(T27)討論MultiThread(多執行緒)、Monitor(監視)、Lock(鎖)  
CourseGUID: 29f1196a-1950-41a4-b9c1-dd13a9e92d92  
=======================================================================  
(T27)討論MultiThread(多執行緒)、Monitor(監視)、Lock(鎖)  
=======================================================================  
0. Summary

-----------

1. New Project

1.1. Create New Project : Sample

-----------

2. Sample : Program.cs  
=======================================================================

0. Summary

1.

Thread

1.1.

In computing, a process is an instance of

a computer program that is being executed.

Windows Task Manager provide limit functions

to control the process in Windows operation system.

A process has one main thread and

might have some other threads.

Each tread executes the different piece of code.

For a single core machine,

Asynchronous Programming might reduce performance

because of context-switching.

However, Most machine nowadays normally have multiple cores CPU

which allows Asynchronous Programming provide a way

to run thread/Tasks simultaneously.

1.2.

Thread member.

1.2.1.

ThreadObject.Join()

ThreadObject.Join(int millisecondsTimeout)

ThreadObject.Join(TimeSpan timeout)

Reference:

[https://msdn.microsoft.com/en-us/library/system.threading.thread.join(v=vs.110).aspx](https://msdn.microsoft.com/en-us/library/system.threading.thread.join%28v=vs.110%29.aspx)

Blocks the calling thread until ThreadObject terminates or until timeout.

Return true if the thread has been terminated;

Return false if the thread has not been terminated and time out.

1.2.2.

Thread.IsAlive

Reference:

[https://msdn.microsoft.com/en-us/library/system.threading.thread.isalive(v=vs.110).aspx](https://msdn.microsoft.com/en-us/library/system.threading.thread.isalive%28v=vs.110%29.aspx)

Return true if this thread has been started

and has not terminated normally or aborted;

otherwise, false.

2.

//Stopwatch stopwatch = Stopwatch.StartNew();

//stopwatch.Stop();

//stopwatch.ElapsedMilliseconds

will return the timespan by milliseconds

3.

Get the number of CPU cores.

3.1.

See the Task Manager.

3.2.

In .Net,

Environment.ProcessorCount

3.3.

In command line,

echo %NUMBER\_OF\_PROCESSORS%

=============================================

1. New Project

1.1. Create New Project : Sample

File --> New --> Project... -->

Visual C# -->  **Console App** **(.Net Framework)** -->

Name: **Sample**







=============================================

2. Sample : Program.cs

using System;

using System.Threading;

namespace Sample

{

    class Program

    {

        static void Main(string[] args)

        {

            // 1. =============================================

            // NoThreadNoTask();

            Console.WriteLine("1. NoThreadNoTask(); ====================");

            NoThreadNoTask();

            // 2. =============================================

            // UnprotectedThread();

            Console.WriteLine("2. UnprotectedThread(); ====================");

            UnprotectedThread();

            // 3. =============================================

            // ProtectedThreadV3(), Interlocked

            Console.WriteLine("3. ProtectedThreadV3(), Interlocked ====================");

            ProtectedThreadV3();

            // 4. =============================================

            // ProtectedThreadV4(), lock()

            Console.WriteLine("4. ProtectedThreadV4(), lock() ====================");

            ProtectedThreadV4();

            // 5. =============================================

            // ProtectedThreadV5(), Monitor.Enter

            Console.WriteLine("5. ProtectedThreadV5(), Monitor.Enter ====================");

            ProtectedThreadV5();

            // 6. =============================================

            //ProtectedThreadV6(), Monitor.Enter(Object obj, Boolean lockTaken)

            Console.WriteLine(

                "6. ProtectedThreadV6(), Monitor.Enter(Object obj, Boolean lockTaken) ====================");

            ProtectedThreadV6();

            Console.ReadLine();

        }

        // 1. =============================================

        // NoThreadNoTask();

        static int Total = 0;

        static void SumTo10000()

        {

            for (int i = 1; i <= 10000; i++)

            {

                Total++;

            }

        }

        static void NoThreadNoTask()

        {

            Console.WriteLine("Beginning of NoThreadNoTask()");

            SumTo10000();

            SumTo10000();

            SumTo10000();

            Console.WriteLine("Total = " + Total);

            Console.WriteLine("End of NoThreadNoTask()");

        }

        /\*

        1. NoThreadNoTask(); ====================

        Beginning of NoThreadNoTask()

        Total = 30000

        End of NoThreadNoTask()

        \*/

        // 2. =============================================

        // UnprotectedThread();

        static int Total2 = 0;

        static void SumTo10000V2()

        {

            for (int i = 1; i <= 10000; i++)

            {

                Total2++;

            }

        }

        static void UnprotectedThread()

        {

            Console.WriteLine("Beginning of UnprotectedThread()");

            Thread t1 = new Thread(SumTo10000V2);

            Thread t2 = new Thread(SumTo10000V2);

            Thread t3 = new Thread(SumTo10000V2);

            t1.Start();

            t2.Start();

            t3.Start();

            t1.Join();

            t2.Join();

            t3.Join();

            Console.WriteLine($"Total2=={Total2}");

            Console.WriteLine("End of UnprotectedThread()");

        }

        /\*

        2. UnprotectedThread(); ====================

        Beginning of UnprotectedThread()

        Total2==23489

        End of UnprotectedThread()

        --------------

        A.

        每次跑的時候, 我們都將得到不同的結果

        這裡的23489是目前這次的結果,

        下次再跑的時候, 就不一定是23489

        原因是因為, 這裡的Total2是一個global variable

        Total2同時被t1, t2, t3所共用

        舉例來說, 當t1執行的時候, t1正在把Total2++

        然後"同時"剛好t2想要存取Total2

        但是此時Total2剛好被t1佔用了

        所以, t2沒辦法執行Total2++

        t2只好暫時跳過for loop此輪, 進入下一輪

        因此, 此次的Total2++就被忽略了

        以此類推,

        運氣好的話, Total2最高可達30000

        但是運氣不好的話, Total2會得到30000以下的任何一個數字

        //------

        B.

        由此可知, 我們需要一個Lock來確保永遠可以得到30000

        我們下一點來解釋Lock

        --------------

        Every time we get the different output of total.

        Because int Total field is a shared resource which

        is unprotected from concurrent access by multiple threads.

        Thus, it need lock or interlock for shared resource.

         \*/

        // 3. =============================================

        // ProtectedThreadV3(), Interlocked

        static int Total3 = 0;

        static void SumTo10000V3()

        {

            for (int i = 1; i <= 10000; i++)

            {

                Interlocked.Increment(ref Total3);

            }

        }

        static void ProtectedThreadV3()

        {

            Console.WriteLine("Beginning of ProtectedThreadV3()");

            Thread t1 = new Thread(SumTo10000V3);

            Thread t2 = new Thread(SumTo10000V3);

            Thread t3 = new Thread(SumTo10000V3);

            t1.Start();

            t2.Start();

            t3.Start();

            t1.Join();

            t2.Join();

            t3.Join();

            Console.WriteLine($"Total3=={Total3}");

            Console.WriteLine("End of ProtectedThreadV3()");

        }

        /\*

        3. ProtectedThreadV3(), Interlocked ====================

        Beginning of ProtectedThreadV3()

        Total3==30000

        End of ProtectedThreadV3()

        -----------------------

        A.

        Reference:

<https://docs.microsoft.com/en-us/dotnet/api/system.threading.interlocked?view=netframework-4.8>

        Interlocked.Increment(ref Total3);

        這行的意思就是, 執行Total3++

        只是執行的時候會把Total3給"安全地"鎖住

        讓Total3不會被其他的其他thread所影響

        //------

        B.

        Interlocked會經過Dot Net自己特別的運算機制,

        你不需要太過了解機制是怎麼運作的

        你只需要知道, 通常Interlocked速度很快

        你可以很安心的使用

        -----------------------

        Interlocked is safe and effectively does

        the read, increment, and write in 'one hit'

        which can't be interrupted.

        Because of this it won't affect any other code,

        and you don't need to remember to lock elsewhere either.

        It's also very fast

        (as MSDN says, on modern CPUs this is often literally a single CPU instruction).

        \*/

        // 4. =============================================

        //ProtectedThreadV4(), lock()

        static int Total4 = 0;

        static object \_lockV4 = new object();

        static void SumTo10000V4()

        {

            for (int i = 1; i <= 10000; i++)

            {

                lock (\_lockV4)

                {

                    Total4++;

                }

            }

        }

        static void ProtectedThreadV4()

        {

            Console.WriteLine("Beginning of ProtectedThreadV4()");

            Thread t1 = new Thread(SumTo10000V4);

            Thread t2 = new Thread(SumTo10000V4);

            Thread t3 = new Thread(SumTo10000V4);

            t1.Start();

            t2.Start();

            t3.Start();

            t1.Join();

            t2.Join();

            t3.Join();

            Console.WriteLine($"Total4=={Total4}");

            Console.WriteLine("End of ProtectedThreadV4()");

        }

        /\*

        4. ProtectedThreadV4(), lock() ====================

        Beginning of ProtectedThreadV4()

        Total4==30000

        End of ProtectedThreadV4()

        ----------------------

        A.

        static object \_lockV4 = new object();

        這邊我們用創立一個\_lockV4 object當作我們的lock

        lock (\_lockV4)

        {

            Total4++;

        }

        我們用手動的方式檢查將\_lockV4 object這個lock上鎖

        然後在上鎖期間, 執行Total4++;

        然後才將\_lockV4 object這個lock解鎖

        所以Total4並不會被其他thread影響

        --------

        B.

        Interlocked V.S. Lock

        Interlocked的performance(執行效率)通常會比手動的lock好

        Reference:

<https://stackoverflow.com/questions/154551/volatile-vs-interlocked-vs-lock>

<https://docs.microsoft.com/en-us/dotnet/standard/threading/interlocked-operations>

        B.1.

        Lock通常會"鎖住"其他的thread

        並且期待current thread可以"安全地"

        去update這個共用的global variable.

        但是, "手動上鎖"這個動作, 對效率是很傷的

        B.2.

        所以微軟特別提供Interlocked

        Interlocked會經過Dot Net自己特別的運算機制,

        你不需要太過了解機制是怎麼運作的

        你只需要知道, 通常Interlocked速度很快

        你可以很安心的使用

        B.3.

        Interlocked有許多使用上的限制

        有的時候並不能滿足我們的需求

        所以逼不得已的時候, 還是得使用"手動lock"

        但是如果能使用Interlocked的時候

        請盡量使用Interlocked

        ----------------------

        1.

        Interlocked V.S. Lock

        Interlocked has better performance than lock.

        Reference:

<https://stackoverflow.com/questions/154551/volatile-vs-interlocked-vs-lock>

<https://docs.microsoft.com/en-us/dotnet/standard/threading/interlocked-operations>

        1.1.

        Locking locks all other threads and except current thread

        update the shared field, Total variable, and ensure that

        the Total variable is updated safely,

        However, locking reduce the performance.

        1.2.

        Interlocked is safe and effectively does

        the read, increment, and write in 'one hit'

        which can't be interrupted.

        Because of this it won't affect any other code,

        and you don't need to remember to lock elsewhere either.

        It's also very fast

        (as MSDN says, on modern CPUs this is often literally a single CPU instruction).

        \*/

        // 5. =============================================

        //ProtectedThreadV5(), Monitor.Enter

        static int Total5 = 0;

        static object \_lockV5 = new object();

        public static void SumTo10000V5()

        {

            for (int i = 1; i <= 10000; i++)

            {

                // Acquires the exclusive lock

                Monitor.Enter(\_lockV5);

                try

                {

                    Total5++;

                }

                finally

                {

                    // Releases the exclusive lock

                    Monitor.Exit(\_lockV5);

                }

            }

        }

        static void ProtectedThreadV5()

        {

            Console.WriteLine("Beginning of ProtectedThreadV5()");

            Thread t1 = new Thread(SumTo10000V5);

            Thread t2 = new Thread(SumTo10000V5);

            Thread t3 = new Thread(SumTo10000V5);

            t1.Start();

            t2.Start();

            t3.Start();

            t1.Join();

            t2.Join();

            t3.Join();

            Console.WriteLine($"Total5=={Total5}");

            Console.WriteLine("End of ProtectedThreadV5()");

        }

        /\*

        5. ProtectedThreadV5(), Monitor.Enter ====================

        Beginning of ProtectedThreadV5()

        Total5==30000

        End of ProtectedThreadV5()

        -----------------

        A.

        lock其實是Monitor.Enter的"縮寫語法"

        其實他們都是在做"手動上鎖"這個作用相同的事情

        只是Monitor.Enter這個語法允許你做

        Monitor.Exit(\_lockV5);

        來釋放你的"特定的lock"

        通常Monitor.Enter會搭配try-catch-finally一起使用

<https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/try-catch-finally>

        可以在finally裡面執行

        Monitor.Exit(\_lockV5);

        確保\_lockV5肯定會被"手動解鎖"

        --------

        B.

        lock (\_lockV4)

        {

            Total4++;

        }

        如果你只使用lock語法

        那麼你必須等到系統"自動解鎖"

        系統通常都是火燒屁股的才動手

        這對效率可能會有點影響

        所以建議盡量使用

        Monitor.Enter會搭配try-catch-finally一起使用

        -----------------

        1.

        lock is a shortcut of Monitor.Enter.

        Both are doing the same things.

        Monitor.Enter provide more options to control.

        1.1.

        lock (\_lockV4)

        {

            Total4++;

        }

        1.2.

        Monitor.Enter(\_lockV5);

        try

        {

            Total5++;

        }

        finally

        {

            // Releases the exclusive lock

            Monitor.Exit(\_lockV5);

        }

        \*/

        // 6. =============================================

        //ProtectedThreadV6(), Monitor.Enter(Object obj, Boolean lockTaken)

        static int Total6 = 0;

        static object \_lockV6 = new object();

        public static void SumTo10000V6()

        {

            for (int i = 1; i <= 10000; i++)

            {

                bool lockTaken = false;

                Monitor.Enter(\_lockV6, ref lockTaken);

                try

                {

                    Total6++;

                }

                finally

                {

                    // Releases the exclusive lock

                    if (lockTaken)

                        Monitor.Exit(\_lockV6);

                }

            }

        }

        static void ProtectedThreadV6()

        {

            Console.WriteLine("Beginning of ProtectedThreadV6()");

            Thread t1 = new Thread(SumTo10000V6);

            Thread t2 = new Thread(SumTo10000V6);

            Thread t3 = new Thread(SumTo10000V6);

            t1.Start();

            t2.Start();

            t3.Start();

            t1.Join();

            t2.Join();

            t3.Join();

            Console.WriteLine($"Total6=={Total6}");

            Console.WriteLine("End of ProtectedThreadV6()");

            /\*

            6. ProtectedThreadV6(), Monitor.Enter(Object obj,?Boolean lockTaken) ====================

            Beginning of ProtectedThreadV6()

            Total6==30000

            End of ProtectedThreadV6()

            ---------------------------------

            A.

            Monitor.Enter(Object obj,?Boolean lockTaken)

            這個語法, 又比上面討論的

            Monitor.Enter(\_lockV5);

            又更嚴謹一點, 推薦盡量使用

            Monitor.Enter(Object obj,?Boolean lockTaken)

            ---------

            B.

            bool lockTaken = false;

            Monitor.Enter(\_lockV6, ref lockTaken);

            try

            {

                Total6++;

            }

            finally

            {

                // Releases the exclusive lock

                if (lockTaken)

                    Monitor.Exit(\_lockV6);

            }

            -----

            B.1.

            這邊我們介紹一下

            Monitor.Enter(\_lockV6, ref lockTaken);

            這個語法的固定用法

            -----

            B.2.

            bool lockTaken = false;

            首先, 我們必須要先設立一個bool variable

            \*\*\*也就是lockTaken, 並且value一定要設為false\*\*\*

            -----

            B.3.

            接著, 把這個bool variable也就是lockTaken

            放進Monitor.Enter(\_lockV6, ref lockTaken);

            \*\*\*要確認這邊必須要使用ref這個keyword\*\*\*

            -----

            B.4.

            當\_lockV6這個object還在被"上鎖的時候"

            Monitor.Enter會把ref lockTaken這個變數設為true.

            所以在

            finally

            {

                // Releases the exclusive lock

                if (lockTaken)

                    Monitor.Exit(\_lockV6);

            }

            我們可以判斷

            如果 lockTaken==true

            代表這個\_lockV6 object還正在被上鎖中

            所以我們使用

            Monitor.Exit(\_lockV6);

            來將\_lockV6 object解鎖

            -----

            B.5.

            因為我們有先用if確認是否有上鎖

            如果有上鎖

            才解鎖

            不像之前討論的sample code

            是不分青紅皂白, 不管有沒有上鎖, 都直接解鎖

            因為多了一層if來確認是否上鎖

            所以這種寫法更嚴謹一點,

            請盡量使用

            Monitor.Enter(\_lockV6, ref lockTaken);

            這是此篇tutorial的結論

            ---------------------------------

            1.

            Monitor.Enter(Object obj, Boolean lockTaken)

            1.1.

            Acquires an exclusive lock on the specified object,

            and atomically sets a value that

            indicates whether the lock was taken.

            1.2.

            Boolean lockTaken

            The result of the attempt to acquire the lock, passed by reference.

            The input must be false.

            The output is true if the lock is acquired

            \*/

        }

    }

}

