Synchronisation en C#

Rachid Kadouche

Problème de Synchronisation Exemple1

```
using System;
using System.Threading;
class ThreadTest
{
    static int _val1 = 1, _val2 = 1;
    static void Go()
    {
        if (_val2 != 0)
        {
      //Thread.Sleep(10);
        Console.WriteLine(_val1 / _val2);
        }_val2 = 0;

    }
    static void Main(string[] args)
    {
        Thread worker = new Thread(Go);
        worker.Start();
      //Thread.Sleep(5);
        Go();
    }
}
```

Problème de Synchronisation Exemple2

```
using System;
using System.Threading;
class ThreadSafe
{
    static bool done;
    static void Main()
    {
        Thread t = new Thread(Go);
        t.Start();
        Go();
    }
    static void Go()
    {

     if (!done)
     {
         done = true; Console.WriteLine("Done");
        //Console.WriteLine("Done");done = true; pas le même affichage
     }

    }
}
```

lock

Le mot clé **lock** marque un bloc d'instructions en tant que section critique en assurant le verrouillage par exclusion mutuelle d'un objet particulier,

lock(expression) statement_block

expression
Spécifie l'objet que vous souhaitez verrouiller. expression doit être un type référence.

statement_block
Les instructions de la section critique.

lock permet de s'assurer qu'un thread n'entre pas dans une section critique pendant qu'un autre thread est dans la section critique de code. Si un autre thread tente d'entrer dans un code verrouillé, il attendra (blocage) que l'objet soit libéré.

lock

```
using System;
using System.Threading;
class ThreadSafe
{
    static bool done;
    static readonly object locker = new object();
    static void Main()
    {
      Thread t=  new Thread(Go);
      t.Start();
        Go();
    }
    static void Go()
    {
        lock (locker)
        {
            if (!done) { Console.WriteLine("Done"); done = true; }
        }
    }
}
```

Join

```
using System;
using System.Threading;
namespace TestThread
{
    class Program
    {
        static void Main(string[] args)
        {
            Thread th1 = new Thread(new ParameterizedThreadStart(Afficher));
            Thread th2 = new Thread(new ParameterizedThreadStart(Afficher));
            th1.Start("A");
            //On attend la fin du thread A avant de commencer le thread B.
            th1.Join();
            th2.Start("B");
            Console.ReadKey();
        }
        static void Afficher(object texte)
        {
            for (int i = 0; i < 10000; i++)
            {
                Console.Write((string)texte);
            }
        }
    }
}
```

Join est une méthode de synchronisation qui bloque le thread appelant (autrement dit, le thread qui appelle la méthode) jusqu'à ce que le thread dont ***Join*** est appelée est terminée. Utilisez cette méthode pour vous assurer qu'un thread a été arrêté. L'appelant sera indéfiniment bloqué si le thread ne se termine pas.

Join

```
using System;
using System.Threading;
class ThreadSafe
{
    static void Main()
    {
        Thread t = new Thread(Go);
        t.Start();
        t.Join();
        Console.WriteLine("Thread t has ended!");
    }
    static void Go()
    {
        for (int i = 0; i < 1000; i++) Console.Write("y");
    }
}
```

MUTEX

```
using System;
using System.Threading;
class Program
{
    static public Mutex mutex;
    class ProtectedObject
    {
        public void CriticalSection()
        {
           //L'ensemble d'instructions situées entre mutex.WaitOne() et mutex.ReleaseMutex()
           //est appelé "section critique".
            Console.WriteLine(Thread.CurrentThread.Name + " tente d'entrer dans la section critique");
            //le thread courrant empèche les autres thread d'accéder à la section critique
            mutex.WaitOne();  //début de la section critique
            Thread.Sleep(1000);
            Console.WriteLine("\n" + Thread.CurrentThread.Name + " execute la section critique");
 Console.WriteLine("SectionCritique a été exécutée par " + Thread.CurrentThread.Name + " (la pauvre ...)");
            Console.WriteLine(Thread.CurrentThread.Name + " sort de la section critique\n");
            //fin de la section critique
            //le thread courant permet aux autres threads d'accéder à la section critique.
            mutex.ReleaseMutex();
        }
    }
    static void Main(string[] args)
    {
      ProtectedObject obj = new ProtectedObject();
      mutex = new Mutex();
      //création, instanciation, et démarrage de trois threads
      Thread premierThread = new Thread(obj.CriticalSection);premierThread.Name = "le premier thread";
      Thread secondThread =new Thread(obj.CriticalSection); secondThread.Name = "le second thread";
      Thread troisiemeThread =new Thread(obj.CriticalSection);troisiemeThread.Name= "le troisième thread";
      premierThread.Start(); secondThread.Start(); troisiemeThread.Start();
      Console.ReadLine();
    }
}
```

Semaphore

```
using System;
using System.Threading;
class TheClub //
{
    static SemaphoreSlim sem = new SemaphoreSlim(3); // capacité de 3
    static Thread[] t = new Thread[5];
    static void Main()
    {
        for (int i = 0; i < 5; i++)
        {
         t[i] = new Thread(Enter);
         t[i].Start(i+1);
        }
    }
    static void Enter(object id)
    {
        Console.WriteLine(id + " veut entrer");
        sem.Wait();
        Thread.Sleep(1000);
        Console.WriteLine(id + " est dedans!"); // 3 threads
        Thread.Sleep(1000 * (int)id);
        Console.WriteLine(id + " a quitté");
        sem.Release();
    }
}
```

Exercice

En utilisant les mutex, écrire un programme en c# qui génère une situation d'interblocage (deadlock).

Solution

```
using System;
using System.Threading;
class TheClub //
{
    static Mutex m1,m2;
    static Thread t1,t2;
    static void Main()
    {
        m1 = new Mutex();
        m2 = new Mutex();
        t1 = new Thread(fct1);
        t1.Start();
        t2 = new Thread(fct2);
        t2.Start();

    }
    static void fct1()
    {
        m1.WaitOne();
        Thread.Sleep(1000);
        m2.WaitOne();
        m1.ReleaseMutex();
        m2.ReleaseMutex();
    }
    static void fct2()
    {
        m2.WaitOne();
        Thread.Sleep(1000);
        m1.WaitOne();
        m2.ReleaseMutex();
        m1.ReleaseMutex();
    }

}
```