C# OO Programmeren

LES 10

JOHAN DONNÉ

Dienstmededeling

- Volgende week: Async programming
- Laatste week: verzoeknummer(s) en/of extra uitleg
- Afwerken blok 2: einde kerstvakantie scores tot 10 januari

Overzicht

Task synchronization
Concurrent collections
Parallel loops

Task Synchronisation

Task – interferentie

```
public void Run()
{
    const long MAX = 500000;
    int Count = 0;

    var job = Task.Run(() =>
        {
            for (long i = 0; i < MAX; i++)
            {
                Count--;
            }
        });

    for (long i = 0; i < MAX; i++)
    {
        Count++;
    }

    job.Wait();
    Console.WriteLine("Count Value: " + Count.ToString());
}
```

Task interferentie:

Voorbeeld van probleem:

Count ++ :

1. register = Count
2. register = register + 1
3. Count = register

Count -- :

1. register = Count
2. register = register - 1
3. Count = register

Beide task starten met dezelfde ‘Count’ waarde,
Rechter task eindigt laatst ⇒ overschrijft resultaat van linker.

Task synchronisatie - Lock

```
private void LockDemo()
{
   const long MAX = 500000;
   int Count = 0;
   object mutex = new object();

   var job = Task.Run(() =>
       {
           for (long i = 0; i < MAX; i++)
           {
              lock (mutex)
              {
                  Count--;
              }
           }
        });

   for (long i = 0; i < MAX; i++)
   {
      lock (mutex)
      {
          Count++;
      }
   }

   job.Wait();
   Console.WriteLine("Count Value (with lock): " + Count.ToString());
 }
```

Task synchronisatie - Lock

Lock:

- gemeenschappelijk object (om het even welk)

```
object mutex = new object();
```

- ‘kritische sectie’ afschermen:

```
lock (mutex)
{
    ...
}
```

- ‘kritische sectie’ kan enkel betreden worden als geen ander task een lock heeft op zelfde object. Anders: wachten…

Task synchronisatie - Lock

Opletten met locks:

- Kritische sectie zo klein/kort mogelijk houden
  (risico om andere tasks onnodig te blokkeren).

- Opletten voor 'Deadlock':

```
lock(a)
{
    lock (b)
    {
        ...
    }
}
```

```
lock(b)
{
    lock (a)
    {
        ...
    }
}
```

Concurrent collections

Immutable collections

Immutable collections kunnen na het aanmaken niet meer gewijzigd worden.

⇨ Geen wijzigingen mogelijk, dus geen synchronisatie nodig

Systems.Collections.Immutable (.Net Core only!!!)

- ImmutableList<T>
- ImmutableHashSet<T>
- ImmutableSortedSet<T>
- ImmutableDictionary<TKey, TValue>
- ImmutableSortedDictionary<TKey, TValue>
- ImmutableQueue<T>
- ImmutableStack<T>

Concurrent collections

Ook bij Add, Remove… van collections is interferentie mogelijk

⇨ 'Thread safe' versies van enkele collections

Systems.Collections.Concurrent

BlockingCollection<T>
A collection that can have a limited capacity with blocking 'Add' and 'Take' methods

ConcurrentDictionary<TKey, TValue>
A thread-safe implementation of 'Dictionary<TKey, TValue>'

ConcurrentQueue<T>
A thread-safe implementation of 'Queue<T>'

ConcurrentStack<T>
A thread-safe implementation of 'Stack<T>'

ConcurrentBag<T>
A ConcurrentBag represents a thread safe set of unordered objects that can contain duplicates.

Parallel loops

Parallel Loops

Versie van for & foreach die de verschillende iteraties over verschillende tasks uitvoert.

Algemene vormen:

```
Parallel.For(int start, int exclusiveEnd, Action<int>)

Parallel.ForEach(IEnumerable<T>, Action<T>)
```

Parallel Loops

Voorbeeld:

```
for (int i = 0; i < list.Count; i++)
{
   Console.WriteLine(list[i]);
}

Parallel.For(0, list.Count, (i) =>
{
   Console.WriteLine(list[i]);
});

Parallel.ForEach(list, (element) =>
{
   Console.WriteLine(element);
});
```

Parallel Loops - Break

Parallele lus afbreken: ParallelLoopState parameter bij Action.

```
var array = new int[30];
Parallel.For(0, array.Length, (index, state) =>
{
    if (index == 5) state.Stop();
    array[index] = index;
    Console.Write(" " + index.ToString());
});
Console.WriteLine();
```

State.Stop(): geen nieuwe threads meer gescheduled,
lopende worden afgewerkt.

Parallel Loops

Opgelet:

Geen enkele garantie over volgorde waarin de verschillende iteraties uitgevoerd worden!

```
var list = new List<int>() { 1, 2, 3, 4, 5, 6, 7, 8, 9 };

Parallel.For(0, list.Count, (i) =>
{
   Console.Write($"{list[i]},");
});
```

```
1,3,4,7,8,9,5,2,6,
```

Parallel Loops

Geen enkele garantie over volgorde waarin de verschillende iteraties uitgevoerd worden!

Dus ‘state.Stop()’ bij i = 5 betekent niet dat de lus niet uitgevoerd wordt voor i = 6 . . 9!

```
Parallel.For(0, 30, (index, state) =>
{
    Console.Write(" " + index.ToString());
    if (index == 5) state.Stop();
});
```

```
0 1 12 18 19 20 22 23 2 4 5 9 24 13 25 3 21 15 6
```

Demo – ‘Roll the Dice’

Roll The Dice

Simulatie van het gooien van 3 teerlingen.

Berekenen van kans om een waarde (3 .. 18) te gooien door random te gooien en te tellen hoe vaak elke waarde voorkomt.