



Primitivas de Sincronización Comunes: CountdownEvent

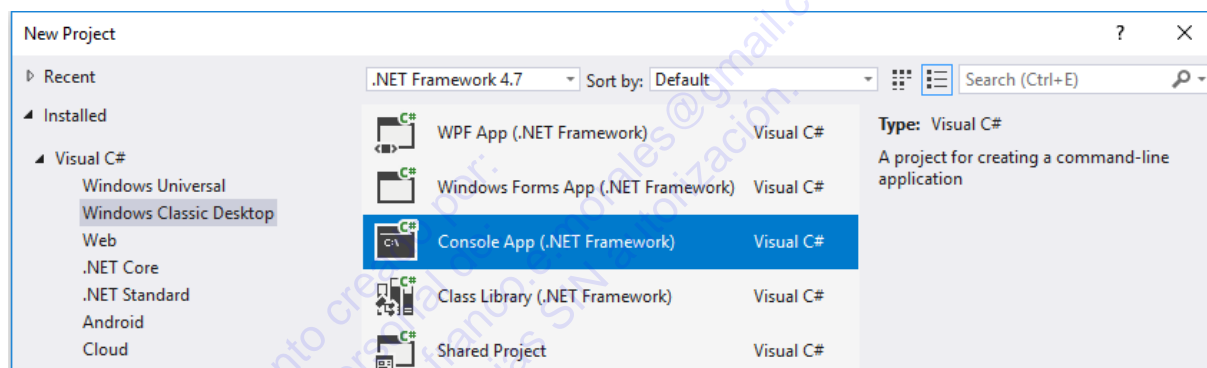
Otra de las primitivas de sincronización comunes que podemos utilizar con la biblioteca *Task Parallel* es a través de la clase **CountdownEvent**. La clase *CountdownEvent* permite bloquear un hilo hasta que otros hilos hayan señalado al objeto *CountdownEvent* un determinado número de veces.

Ejercicio

Utilizando CountdownEvent con Task Parallel Library

En este ejercicio conocerás el uso de la primitiva de sincronización *CountdownEvent* 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 dentro de la clase **Program** para definir un método que nos permita ejemplificar el uso de la primitiva **CountdownEvent**.

```
static void UseCountdownEvent ()  
{  
}
```

Supongamos que tenemos una aplicación que se encarga de procesar pedidos. Los números de pedidos son almacenados en una cola que permite el acceso concurrente.

4. 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;
```



5. Agrega el siguiente código dentro del método **UseCountdownEvent** para definir un objeto **ConcurrentQueue** que almacenará los números de pedidos que deberán ser procesados.

```
ConcurrentQueue<int> OrdersQueue;
```

6. Agrega el siguiente código dentro del método **UseCountdownEvent** para asignar a la variable **OrdersQueue** una lista de números enteros. Esto permitirá simular 10 números de pedidos consecutivos iniciando con el 1.

```
OrdersQueue = new ConcurrentQueue<int>(Enumerable.Range(1, 10));
```

7. Agrega el siguiente código dentro del método **UseCountdownEvent** para simular la acción que procesa los pedidos.

```
Action ProcessOrder = () =>
{
    int Order;
    int ProcessedOrders = 0;
    // Mientras haya pedidos por procesar
    while (OrdersQueue.TryDequeue(out Order))
    {
        // Simulamos el procesamiento del pedido
        Console.Write($"Hilo: {Thread.CurrentThread.ManagedThreadId}, ");
        Console.WriteLine($"Pedido: {Order}");
        Thread.Sleep(1000);
        ProcessedOrders++;
    }
    Console.Write($"Hilo: {Thread.CurrentThread.ManagedThreadId}, ");
    Console.WriteLine($"Pedidos procesados: {ProcessedOrders}");
};
```

8. Agrega el siguiente código dentro del método **UseCountdownEvent** para lanzar 3 hilos que procesen los pedidos de forma simultánea y mostrar un mensaje cuando los pedidos hayan sido procesados.

```
Task.Run(ProcessOrder);
Task.Run(ProcessOrder);
Task.Run(ProcessOrder);

Console.WriteLine("Todos los pedidos han sido procesados.");
```

9. Agrega el siguiente código dentro del método **Main** para invocar al método **UseCountdownEvent**.

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



10. Ejecuta la aplicación. Puedes notar que el mensaje que indica que todos los pedidos han sido procesados es mostrado sin que esto sea realmente cierto.

```
C:\Demos\csasync\Activity10\Activity10\bin\Debug\Activity10.exe
Todos los pedidos han sido procesados.
Presione <enter> para finalizar.
Hilo: 5, Pedido: 1
Hilo: 3, Pedido: 3
Hilo: 4, Pedido: 2
Hilo: 4, Pedido: 4
Hilo: 5, Pedido: 6
Hilo: 3, Pedido: 5
Hilo: 5, Pedido: 7
Hilo: 4, Pedido: 8
Hilo: 3, Pedido: 9
Hilo: 5, Pedido: 10
Hilo: 3, Pedidos procesados: 3
Hilo: 4, Pedidos procesados: 3
Hilo: 5, Pedidos procesados: 4
```

Si queremos que el mensaje aparezca solo cuando se hayan procesado todos los pedidos, podemos utilizar la primitiva **CountdownEvent**. Esta clase permite bloquear un hilo hasta que otros hilos hayan señalado al objeto **CountdownEvent** un determinado número de veces. Cuando creamos un objeto **CountdownEvent**, especificamos un valor entero inicial que especifica el número de señalamientos requeridos, esto es útil debido a que nuestro código puede establecer dinámicamente el valor inicial del contador dependiendo de cuanto trabajo tiene que ser realizado.

11. Agrega el siguiente código después de la inicialización del objeto **ConcurrentQueue** para crear un objeto **CountdownEvent** que espere 10 señalamientos (un señalamiento por cada pedido procesado).

```
static void UseCountdownEvent()
{
    ConcurrentQueue<int> OrdersQueue;
    OrdersQueue = new ConcurrentQueue<int>(Enumerable.Range(1, 10));
    CountdownEvent CDE = new CountdownEvent(10);
}
```

Cuando un hilo complete una operación, puede invocar al método **Signal** del objeto **CountdownEvent** para decrementar el contador.

12. Agrega el siguiente código para decrementar el valor del contador una vez que un pedido haya sido procesado.

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



Cualquier hilo que invoque al método **Wait** del objeto **CountdownEvent** es bloqueado hasta que el contador llegue a cero.

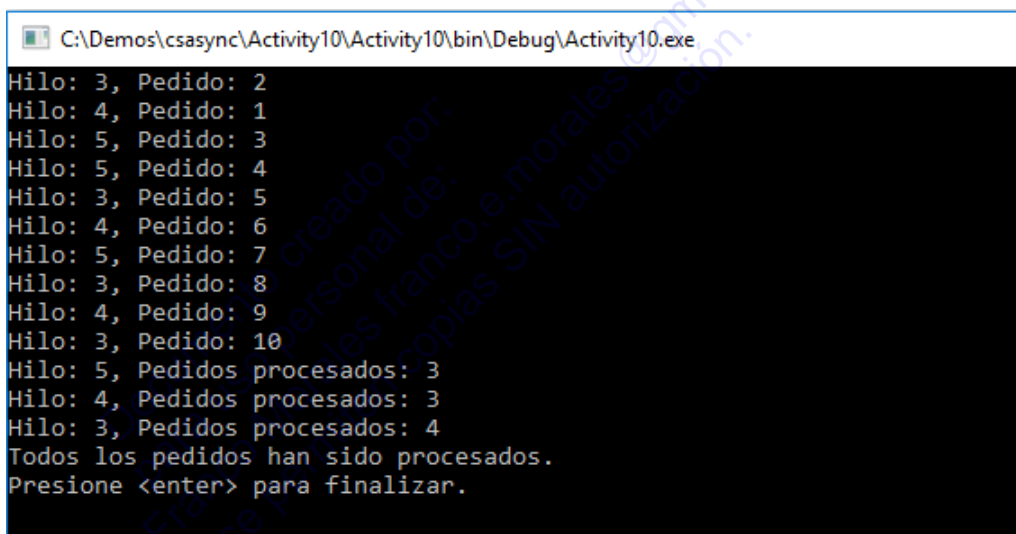
13. Agrega el siguiente código para esperar a que el contador de **CountdownEvent** llegue a cero. Esto indicará que todos los pedidos han sido procesados.

```
Task.Run(ProcessOrder);  
Task.Run(ProcessOrder);  
Task.Run(ProcessOrder);
```

```
CDE.Wait();
```

```
Console.WriteLine("Todos los pedidos han sido procesados.");
```

14. Ejecuta la aplicación. Puedes notar que el mensaje que indica que todos los pedidos han sido procesados es mostrado correctamente después de que efectivamente todos los pedidos hayan sido procesados.



```
C:\Demos\csasync\Activity10\Activity10\bin\Debug\Activity10.exe  
Hilo: 3, Pedido: 2  
Hilo: 4, Pedido: 1  
Hilo: 5, Pedido: 3  
Hilo: 5, Pedido: 4  
Hilo: 3, Pedido: 5  
Hilo: 4, Pedido: 6  
Hilo: 5, Pedido: 7  
Hilo: 3, Pedido: 8  
Hilo: 4, Pedido: 9  
Hilo: 3, Pedido: 10  
Hilo: 5, Pedidos procesados: 3  
Hilo: 4, Pedidos procesados: 3  
Hilo: 3, Pedidos procesados: 4  
Todos los pedidos han sido procesados.  
Presione <enter> para finalizar.
```

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



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

CountdownEvent Class

<http://go.microsoft.com/fwlink/?LinkID=267852>