



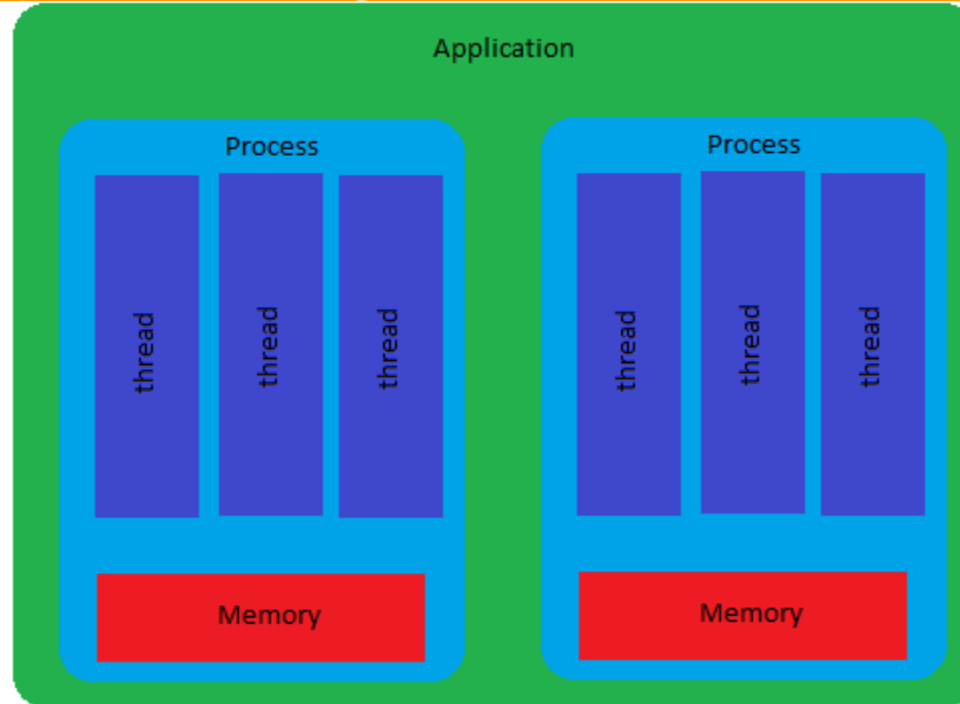
# LẬP TRÌNH C# 2

## BÀI 8 : MULTI-THREADING

- ⊙ Multithreading
- ⊙ Các tính năng Multithreading



- ❑ Multitasking trên hệ điều hành có nghĩa là có thể chạy nhiều ứng dụng tại một thời điểm
- ❑ Multithreading : khả năng thực hiện nhiều thread đồng thời tại một thời điểm.
- ❑ Thread: hay còn gọi là tiểu trình, cho phép chương trình thực hiện đồng thời nhiều tác vụ, và giúp quá trình tương tác với người dùng không bị gián đoạn
- ❑ Process: là một instance của chương trình máy tính được thực thi, dựa trên hệ điều hành, hoàn toàn độc lập với các tiến trình khác



- ❑ Mỗi 1 chương trình có thể gồm 1 hoặc nhiều process. Các process sẽ có thể sử dụng lượng tài nguyên riêng không liên quan đến process khác. Mỗi process có thể gồm nhiều thread. Các thread trong cùng 1 process sẽ sử dụng chung tài nguyên với nhau.

## ❑ Một số thuộc tính được sử dụng phổ biến nhất của lớp Thread trong C#

Thuộc tính	Miêu tả
CurrentContext	Lấy ngữ cảnh (context) hiện tại mà trong đó Thread đang thực thi
CurrentCulture	Lấy hoặc thiết lập culture gồm language, date, time, currency, ... cho Thread hiện tại
CurrentPrinciple	Lấy hoặc thiết lập nguyên lý hiện tại của Thread
CurrentThread	Lấy Thread đang chạy hiện tại
CurrentUICulture	Lấy hoặc thiết lập culture hiện tại được sử dụng bởi Resource Manager để tìm kiếm cho Resource cụ thể tại runtime
ExecutionContext	Lấy một đối tượng ExecutionContext mà chứa thông tin về các context đa dạng của Thread hiện tại
IsAlive	Lấy một giá trị chỉ trạng thái thực thi của Thread hiện tại
IsBackground	Lấy hoặc thiết lập một giá trị chỉ rằng có hay không một Thread là Background Thread
IsThreadPoolThread	Lấy một giá trị chỉ rằng có hay không một Thread là của Managed Thread Pool
ManagedThreadId	Lấy một định danh duy nhất cho Managed Thread hiện tại
Name	Lấy hoặc thiết lập tên của Thread
Priority	Lấy hoặc thiết lập một giá trị chỉ quyền ưu tiên của một Thread
ThreadState	Lấy một giá trị chứa các trạng thái của Thread hiện tại

## ❑ Một số phương thức được sử dụng phổ biến nhất của lớp Thread trong C#

### Phương thức

#### **public void Abort()**

Tạo một ThreadAbortException trong Thread mà trên đó nó được triệu hồi, để bắt đầu tiến trình kết thúc Thread đó. Gọi phương thức này thường kết thúc Thread

#### **public static LocalDataStoreSlot AllocateDataSlot()**

Cấp phát một Unnamed Data Slot cho tất cả Thread. Để tăng hiệu suất, sử dụng các Field mà được đánh dấu với attribute là ThreadStaticAttribute để thay thế

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

Cấp phát một Named Data Slot cho tất cả Thread. Để tăng hiệu suất, sử dụng các Field mà được đánh dấu với attribute là ThreadStaticAttribute để thay thế

#### **public static void BeginCriticalRegion()**

Thông báo cho một host rằng sự thực thi là chuẩn bị đi vào một khu vực code, mà trong đó các ảnh hưởng của việc hủy bỏ một Thread hoặc các Exception không được xử lý có thể gây nguy hại tới các tác vụ khác trong miền ứng dụng

#### **public static void BeginThreadAffinity()**

Thông báo cho một Host rằng Managed code là chuẩn bị thực thi các chỉ lệnh mà phụ thuộc vào tính đồng nhất của Physical operating system thread hiện tại

#### **public static void EndCriticalRegion()**

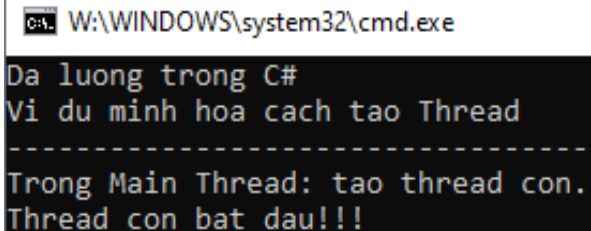
Thông báo cho một host rằng sự thực thi là chuẩn bị đi vào một khu vực code, mà trong đó các ảnh hưởng của hủy bỏ một Thread hoặc các Exception không được xử lý bị hạn chế tới tác vụ hiện tại

## ❑ Tạo Thread

- ❖ Các Thread được tạo bằng việc kế thừa lớp Thread
- ❖ Lớp Thread được kế thừa gọi phương thức **Start()** để bắt đầu sự thực thi của Thread con

```
1 reference
public static void CallToChildThread()
{
    Console.WriteLine("Thread con bat dau!!!");
}

0 references
static void Main(string[] args)
{
    Console.WriteLine("Da luong trong C#");
    Console.WriteLine("Vi du minh hoa cach tao Thread");
    Console.WriteLine("-----");
    ThreadStart childref = new ThreadStart(CallToChildThread);
    Console.WriteLine("Trong Main Thread: tao thread con.");
    Thread childThread = new Thread(childref);
    childThread.Start();
    Console.ReadKey();
}
```



```
C:\ W:\WINDOWS\system32\cmd.exe
Da luong trong C#
Vi du minh hoa cach tao Thread
-----
Trong Main Thread: tao thread con.
Thread con bat dau!!!
```

## ❑ Ví dụ tạo và thực thi thread

```
static void Main()
{
    Thread t = new Thread(new ThreadStart(MethodA));
    t.Start();
    MethodB();
}

1 reference
static void MethodA()
{
    for (int i = 0; i < 50; i++)
        Console.Write("0");
}

1 reference
static void MethodB()
{
    for (int i = 0; i < 50; i++)
        Console.Write("1");
}
```

❑ 2 phương thức A và B chạy song song

[illegible]



- ❑ Quản lý Thread: Ví dụ sau minh họa cách sử dụng phương thức `sleep()` để làm một Thread dừng trong một khoảng thời gian cụ thể.

```
public static void CallToChildThread()
{
    Console.WriteLine("Bat dau Thread con!!!");

    // Thread nay dung khoang 5000 milisecond
    int sleepfor = 5000;

    Console.WriteLine("Thread con dung trong khoang {0} giay", sleepfor / 1000);
    Thread.Sleep(sleepfor);
    Console.WriteLine("Thread con phuc hoi!!!");
}
```

0 references

```
static void Main(string[] args)
{
    Console.WriteLine("Da luong trong C#");
    Console.WriteLine("Vi du minh hoa quan ly Thread");
    Console.WriteLine("-----");

    ThreadStart childref = new ThreadStart(CallToChildThread);
    Console.WriteLine("Trong Main Thread: tao Thread con.");
    Thread childThread = new Thread(childref);
    childThread.Start();
    Console.ReadKey();
}
```

C:\ W:\WINDOWS\system32\cmd.exe

```
Da luong trong C#
Vi du minh hoa quan ly Thread
-----
Trong Main Thread: tao Thread con.
Bat dau Thread con!!!
Thread con dung trong khoang 5 giay
Thread con phuc hoi!!!
```

## ❑ Hủy Thread

- ❖ Phương thức `Abort()` được sử dụng để hủy các Thread trong C#.
- ❖ Trong thời gian runtime, chương trình hủy bỏ Thread bằng việc ném một *ThreadAbortException*

## ❑ Hủy Thread

```
public static void CallToChildThread()
{
    try
    {
        Console.WriteLine("Bat dau Thread con!!!");

        // gia su chung ta dem tu 0 toi 10
        for (int counter = 0; counter <= 10; counter++)
        {
            Thread.Sleep(500); //dung trong khoang 5 giay
            Console.WriteLine(counter);
        }

        Console.WriteLine("Thread con hoan thanh.");
    }

    catch (ThreadAbortException e)
    {
        Console.WriteLine("Thread Abort Exception!!!");
    }
    finally
    {
        Console.WriteLine("Khong the bat Thread Exception!!!");
    }
}
```

```
static void Main(string[] args)
{
    Console.WriteLine("Da luong trong C#");
    Console.WriteLine("Vi du minh hoa huy Thread");
    Console.WriteLine("-----");

    ThreadStart childref = new ThreadStart(CallToChildThread);
    Console.WriteLine("Trong Main Thread: tao Thread con.");
    Thread childThread = new Thread(childref);
    childThread.Start();

    //dung Main Thread trong khoang 2 giay
    Thread.Sleep(2000);

    //bay gio huy thread con
    Console.WriteLine("Trong Main Thread: huy Thread con.");

    childThread.Abort();
    Console.ReadKey();
}
```

```
Da luong trong C#
Vi du minh hoa huy Thread
-----
Trong Main Thread: tao Thread con.
Bat dau Thread con!!!
0
1
2
Trong Main Thread: huy Thread con.
Thread Abort Exception!!!
Khong the bat Thread Exception!!!
```



DEMO

- Hiện thực các ví dụ





# **LẬP TRÌNH C# 2**

## **BÀI 8: MULTI-THREADING (P2)**

## ❑ Truyền tham số cho Thread

- ❖ ParameterizedThreadStart là một giải pháp thay thế cho ThreadStart trong trường hợp bạn muốn truyền tham số cho thread.
- ❖ Đối tượng delegate ParameterizedThreadStart này chỉ chấp nhận một tham số kiểu object

## ❑ Truyền tham số cho Thread

```
class Student
{
    2 references
    public string Name { get; set; }
    2 references
    public DateTime BirthDay { get; set; }
}
0 references
static void Main()
{
    Thread t1 = new Thread(Print);
    //chạy thread t1 và truyền tham số cho phương thức Print
    t1.Start(new Student() { Name = "Fpoly", BirthDay = new DateTime(2010, 07, 01) });
    Console.ReadKey();
}
1 reference
static void Print(object obj)
{
    Student st = (Student)obj;
    Console.WriteLine("BirthDay's " + st.Name + ":\t" + st.BirthDay.ToShortDateString());
}
```

## ❑ Truyền tham số cho Thread sử dụng lambda expression

```
class Student
{
    2 references
    public string Name { get; set; }
    2 references
    public DateTime BirthDay { get; set; }
}
0 references
static void Main()
{
    Thread t1 = new Thread((obj) =>
    {
        Student st = (Student)obj;
        Console.WriteLine(st.Name + "\t" + st.BirthDay.ToShortDateString());
    });

    t1.Start(new Student() { Name = "Fpoly", BirthDay = new DateTime(2010, 07, 01) });

    Console.ReadKey();
}
```

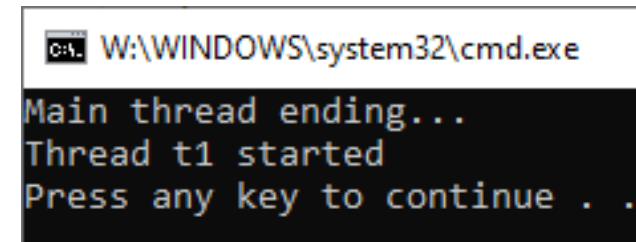


## ❑ Foreground và Background Thread

- ❖ Các thread ban đầu được tạo ra đều là foreground.
- ❖ Ứng dụng sẽ vẫn tiếp tục chạy nếu như tất cả các foreground thread chưa chạy xong mặc dù bạn đã thực hiện lệnh tắt ứng dụng
- ❖ Background thread ứng dụng rất nhiều để thực hiện các tác vụ nền trong ứng dụng
- ❖ Khi thuộc tính `IsBackground` = `true` thì thread đang ở chế độ Background thread

- ❑ Ví dụ foreground thread, thread t1 vẫn tiếp tục chạy mặc dù Main() đã hoàn thành công việc

```
static void Main(string[] args)
{
    Thread t1 = new Thread(() =>
    {
        Thread.Sleep(1000);
        Console.WriteLine("Thread t1 started");
    });
    // t1.IsBackground = true;
    t1.Start();
    Console.WriteLine("Main thread ending...");
}
```



```
C:\ W:\WINDOWS\system32\cmd.exe
Main thread ending...
Thread t1 started
Press any key to continue . .
```

- ❑ Ví dụ Background Thread, thread t1 bị dừng khi main hoàn thành nếu t1.IsBackground = true

```
static void Main(string[] args)
{
    Thread t1 = new Thread(() =>
    {
        Thread.Sleep(1000);
        Console.WriteLine("Thread t1 started");
    });
    t1.IsBackground = true;
    t1.Start();
    Console.WriteLine("Main thread ending...");
}
```

W:\WINDOWS\system32\cmd.exe

Main thread ending...  
Press any key to continue . . .

## ❑ Thread Pooling

- ❖ Thread Pooling là một kĩ thuật cho phép bạn sử dụng các thread hiệu quả hơn bằng cách quản lý và phân phối chúng hợp lý, tận dụng tối đa thời gian nhàn rỗi và tăng hiệu suất của chương trình.
- ❖ Thread pooling là một kĩ thuật được áp dụng phổ biến trong các ứng dụng về I/O bất đồng bộ tập tin và truyền tải dữ liệu trên mạng.
- ❖ Các tác vụ khi được thêm vào Thread pool sẽ được thực thi khi có một thread đang ở trạng thái sẵn sàng
- ❖ Để sử dụng thread pool, bạn chỉ sử dụng phương thức tĩnh `QueueUserWorkItem()` của lớp `ThreadPool`

## ❑ Ví dụ thread Pooling thực thi như hàng đợi

```

public void one(object o)
{
    for (int i = 0; i <= 3; i++)
    {
        Console.WriteLine("One executed");
    }
}

1 reference
public void two(object o)
{
    for (int i = 0; i <= 3; i++)
    {
        Console.WriteLine("Two executed");
    }
}

1 reference
public void three(object o)
{
    for (int i = 0; i <= 3; i++)
    {
        Console.WriteLine("Three executed");
    }
}
  
```

C:\&gt; W:\WINDOWS\system32\cr

```

Two executed
Two executed
Three executed
Three executed
Three executed
Three executed
One executed
One executed
One executed
One executed
Three executed
Three executed
One executed
One executed
One executed
Three executed
Two executed
Two executed
Two executed
Two executed
Two executed
Two executed
Two executed
Three executed
Three executed
Three executed
Three executed
Three executed
  
```

## ❑ Đặt tên cho Thread:

- ❖ Trong lập trình đa luồng bạn có thể chủ động đặt tên cho luồng (thread), nó thực sự có ích trong trường hợp gỡ lỗi (Debugging), để biết đoạn code đó đang được thực thi trong thread nào.
- ❖ Trong một thread bạn có thể gọi ***Thread.CurrentThread.Name*** để lấy ra tên của luồng đang thực thi tại thời điểm đó.

## ❑ Đặt tên cho Thread:

```
public static void LetGo()  
{  
    for (int i = 0; i < 10; i++)  
    {  
        Console.WriteLine("Code of " + Thread.CurrentThread.Name);  
        Thread.Sleep(50);  
    }  
}
```

```
Code of Main  
Create new thread  
Code of Main  
Code of Let's Go  
Code of Main  
Code of Let's Go  
Code of Main  
Code of Main  
Code of Let's Go  
Code of Main  
Code of Let's Go  
Code of Let's Go  
Code of Let's Go  
Code of Let's Go  
Code of Let's Go  
Code of Let's Go  
Code of Let's Go
```

```
public static void Main(string[] args)  
{  
    // Sét đặt tên cho thread hiện thời  
    // (Đang là thread chính).  
    Thread.CurrentThread.Name = "Main";  
    Console.WriteLine("Code of " + Thread.CurrentThread.Name);  
    Console.WriteLine("Create new thread");  
    // Tạo một thread.  
    Thread letgoThread = new Thread(LetGo);  
    // Đặt tên cho thread này.  
    letgoThread.Name = "Let's Go";  
    letgoThread.Start();  
    for (int i = 0; i < 5; i++)  
    {  
        Console.WriteLine("Code of " + Thread.CurrentThread.Name);  
        Thread.Sleep(30);  
    }  
    Console.Read();  
}
```

## □ Độ ưu tiên giữa các Thread

❖ Trong C# có 5 mức độ ưu tiên của một luồng, chúng được định nghĩa trong enum ThreadPriority.

❖ Thông thường với các máy tính tốc độ cao, nếu các luồng chỉ làm số lượng công việc ít, bạn rất khó phát hiện ra sự khác biệt giữa các luồng có ưu tiên cao và luồng có ưu tiên thấp.

**\*\* ThreadPriority enum \*\***

```
enum ThreadPriority {  
    Lowest,  
    BelowNormal,  
    Normal,  
    AboveNormal,  
    Highest  
}
```



- ❑ Độ ưu tiên giữa các Thread, ví dụ dưới đây có 2 luồng, mỗi luồng in ra 100K dòng text (Một số lượng đủ lớn để thấy sự khác biệt).

```
public static void Hello1()
{
    for (int i = 0; i < 100000; i++)
    {
        Console.WriteLine("Hello from thread 1: " + i);
    }
    // Thời điểm thread1 kết thúc.
    endsDateTime1 = DateTime.Now;

    PrintInterval();
}

public static void Hello2()
{
    for (int i = 0; i < 100000; i++)
    {
        Console.WriteLine("Hello from thread 2: " + i);
    }
    // Thời điểm thread2 kết thúc.
    endsDateTime2 = DateTime.Now;

    PrintInterval();
}

private static void PrintInterval()
{
    // Khoảng thời gian (Mili giây)
    TimeSpan interval = endsDateTime2 - endsDateTime1;

    Console.WriteLine("Thread2 - Thread1 = " + interval.TotalMilliseconds + " milliseconds");
}
```

```
private static DateTime endsDateTime1;
private static DateTime endsDateTime2;

public static void Main(string[] args)
{
    endsDateTime1 = DateTime.Now;
    endsDateTime2 = DateTime.Now;

    Thread thread1 = new Thread(Hello1);

    // Sét độ ưu tiên cao nhất cho thread1
    thread1.Priority = ThreadPriority.Highest;

    Thread thread2 = new Thread(Hello2);

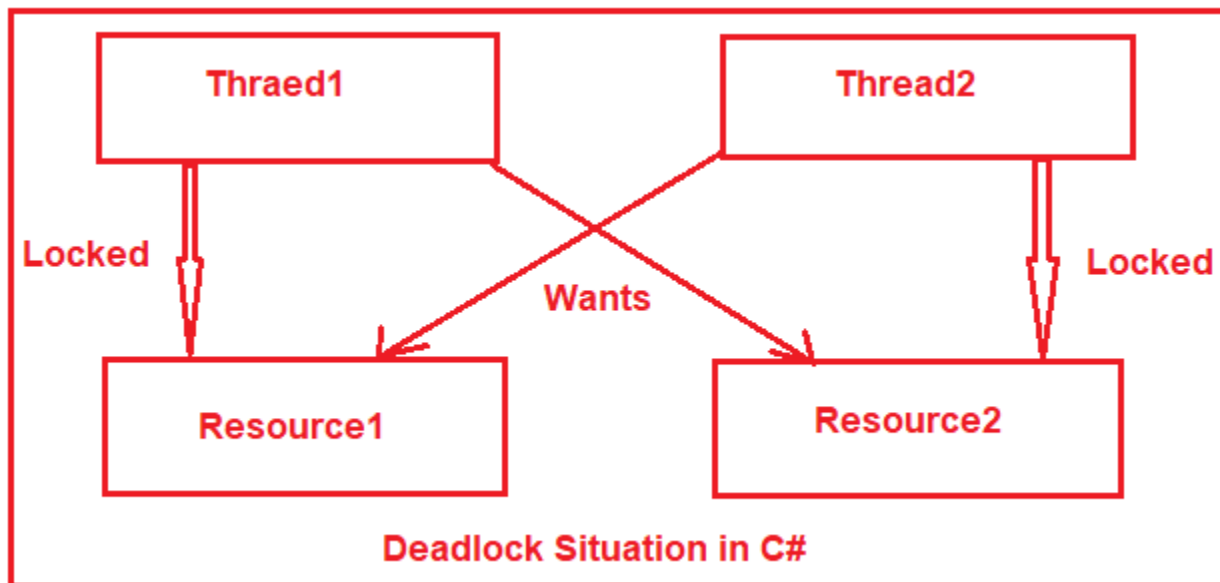
    // Sét độ ưu tiên thấp nhất cho thread2.
    thread2.Priority = ThreadPriority.Lowest;

    thread2.Start(); thread1.Start();

    Console.Read();
}
```

## ❑ Deadlock

- ❖ Đồng bộ hóa khi sử dụng thread là một công việc cần thiết, tuy nhiên nếu không cẩn thận bạn sẽ gặp phải tình trạng chương trình dừng hoạt động vô thời hạn
- ❖ Deadlock xảy ra khi có ít nhất hai thread cùng đợi thread kia giải phóng → 2 thread đợi nhau vô tận



- ❑ Ví dụ Deadlock: cho object "syncObj1" và "syncObj2"
  - ❖ tạo phương thức Foo() sẽ lock "syncObj1" và phương thức Bar() đang cần sử dụng "syncObj1" để hoàn thành thread
  - ❖ tạo phương thức Bar() sẽ lock "syncObj2" và phương thức Foo() đang cần sử dụng "syncObj2" để hoàn thành thread
  - ❖ ➔ Foo() và Bar() chờ nhau vô tận

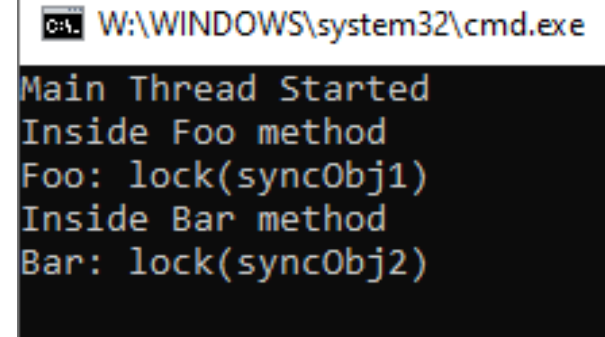
## ❑ Ví dụ Deadlock:

```
static object syncObj1 = new object();
static object syncObj2 = new object();
1 reference
static void Foo()
{
    Console.WriteLine("Inside Foo method");
    lock (syncObj1)
    {
        Console.WriteLine("Foo: lock(syncObj1)");
        Thread.Sleep(100);
        lock (syncObj2)
        {
            Console.WriteLine("Foo: lock(syncObj2)");
        }
    }
}
1 reference
static void Bar()
{
    Console.WriteLine("Inside Bar method");
    lock (syncObj2)
    {
        Console.WriteLine("Bar: lock(syncObj2)");
        Thread.Sleep(100);
        lock (syncObj1)
        {
            Console.WriteLine("Bar: lock(syncObj1)");
        }
    }
}
```

- ❑ Ví dụ Deadlock: câu lệnh lock(syncObj2) của Foo() và lock(syncObj1) của Bar() sẽ không bao giờ được thực hiện vì hai đối tượng syncObj1 và syncObj2 đã bị khóa bởi hai thread khác nhau.

```
static void Main()
{
    Console.WriteLine("Main Thread Started");
    Thread t1 = new Thread(Foo);
    Thread t2 = new Thread(Bar);

    t1.Start();
    t2.Start();
    t1.Join();
    t2.Join();
    Console.WriteLine("Main Thread Completed");
    Console.ReadKey();
}
```



```
C:\> W:\WINDOWS\system32\cmd.exe
Main Thread Started
Inside Foo method
Foo: lock(syncObj1)
Inside Bar method
Bar: lock(syncObj2)
```

- ❑ Tránh Deadlock bằng cách dùng phương thức `Monitor.TryEnter` chỉ định thời gian chờ tối đa và chủ động giải phóng lock.
- ❑ Ví dụ cải tiến phương thức `Bar()` cho chờ 1000ms

```
static void Bar()
{
    Console.WriteLine("Inside Bar method");
    lock (syncObj2)
    {
        Console.WriteLine("Bar: lock(syncObj2)");
        Thread.Sleep(100);
        if (Monitor.TryEnter(syncObj2, 1000))
        {
            try
            {
                Console.WriteLine("Bar: lock(syncObj1)");
            }
            finally
            {
                Monitor.Exit(syncObj2);
            }
        }
    }
}
```

```
A:\E#(Data - NTFS 1)\aptech\
Main Thread Started
Inside Bar method
Bar: lock(syncObj2)
Inside Foo method
Foo: lock(syncObj1)
Bar: lock(syncObj1)
Foo: lock(syncObj2)
Main Thread Completed
```



DEMO

- Hiện thực các ví dụ



## Tổng kết bài học

- ☉ Multithreading

- ☉ Các tính năng Multithreading







**KẾT THÚC**