1. **hat is Multitasking?**
2. **How the operating system executes multiple applications at a time?**
3. **What is Thread?**
4. **Understanding the Thread class.**
5. **What are the drawbacks of Single-Threaded Applications?**
6. **How to overcome the drawbacks of the Single-Threaded Application using Multithreading?**

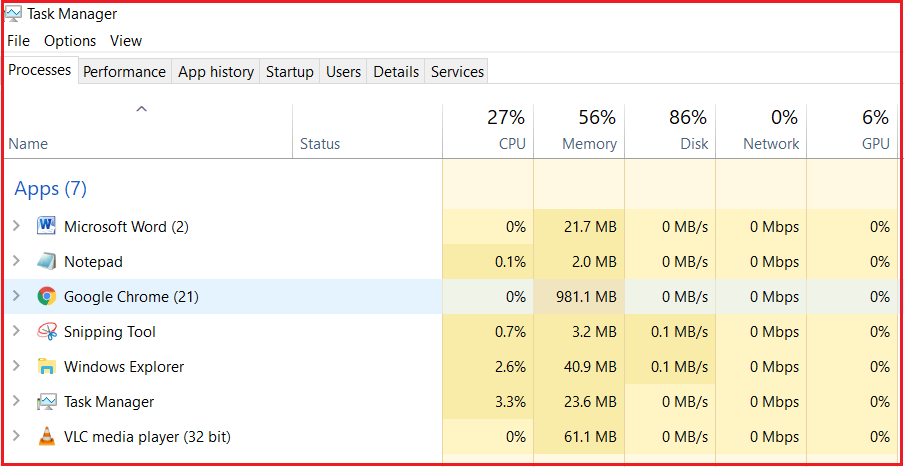
**What is Multitasking?**

Before understanding the concept of Multithreading in C#, let us first understand multitasking. Windows operating system is a multitasking operating system. It means it has the ability to run multiple applications at the same time. For example, in my machine, I can open the Google Chrome Browser, Microsoft word document, Notepad, VLC Media Player, Windows Explorer, etc. at the same time. This is possible because in my machine I have installed the Windows operating system and the Windows operating system is a multitasking operating system.

**How the operating system executes multiple applications at a time?**

To execute all the above applications, the operating system internally makes use of processes. A process is a part of the operating system (or a component under the operating system) which is responsible for executing the program or application. So, to execute each and every program or application, there will be a process.

You can see this using the Task Manager. Just right-click on the Taskbar and click on the Task Manager option which will open the Task Manager window. From that window, just click on the “Processes” button as shown below.



As you can see from the above image, each application is executing by one corresponding process. Along the same line, there are also multiple processes that are running in the background which are known as the background processes. These background processes are known as windows services and the Operating system runs a lot of windows services in the background.

So, we have an operating system and under the operating system, we have processes that running our applications. So under the process, an application runs. To run the code of an application the process will make use of a concept called Thread.

**What is Thread?**

Generally, a Thread is a lightweight process. In simple words, we can say that a Thread is a unit of a process that is responsible for executing the application code. So, every program or application has some logic or code and to execute that logic or code, Thread comes into the picture.

By default, every process has at least one thread which is responsible for executing the application code and that thread is called as Main Thread. So, every application by default is a single-threaded application.

**Note:** All the threading related classes in C# belong to the **System.Threading** namespace.

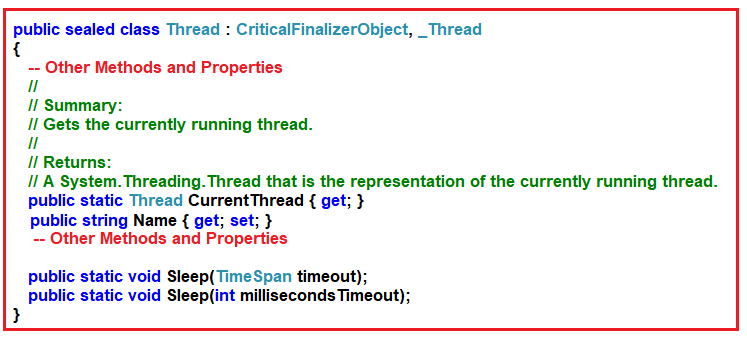
**Let us see an example to understand Threading.**

* **using** System;
* **namespace** ThreadingDemo
* **{**
* class Program
* **{**
* static **void** Main**(**string**[]** args**)**
* **{**
* Console.WriteLine**(**"Welcome to Dotnet world!"**)**;
* **}**
* **}**
* **}**

This is a very simple program. But internally there is a thread that is going to execute this code and that thread is called Main Thread. Now let us prove this.

**Understanding the Thread class in C#:**

The Thread class contains one static property i.e. **CurrentThread** which is going to return the instance of the currently executing thread. If you go to the definition of **Thread** class then you will find the following signature.



As you can see the **CurrentThread** static property return type is Thread i.e. it is going to return the instance of the currently executing thread. Along the same line, there is a non-static property called Name using which we can set and get the Name of the currently executing thread.

**Note:** By default, the thread does not have any name. If you want then you can provide any name to the thread by using the Name property of the Thread class. So, modify the program as shown below.

* **using** System.Threading;
* **using** System;
* **namespace** ThreadingDemo
* **{**
* class Program
* **{**
* static **void** Main**(**string**[]** args**)**
* **{**
* Thread t = Thread.CurrentThread;
* //By Default the Thread does not have any name
* //if you want then you can provide the name explicitly
* t.Name = "Main Thread";
* Console.WriteLine**(**"Current Executing Thread Name :" + t.Name**)**;
* Console.WriteLine**(**"Current Executing Thread Name :" + Thread.CurrentThread.Name**)**;
* Console.Read**()**;
* **}**
* **}**
* **}**

**Output:**

Printing the Single Thread Name in C#

As you can see in order to run the application code one thread is created and i.e. the Main Thread. So, this proves that, by default, every application is a single-threaded application.

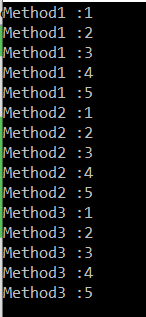
**What are the drawbacks of Single-Threaded Applications?**

In a single thread application, all the logic or code present in the program will be executed by a single thread only i.e. the Main thread. For example, if we have three methods in our application and if all these three methods are going to be called from the Main method. Then the main thread is responsible to execute all these three methods sequentially i.e. one by one. It will execute the first method and once it completes the execution of the first method then only it executes the second method and so on.

Let us understand this with an example. Modify the program as shown below.

* **using** System;
* **namespace** ThreadingDemo
* **{**
* class Program
* **{**
* static **void** Main**(**string**[]** args**)**
* **{**
* Method1**()**;
* Method2**()**;
* Method3**()**;
* Console.Read**()**;
* **}**
* static **void** Method1**()**
* **{**
* **for** **(**int i = 1; i <= 5; i++**)**
* **{**
* Console.WriteLine**(**"Method1 :" + i**)**;
* **}**
* **}**
* static **void** Method2**()**
* **{**
* **for** **(**int i = 1; i <= 5; i++**)**
* **{**
* Console.WriteLine**(**"Method2 :" + i**)**;
* **}**
* **}**
* static **void** Method3**()**
* **{**
* **for** **(**int i = 1; i <= 5; i++**)**
* **{**
* Console.WriteLine**(**"Method3 :" + i**)**;
* **}**
* **}**
* **}**
* **}**

**Output:**



As you can see in the above output, the methods are called and execute one after the other. The Main thread first executes Method1 and once it completes the execution of Method1 then it calls Method2 and then Method3.

**What is the problem with the above program execution?**

In our example, we are just writing some simple code to print the values from 1 to 5. What will you do if one method is taking more than the expected time? Suppose Method2 is going to interact with a database or it is going to invoke any web service which will take more than 10 seconds to provide the response. In that case, the Method2 execution will be delayed for 10 seconds as it is waiting there to get a response back either from the database or from the Web Service. Until Method2 is not completed its execution, Method3 is not going to be executed because of the sequential execution of the program i.e. one by one.

**Let us understand this with an example.**

Note: Here we are not going to perform any database or Web Service call instead we can use the Thread class Sleep method to delay the execution of Method2 for 10 seconds. Following is the signature of Sleep Method:

**public static void Sleep(int millisecondsTimeout);**

The sleep method takes the time in milliseconds as input and then suspends the current thread execution for that specified number of milliseconds. So, please modify the Program as shown below.

* **using** System.Threading;
* **using** System;
* **namespace** ThreadingDemo
* **{**
* class Program
* **{**
* static **void** Main**(**string**[]** args**)**
* **{**
* Method1**()**;
* Method2**()**;
* Method3**()**;
* Console.Read**()**;
* **}**
* static **void** Method1**()**
* **{**
* **for** **(**int i = 1; i <= 5; i++**)**
* **{**
* Console.WriteLine**(**"Method1 :" + i**)**;
* **}**
* **}**
* static **void** Method2**()**
* **{**
* **for** **(**int i = 1; i <= 5; i++**)**
* **{**
* Console.WriteLine**(**"Method2 :" + i**)**;
* **if** **(**i == 3**)**
* **{**
* Console.WriteLine**(**"Performing the Database Operation Started"**)**;
* //Sleep for 10 seconds
* Thread.Sleep**(**10000**)**;
* Console.WriteLine**(**"Performing the Database Operation Completed"**)**;
* **}**
* **}**
* **}**
* static **void** Method3**()**
* **{**
* **for** **(**int i = 1; i <= 5; i++**)**
* **{**
* Console.WriteLine**(**"Method3 :" + i**)**;
* **}**
* **}**
* **}**
* **}**

Now run the application and notice that the Method2 execution is delayed for 10 seconds. Once Method2 completes its execution then only Method3 start its execution. This is because all these three methods are executed by a single thread and this is the drawback of the single-threaded application.

**How to solve the above problem?**

To solve the above problem, we are provided with a concept called **Multithreading in C#**. As we already discussed Operating System has Processes which is used to run our applications. The Process contains Thread which will actually run our application code.

A process can have multiple threads and each thread can perform a different task. In simple words, we can say that the three methods we define in our program can be executed by three different threads. The advantage is that the execution takes place simultaneously. So when multiple threads trying to execute the application code, then the operating system allocates some time period to each thread to execute.

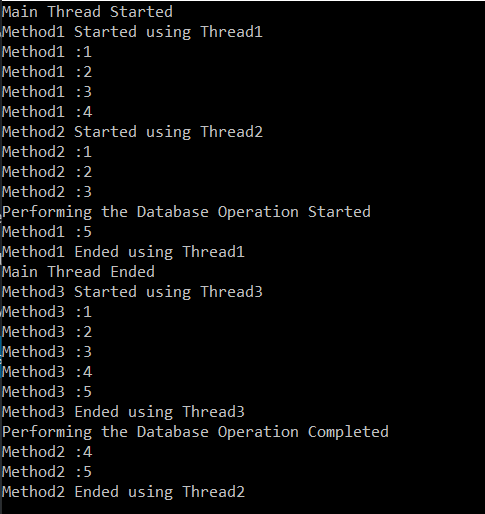
Now, in our example, we want to execute the three methods using three different threads let say t1, t2, and t3. The thread t1 is going to execute Method1, thread t2 is going to execute the Method2. At the same time, the Method3 is going to be executed by thread t3. Let us modify the Program as shown below to execute the methods with different Threads.

* **using** System.Threading;
* **using** System;
* **namespace** ThreadingDemo
* **{**
* class Program
* **{**
* static **void** Main**(**string**[]** args**)**
* **{**
* Console.WriteLine**(**"Main Thread Started"**)**;
* //Creating Threads
* Thread t1 = **new** Thread**(**Method1**)**
* **{**
* Name = "Thread1"
* **}**;
* Thread t2 = **new** Thread**(**Method2**)**
* **{**
* Name = "Thread2"
* **}**;
* Thread t3 = **new** Thread**(**Method3**)**
* **{**
* Name = "Thread3"
* **}**;
* //Executing the methods
* t1.Start**()**;
* t2.Start**()**;
* t3.Start**()**;
* Console.WriteLine**(**"Main Thread Ended"**)**;
* Console.Read**()**;
* **}**
* static **void** Method1**()**
* **{**
* Console.WriteLine**(**"Method1 Started using " + Thread.CurrentThread.Name**)**;
* **for** **(**int i = 1; i <= 5; i++**)**
* **{**
* Console.WriteLine**(**"Method1 :" + i**)**;
* **}**
* Console.WriteLine**(**"Method1 Ended using " + Thread.CurrentThread.Name**)**;
* **}**
* static **void** Method2**()**
* **{**
* Console.WriteLine**(**"Method2 Started using " + Thread.CurrentThread.Name**)**;
* **for** **(**int i = 1; i <= 5; i++**)**
* **{**
* Console.WriteLine**(**"Method2 :" + i**)**;
* **if** **(**i == 3**)**
* **{**
* Console.WriteLine**(**"Performing the Database Operation Started"**)**;
* //Sleep for 10 seconds
* Thread.Sleep**(**10000**)**;
* Console.WriteLine**(**"Performing the Database Operation Completed"**)**;
* **}**
* **}**
* Console.WriteLine**(**"Method2 Ended using " + Thread.CurrentThread.Name**)**;
* **}**
* static **void** Method3**()**
* **{**
* Console.WriteLine**(**"Method3 Started using " + Thread.CurrentThread.Name**)**;
* **for** **(**int i = 1; i <= 5; i++**)**
* **{**
* Console.WriteLine**(**"Method3 :" + i**)**;
* **}**
* Console.WriteLine**(**"Method3 Ended using " + Thread.CurrentThread.Name**)**;
* **}**
* **}**
* **}**

**Code Explanation:**

As you can see in the above code, we have created three different instances of Thread class. To the constructor of Thread class, we need to pass the method name which needs to be executed by that Thread. Then we call the **Start()** method on the Thread class which will start executing the method. Here the Main thread is going to create all other Threads.

**Note:** You will not get the output in a sequential manner. Run the application and see the output as shown below. The output may vary in your machine.



**Note:** The main advantage of using Multithreading is the maximum utilization of CPU resources.

Here, in this article, I try to explain the concept of Multithreading in C# with some simple examples