

C# programmering basis tråd øvelser

Øvelse 0

Skriv nedenstående program af og kommenter det. Linje 28 og 29 kan erstattes med

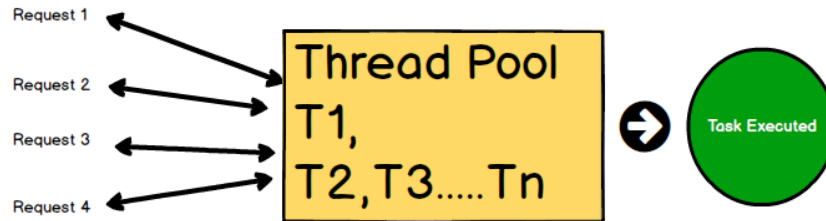
```
ThreadPool.QueueUserWorkItem(tpd.task1);  
ThreadPool.QueueUserWorkItem(tpd.task2);
```

Hvad er forskellen?

```
1. /*  
2.  * C# Program to Create Thread Pools  
3.  */  
4. using System;  
5. using System.Threading;  
6. class ThreadPoolDemo  
7. {  
8.     public void task1(object obj)  
9.     {  
10.         for (int i = 0; i <= 2; i++)  
11.         {  
12.             Console.WriteLine("Task 1 is being executed");  
13.         }  
14.     }  
15.     public void task2(object obj)  
16.     {  
17.         for (int i = 0; i <= 2; i++)  
18.         {  
19.             Console.WriteLine("Task 2 is being executed");  
20.         }  
21.     }  
22.  
23.     static void Main()  
24.     {  
25.         ThreadPoolDemo tpd = new ThreadPoolDemo();  
26.         for (int i = 0; i < 2; i++)  
27.         {  
28.             ThreadPool.QueueUserWorkItem(new WaitCallback(tpd.task1));  
29.             ThreadPool.QueueUserWorkItem(new WaitCallback(tpd.task2));  
30.         }  
31.  
32.         Console.Read();  
33.     }  
34. }
```

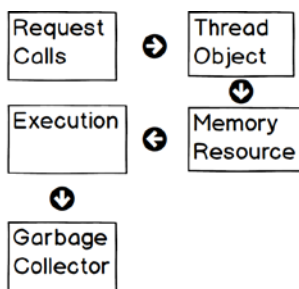
Tråd Livscyklus

Før du skal arbejde med threads pools, skal du forstå en livscyklus for en tråd. Når .NET-frameworket modtager anmodning om et metodeopkald, danner frameworket et nyt tråd objekt, der allokeres RAM, og derefter udføres opgaven.



Thread Pooling

Når opgaven er færdig, fjerner garbage collector tråd objektet for at frigøre af hukommelse.



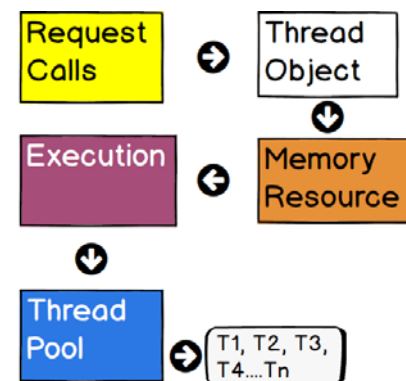
Hver gang der sker en ny anmodning/request til en metode skaber frameworket altså et nyt trådobjekt, der allokeres i hovedlageret. Hvis der er mange anmodninger, vil der være mange tråd objekter. Mange trådobjekter kan blive en belastning for hukommelsen og dermed kan applikationen blive langsom.

For at overkomme ovenstående problem, kan vi selv styre tråd objekterne i en thread pool i stedet for at benytte standardmetoden til tråd fremstilling.

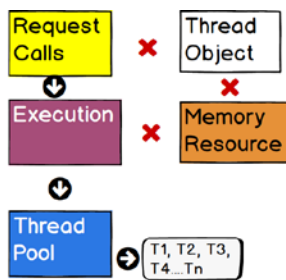
Thread Life Cycle

Thread pool i C#

Thread poolen er en samling af tråde, der kan benyttes til at udføre opgaver i baggrunden. Når en tråd fuldfører sin opgave, sendes tråden tilbage i puljen af ventende tråde, hvor den så kan genbruges. Med denne genbrugelighed undgås det at en applikation skaber for mange tråde, og herved gøres det muligt at reducere hukommelsesforbruget. Tråden bliver altså ikke "ryddet af vejen" af garbage collector



Thread Pool Life Cycle



Hvis der kommer nye opgaver (metodeklad) vil .NET Framework, tildele opgaven til en tråd i thread pool

Fordelen ved at arbejde med en puljer af tråde er mindre hukommelsesforbrug til et stort antal opgaver og derved hurtigere eksekvering af opgaver.

Thread Pool Life Cycle

Øvelse 1

Denne øvelse er en lille tutorial, hvor der oprettes en thread pool som vi styre til en given opgave samt hvor frameworket styre den samme opgave. Til sidst tager vi tid på disse to måder af udføre den samme opgave.

Trin 1

For at implementere thread pool i en applikation skal vi bruge Threading namespace som vist i nedenstående kode.

```
using System.Threading;
```

Trin 2

Efter brug af threading namespace skal vi kalde threadpool klassens metoden "QueueUserWorkItem" – styre kø funktionalitet samt tråde.

```
ThreadPool.QueueUserWorkItem(Process);
```

Her er Process den metode, der skal udføres af en tråd i puljen

Trin 3

I dette trin vil vi sammenligne, hvor meget tid det tager et ikke-tråd-pulje-tråd-objekt (traditionelt tråd objekt), og hvor lang tid tager en tråd fra vore pulje til at udføre en opgave (metode). Til dette forsøg kaldes to ens metoder kaldet ProcessWithThreadMethod () og ProcessWithThreadPoolMethod () og inde i metoderne afvikles et for-loop 10 gange. Disse to metoder vil kalde en simpel metode kaldet Process()

```
static void Process()
{
}
}
```

Som du ser, er dette bare en tom metode, og denne metode kaldes af de to andre metoder.

Trin 4

De to metoder kodes på nedenstående måde

```
static void ProcessWithThreadMethod()  
{  
    for (int i = 0; i <= 10; i++)  
    {  
        Thread obj = new Thread(Process);  
        obj.Start();  
    }  
}
```

```
static void ProcessWithThreadPoolMethod()  
{  
    for (int i = 0; i <= 10; i++)  
    {  
        ThreadPool.QueueUserWorkItem(Process);  
    }  
}
```

Trin 5

Vores hoveddel er klar nu, vi skal bare kalde disse metoder i vores main metode. Da vi tester tidsforbrug kan vi kalde disse metoder mellem stopur start og slut. Nedenstående er den komplet kode.

```
using System;  
using System.Threading;  
using System.Diagnostics;  
  
namespace ThreadPooling  
{  
    class Program  
    {  
        static void Main(string[] args)  
        {  
            Stopwatch mywatch = new Stopwatch();  
            Console.WriteLine("Thread Pool Execution");  
            mywatch.Start();  
            ProcessWithThreadPoolMethod();  
        }  
    }  
}
```

```
        mywatch.Stop();

        Console.WriteLine("Time consumed by ProcessWithThreadPoolMethod is : " + mywatch.ElapsedTicks.ToString());
        mywatch.Reset();
        Console.WriteLine("Thread Execution");

        mywatch.Start();
        ProcessWithThreadMethod();
        mywatch.Stop();

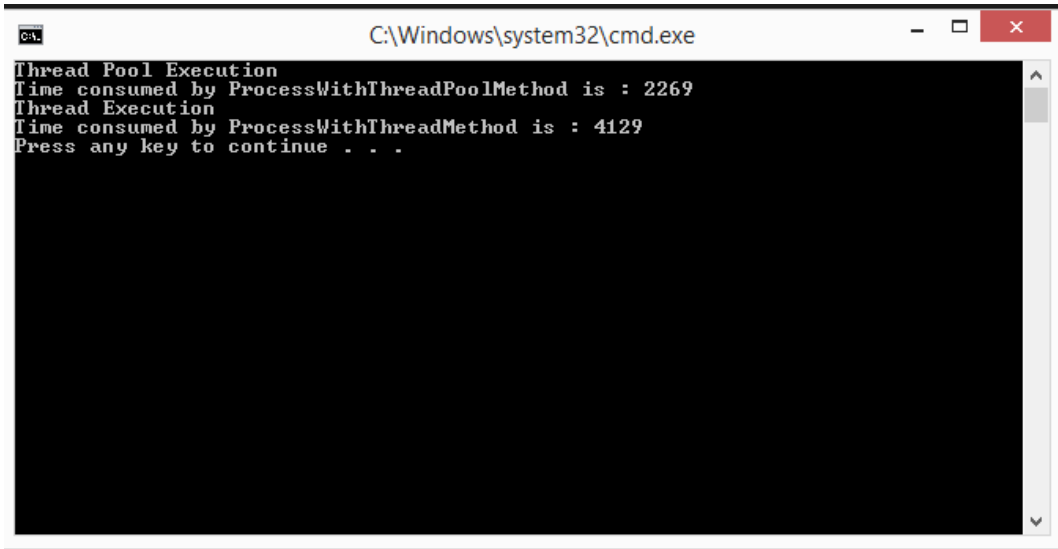
        Console.WriteLine("Time consumed by ProcessWithThreadMethod is : " + mywatch.ElapsedTicks.ToString());
    }

    static void ProcessWithThreadPoolMethod()
    {
        for (int i = 0; i <= 10; i++)
        {
            ThreadPool.QueueUserWorkItem(Process);
        }
    }

    static void ProcessWithThreadMethod()
    {
        for (int i = 0; i <= 10; i++)
        {
            Thread obj = new Thread(Process);
            obj.Start();
        }
    }

    static void Process(object callback)
    {
        }
    }
}
```

Resultatet på Mikkel computer



```
C:\Windows\system32\cmd.exe
Thread Pool Execution
Time consumed by ProcessWithThreadPoolMethod is : 2269
Thread Execution
Time consumed by ProcessWithThreadMethod is : 4129
Press any key to continue . . .
```

Konklusion

Ovenstående output, viser tidsforbrug med metoden `ProcessWithThreadPoolMethod` er 2269ms og tidsforbrug med metoden `ProcessWithThreadMethod` er 4129ms der en stor forskel mellem tidsforbrug og hukommelsesforbrug på de to metoder – Husk på metoden `Process()` er tom .

Skal metoden `Process()` tage et `object` som argument, begrund dit svar?

```
static void Process(object callback)
{
}
}
```

Øvelse 2

Udvid øvelse 1 prøv at indsætte nedenstående kode i `Process()` -metoden

```
for (int i = 0; i < 100000; i++)
{
}
}
```

Hvad sker der med eksekveringstiden?

Prøv nu at indsætte nedenstående kode i `Process()`

```
for (int i = 0; i < 100000; i++)
{
    for (int j = 0; j < 100000; j++)
    {
    }
}
}
```

Hvad sker der med eksekveringstiden?

Øvelse 3

Lav en konsol applikation der benytter en eller flere thread pool(s).

Udskriv nedenstående poperties på de enkelte tråde, prøv at ændre på set egenskaben og se hvad der sker.

Poperties	Beskrivelse
bool IsAlive	IsAlive er en get-property der fortæller om en tråd er i live
bool IsBackground	IsBackground er både set- og get-property. En baggrunds-tråd kan køre videre efter programmet er termineret
ThreadPriority Priority	<p>Priority er både set- og get-property. ThreadPriority er en enumeration, der erklærer fem prioriteter. Disse er (med den højeste prioritet først):</p> <ul style="list-style-type: none"> • Highest • AboveNormal • Normal • BelowNormal • Lowest <p>Vha. Priority kan man aflæse og indstille den prioritet hvormed tråden kører - default er Normal.</p>

Udvid applikationen og afprøv de forskellige metoder. Beskriv i tabellen, hvad metodernes funktionalitet er

Metode	Beskrivelse
Start()	
Sleep()	
Suspend()	
Resume()	
Abort()	
Join()	