



Primitivas de Sincronización Comunes: ReaderWriterLockSlim

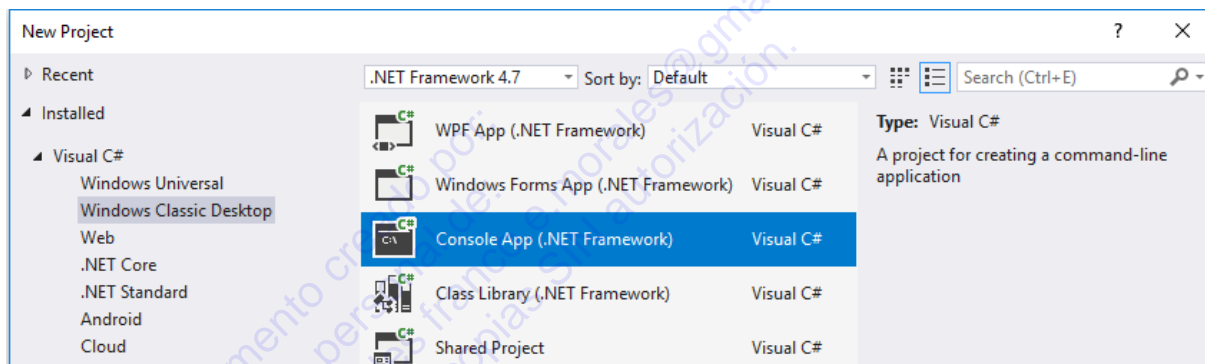
Otra de las primitivas de sincronización comunes que podemos utilizar con la biblioteca *Task Parallel* es a través de la clase **ReaderWriterLockSlim**. La clase **ReaderWriterLockSlim** permite restringir el acceso de escritura sobre un recurso a un hilo a la vez y al mismo tiempo permite que múltiples hilos puedan leer el recurso de forma simultánea.

Ejercicio

Utilizando ReaderWriterLockSlim con Task Parallel Library

En este ejercicio conocerás el uso de la primitiva de sincronización **ReaderWriterLockSlim** soportada por la biblioteca **Task Parallel**.

1. Crea una aplicación de **Consola** utilizando la plantilla **Console App (.NET Framework)**.



2. Agrega la siguiente instrucción al inicio del archivo **Program.cs** para importar el espacio de nombre de la clase **Thread**.

```
using System.Threading;
```

3. Agrega el siguiente código al inicio del archivo **Program.cs** para importar el espacio de nombres que contiene diversas clases colección de hilo-seguro que permiten el acceso concurrente de múltiples hilos de forma segura.

```
using System.Collections.Concurrent;
```

4. Agrega el siguiente código dentro de la clase **Program** para definir un método que nos permita ejemplificar el uso de la primitiva **ReaderWriterLockSlim**.

```
static void UseReaderWriterLockSlim()  
{  
}
```

Supongamos que tenemos un método que procesa pedidos y para cada pedido le asociamos un número de factura. Deseamos que una factura solo pueda ser asociada a un único pedido.



Podemos utilizar el mismo código de procesamiento de pedidos que creamos en un ejercicio anterior.

5. Agrega el siguiente código dentro del método **UseReaderWriterLockSlim**.

```
static void UseReaderWriterLockSlim()
{
    ConcurrentQueue<int> OrdersQueue;
    OrdersQueue = new ConcurrentQueue<int>(Enumerable.Range(1, 10));
    CountdownEvent CDE = new CountdownEvent(10);
    Action ProcessOrder = () =>
    {
        int Order;
        int ProcessedOrders = 0;
        // Mientras haya pedidos por procesar
        while (OrdersQueue.TryDequeue(out Order))
        {
            // Simulamos el procesamiento del pedido
            Console.WriteLine($"Hilo: {Thread.CurrentThread.ManagedThreadId}, ");
            Console.WriteLine($"Pedido: {Order}");
            Thread.Sleep(1000);
            ProcessedOrders++;
            CDE.Signal();
        }
        Console.WriteLine($"Hilo: {Thread.CurrentThread.ManagedThreadId}, ");
        Console.WriteLine($"Pedidos procesados: {ProcessedOrders}");
    };
    Task.Run(ProcessOrder);
    Task.Run(ProcessOrder);
    Task.Run(ProcessOrder);
    CDE.Wait();
    Console.WriteLine("Todos los pedidos han sido procesados.");
}
```

6. Agrega el siguiente código antes de la definición del método **UseReaderWriterLockSlim** para declarar una variable que almacenará el número de factura actual.

```
static int CurrentInvoiceNumber = 0;

static void UseReaderWriterLockSlim()
{
```

7. Modifica el ciclo **while** para simular la asignación de un número de factura a cada pedido procesado.

```
while (OrdersQueue.TryDequeue(out Order))
{
    // Simulamos el procesamiento del pedido
    Console.WriteLine($"Hilo: {Thread.CurrentThread.ManagedThreadId}, ");
    Console.WriteLine($"Pedido: {Order}");

    int NewInvoiceNumber = CurrentInvoiceNumber + 1;
    Console.WriteLine($"Asignando factura {NewInvoiceNumber} a pedido {Order}");
    // Simular una operación de larga duración
```



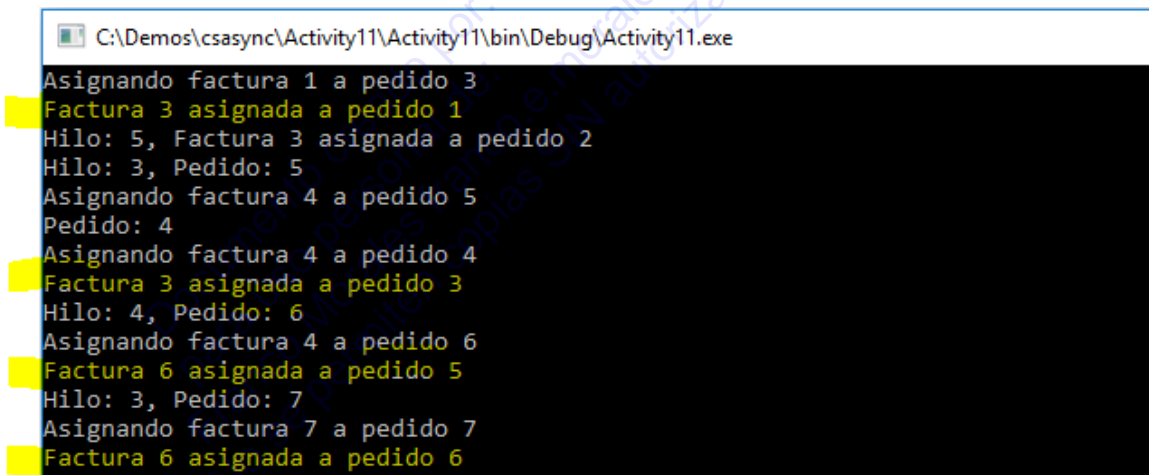
```
Thread.Sleep(500);
CurrentInvoiceNumber++;
// Simular una operación de larga duración
Thread.Sleep(500);
Console.WriteLine(
    $"Factura {CurrentInvoiceNumber} asignada a pedido {Order}");

ProcessedOrders++;
CDE.Signal();
}
```

8. Agrega el siguiente código dentro del método **Main** para invocar al método **UseReaderWriterLockSlim**.

```
static void Main(string[] args)
{
    UseReaderWriterLockSlim();
    Console.Write("Presione <enter> para finalizar...");
    Console.ReadLine();
}
```

9. Ejecuta la aplicación. Puedes notar que hay facturas que son asignadas a más de un pedido.



El acceso concurrente hace que un número de factura pueda ser asignado a más de un pedido, pero lo deseado es que una factura solo esté asociada a un solo pedido.

La clase **ReaderWriterLockSlim** permite restringir el acceso para escritura sobre un recurso a un hilo a la vez al mismo tiempo que permite a múltiples hilos leer el recurso de forma simultánea.

10. Agrega el siguiente código al inicio del método **UseReaderWriterLockSlim** para crear una instancia de **ReaderWriterLockSlim**.

```
ReaderWriterLockSlim RWLS = new ReaderWriterLockSlim();
```



Si un hilo desea leer el recurso, este debe invocar al método **EnterReadLock** del objeto **ReaderWriterLockSlim**. El hilo quedará congelado hasta que pueda conseguir la entrada al bloqueo de solo lectura.

11. Agrega el siguiente código para intentar aplicar un bloqueo de lectura antes de obtener el nuevo posible número de factura.

```
RWLS.EnterReadLock();  
int NewInvoiceNumber = CurrentInvoiceNumber + 1;
```

Después de que el recurso sea leído, el hilo debe invocar al método **ExitReadLock**.

12. Agrega el siguiente código para finalizar el bloqueo de lectura.

```
RWLS.EnterReadLock();  
int NewInvoiceNumber = CurrentInvoiceNumber + 1;  
Console.WriteLine($"Asignando factura {NewInvoiceNumber} a pedido {Order}");  
// Simular una operación de larga duración  
Thread.Sleep(500);  
RWLS.ExitReadLock();
```

Mientras se ejecuta el código entre **RWLS.EnterReadLock()** y **RWLS.ExitReadLock()**, otros hilos pueden solicitar la entrada de un bloqueo de lectura, pero los hilos que soliciten un bloqueo de escritura quedarán en espera (bloqueados).

Si un hilo desea escribir a un recurso, debe invocar al método **EnterWriteLock**.

13. Agrega el siguiente código para esperar a que se pueda aplicar un bloqueo de escritura antes de incrementar el número de factura actual.

```
RWLS.EnterWriteLock();  
CurrentInvoiceNumber++;
```

Si uno o más hilos tienen un bloqueo de lectura sobre el recurso, el método **EnterWriteLock** se congela hasta que todos los bloqueos de lectura sean liberados. Cuando el hilo ha finalizado la escritura sobre el recurso, debe invocar al método **ExitWriteLock**.

14. Agrega el siguiente código para finalizar el bloqueo de escritura.

```
RWLS.EnterWriteLock();  
CurrentInvoiceNumber++;  
// Simular una operación de larga duración  
Thread.Sleep(500);  
Console.WriteLine(  
    $"Factura {CurrentInvoiceNumber} asignada a pedido {Order}");  
RWLS.ExitWriteLock();
```

Las llamadas al método **EnterReadLock** son bloqueadas hasta que el bloqueo de escritura sea liberado. Como resultado, en cualquier momento, un recurso puede ser bloqueado ya sea



por un escritor o por múltiples lectores. Este tipo de bloqueo de lectura/escritura es útil en un rango amplio de escenarios. Por ejemplo, una aplicación bancaria podría permitir a múltiples hilos leer el saldo una cuenta de manera simultánea pero un hilo que requiere modificar el saldo de la cuenta requiere un bloqueo exclusivo.

15. Ejecuta la aplicación y observa el resultado. Puedes notar que ahora un número de factura se asigna a un único pedido.

```
C:\Demos\csasync\Activity11\Activity11\bin\Debug\Activity11.exe
Hilo: 3, Pedido: 1
Asignando factura 1 a pedido 1
Hilo: 4, Hilo: 7, Pedido: 2
Asignando factura 1 a pedido 2
Pedido: 3
Asignando factura 1 a pedido 3
Factura 1 asignada a pedido 1
Hilo: 3, Pedido: 4
Factura 2 asignada a pedido 3
Hilo: 4, Pedido: 5
Factura 3 asignada a pedido 2
Hilo: 7, Pedido: 6
Asignando factura 4 a pedido 6
Asignando factura 4 a pedido 4
Asignando factura 4 a pedido 5
Factura 4 asignada a pedido 6
Hilo: 7, Pedido: 7
Factura 5 asignada a pedido 4
Hilo: 3, Pedido: 8
Factura 6 asignada a pedido 5
Hilo: 4, Pedido: 9
Asignando factura 7 a pedido 9
Asignando factura 7 a pedido 8
Asignando factura 7 a pedido 7
Factura 7 asignada a pedido 7
Hilo: 7, Pedido: 10
Factura 8 asignada a pedido 9
Hilo: 4, Pedidos procesados: 3
Factura 9 asignada a pedido 8
Hilo: 3, Pedidos procesados: 3
Asignando factura 10 a pedido 10
Factura 10 asignada a pedido 10
Hilo: 7, Pedidos procesados: 4
Todos los pedidos han sido procesados.
Presione <enter> para finalizar...
```

Podemos notar que el bloqueo de lectura permitió que otros hilos pudieran leer el número de factura actual por lo que hubo el intento de asignar el mismo número de factura a varios pedidos.



```
Hilo: 3, Pedido: 1  
Asignando factura 1 a pedido 1  
Hilo: 4, Hilo: 7, Pedido: 2  
Asignando factura 1 a pedido 2  
Pedido: 3  
Asignando factura 1 a pedido 3
```

También podemos notar que la escritura de incremento de factura no permitió asignar el mismo número de factura a más de un pedido ya que hubo un bloqueo exclusivo.

En este ejercicio mostramos el uso de la primitiva de sincronización **ReaderWriterLockSlim** soportada por la biblioteca *Task Parallel*.



Para obtener más información sobre la clase **ReaderWriterLockSlim**, se recomienda consultar el siguiente enlace:

ReaderWriterLockSlim Class

<http://go.microsoft.com/fwlink/?LinkID=267853>