# Multithread (Multithread) in C #

[Thread in c #] A thread is defined as an execution path (execution path) of a program. Each Thread defines a single control line. If your application includes complex and time-consuming activities, it is often useful to set up execution paths or Thread, with each Thread performing a specific task.

**[Thread in c #]** . A **thread** is defined as an execution path (execution path) of a program. Each Thread defines a single control line. If your application includes complex and time-consuming activities, it is often useful to set up execution paths or Thread, with each Thread performing a specific task.

Threads are **lightweight processes** . A common example of using Thread is the implementation of concurrent programming by modern operating systems. Using Thread saves CPU cycle wastage and increases the efficiency of an application.

To this chapter, we already know the programs that a single Thread runs as a single process, which is the application launcher. However, in this way, the application can perform one job at a time. To make it execute more than one task at a time, it can be divided into smaller Threads.

## Thread's life cycle in C #

The lifecycle of a Thread begins when an object of the **System.Threading.Thread** class is created and ends when the Thread is terminated or the execution is completed.

Here are the various states in the life of a Thread in C #:

**Unstarted State** : This is the situation when the Thread instance is created, but the Start method is not called.

**Ready State** : This is the situation when the Thread is ready to run and waits for CPU cycle.

**Not Runnable State** : A Thread is not executable (not executable), when:

- Sleep method has been called.

- Wait method has been called.

- Blocked by I / O operation.

**Dead State** : It is the situation when the Thread completes execution or is canceled.

## Main Thread in C #

In C #, the **System.Threading** class. **Thread** is used to work with Thread. It allows creating and accessing separate Threads in a Multithreaded Application. The first thread to be executed in a process is called Main Thread in C #.

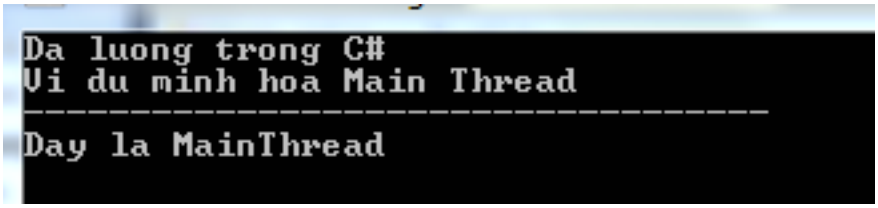
When a C # program starts executing, the **Main** Thread is automatically created. Threads, created using the Thread class, are called the **Thread Thread** of Main Thread. You can access a Thread using the **CurrentThread** property of the Thread class.

Below is an example program illustrating the implementation of Main Thread in C #:

```
using System ; using System . Threading ; namespace QTMCsharp { class TestCshar
```

If you do not use the **Console.ReadKey** () command; then the program will run and finish (so fast that you can not see the results). This command allows us to see the results more clearly.

Compiling and running the above C # program will produce the following results:




# Properties and Methods of Thread class in C #

The table below lists some of the most commonly used properties of the Thread class in C #:

DescriptionCurrentContext Properties Get the current context in which the Thread is executing CurrentCulture Get or set culture including language, date, time, currency, . for the Current CurrentPrinciple Retrieve or set the current principle of Thread CurrentThread Get the currently running Thread CurrentUICulture Get or set the current culture used by Resource Manager to search for a specific Resource at runtime ExecutionContext Get an ExecutionContext object that contains information about the various context of the current IsAlive Thread Retrieved a value indicating the execution state of the current Thread IsBackground Get or set a value indicating whether or not a Thread is Background Thread IsThreadPoolThread Get a value indicating whether or not a Thread is of Managed Thread Pool Manag edThreadId Get a unique identifier for the current Managed Thread Name Get or set the name of Thread Priority Get or set a value indicating the priority of a Thread ThreadState Get a value containing the status of the current Thread
This table lists the most commonly used methods of the **Thread** class in C #:

**Public void Abort ()**

Create a ThreadAbortException in the Thread on which it is summoned, to start the Thread termination process. Calling this method usually ends Thread

2 **public static LocalDataStoreSlot AllocateDataSlot ()**

Allocate an Unnamed Data Slot for all Thread. To increase performance, using Fields marked with the attribute is ThreadStaticAttribute instead

3 **public static LocalDataStoreSlot AllocateNamedDataSlot (string name)**

Allocate a Named Data Slot for all Thread. To increase performance, using Fields marked with the attribute is ThreadStaticAttribute instead

4 **public static void BeginCriticalRegion ()**

Inform a host that the execution is prepared to enter a code area, in which the effects of canceling a Thread or unprocessed Exceptions can jeopardize other domain tasks. application

5 **public static void BeginThreadAffinity ()**

Inform a Host that Managed code is prepared to execute instructions that depend on the integrity of the current Physical operating system thread

6 **public static void EndCriticalRegion ()**

Inform a host that the execution is prepared to enter a code area, in which the effects of canceling a Thread or unprocessed Exceptions are restricted to the current task.

7 **public static void EndThreadAffinity ()**

Inform a Host that Managed code has finished executing instructions that depend on the uniformity of the current Physical Operating System Thread

8 **public static void FreeNamedDataSlot (string name)**

Eliminate the association between a name and a slot, for all Thread in progress. To increase performance, using Fields marked with the attribute is ThreadStaticAttribute instead

9 **public static Object GetData (LocalDataStoreSlot slot)**

Revoke the value from the slot defined on the current Thread, inside the current domain of the current Thread. To increase performance, using Fields marked with the attribute is ThreadStaticAttribute instead

10 **public static AppDomain GetDomain ()**

Returns the current domain in which the Thread is running

11 **public static AppDomain GetDomain ()**

Returns a unique application domain identifier

12 **public static LocalDataStoreSlot GetNamedDataSlot (string name)**

Search for a Named Data Slot. To increase performance, using Fields marked with the attribute is ThreadStaticAttribute instead

13 **public void Interrupt ()**

Interrupt a Thread that is in the WaitSleepJoin state

14 **public void Join ()**

Blocking Thread is calling when a Thread ends, while continuing to execute COM and SendMessage Pumping. This method has different overloaded patterns

15 **public static void MemoryBarrier ()**

Memory access synchronization is as follows: The Prosessor implementing the current Thread cannot reorder the instructions in such a way that the memory access to the call to MemoryBarrier executes after the memory access follows. after that call to MemoryBarrier

16 **public static void ResetAbort ()**

Canceling an Abort is required for the current Thread

17 **public static void SetData (LocalDataStoreSlot slot, Object data)**

Set the data in the slot given on the currently running Thread, for the current domain of that Thread. To increase performance, using Fields marked with the attribute is ThreadStaticAttribute instead

18 **public void Start ()**

Start a Thread

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

Make Thread stop for a period of time

20 **public static void SpinWait (int iterations)**

Making a Thread waits for a specified time period in the iterations parameter

21
**public static byte VolatileRead (ref byte address)**

**public static double VolatileRead (ref double address)**

**public static int VolatileRead (ref int address)**

**public static Object VolatileRead (ref Object address)**

Read the value of a Field. This value is the latest written by any Prosessor in a computer, regardless of the Prosessor's number or the Prosessor Cache status. This method has different overloaded patterns. Those are the above forms

22
**public static void VolatileWrite (ref byte address, byte value)**

**public static void VolatileWrite (ref double address, double value)**

**public static void VolatileWrite (ref int address, int value)**

**public static void VolatileWrite (ref Object address, Object value)**

Write a value to a Field immediately, so that this value is visible to all Processors in the computer. This method has different overloaded patterns. Those are the above forms

23 **public static bool Yield ()**

As the Thread is calling, switch the execution to another Thread that is ready to run on the current Processor. Operating system select Thread to move to

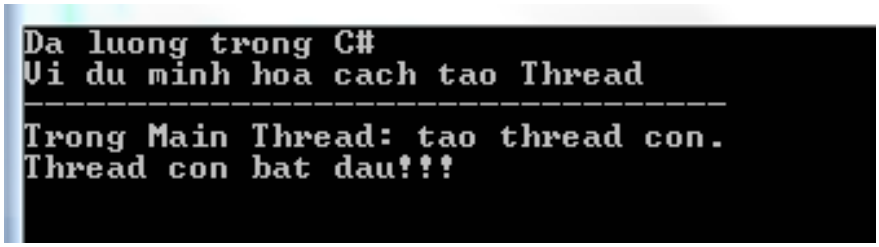
# Create Thread in C #

In C #, Threads are created by inheriting the Thread class. Then, the Thread class is inherited to call the **Start** () method to start the execution of the Thread child.

Here is an example to illustrate the creation of Thread in C #:

```
using System ; using System . Threading ; namespace QTMCsharp { class TestCshar
```

Compiling and running the above C # program will produce the following results:



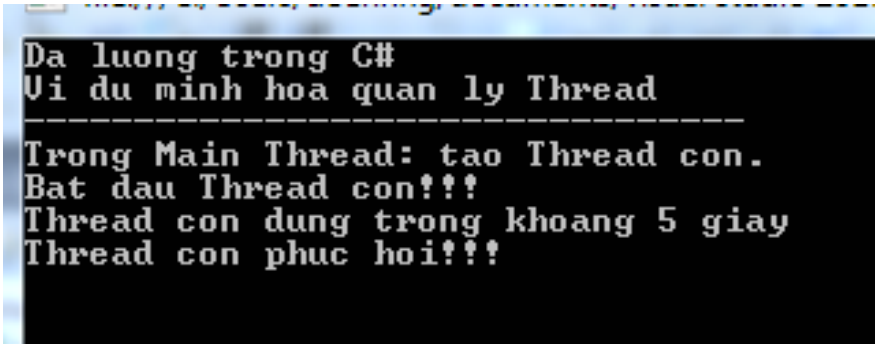

# Thread Management in C #

The Thread class in C # provides a variety of methods for managing Threads.

The following example illustrates how to use the **sleep** () method to make a Thread stop for a specific time period.

```
using System ; using System . Threading ; namespace QTMCsharp { class TestCshar
```

Compiling and running the above C # program will produce the following results:




# Cancel Thread in C #

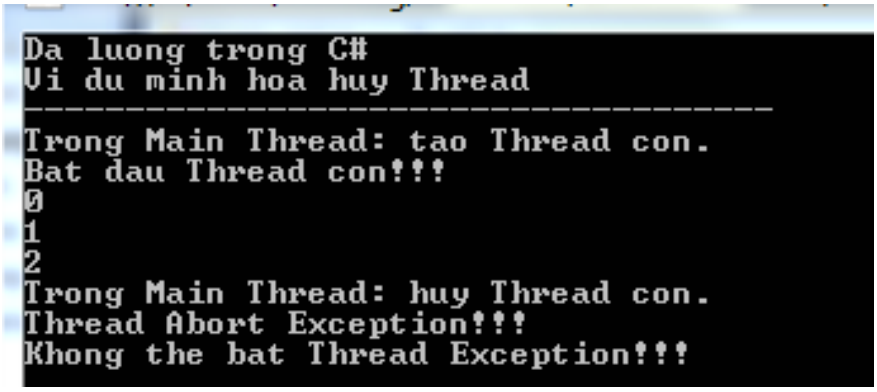
The **Abort** () method is used to cancel the Thread in C #.

During runtime, the program aborts Thread by throwing a ThreadAbortException. This exception cannot be captured, the control is sent to the **finally** block, if not.

Below is a program that illustrates the use of the Abort () method to abort the Thread in C #:

```
using System ; using System . Threading ; namespace QTMCsharp { class TestCshar
```

Compiling and running the above C # program will produce the following results:




**Follow tutorialspoint**

Previous article: Unsafe code in C #

Next post: PHP & AJAX

You finished reading the article "**Multithread (Multithread) in C #**" edited by the **TipsMake** team. We hope this article has provided you with many useful tech tips and tricks. You can search for similar articles on tips and guides. Thank you for reading and for following us regularly.