Programming Concepts

- C# Class & Constructors
- C# Encapsulation
- C# Polymorphism
- C# Method Overloading
- C# Interfaces

C# String Operations

- C# String Length
- C# Substring

C# File Operations

- C# Read Text File
- C# Write to File
- C# Delete File
- C# Copy File

C# Exception Handling

- C# try-catch
- C# finally
- C# throw
- C# Custom Exception
- C# SystemException
- C# DivideByZeroException
- C# NullReferenceException
- C# InvalidCastException

- C# IOException
- C# FieldAccessException

C# Collections

- C# List
- C# SortedList
- C# HashSet
- C# SortedSet
- C# Stack
- C# Queue
- C# LinkedList
- C# Dicionary
- C# SortedDictionary

C# Errors [Solved]

- C# Error: Class does not contain a constructor that takes arguments