

# C# ile Threading işlemleri

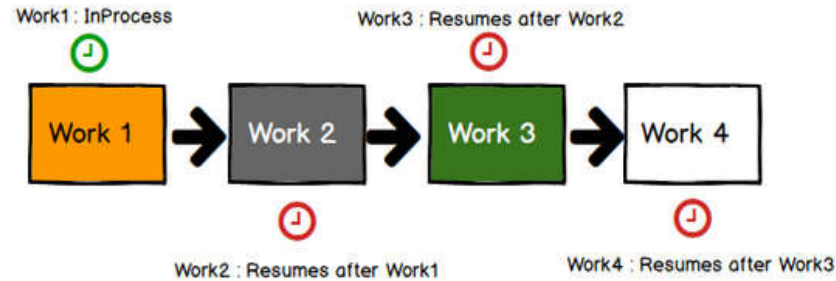
- Thread'ler sayesinde birçok işi aynı anda eş zamanlı olarak yapmak mümkündür. Yürütülen iş parçacıklarını bir süre bekletmek veya istenen anda sonlandırmakta mümkündür.
- Threading işlemlerini yöneten tipler, .Net Framework tarafında **System.Threading** alanında barınmaktadır.
- Threading konusuna başlamadan önce **Thread** sınıfını tanımak gerekmektedir. Bu sınıf tek bir iş parçacığı nesnesidir. Yani bir iş parçacığını başlatmayı ve süreci yönetmeyi sağlar.

# C# ile Threading işlemleri

- Windows'ta yeni bir uygulama başlatıldığında, uygulama için işlem kimliğine sahip bir işlem oluşturur ve bu yeni işleme bazı kaynaklar ayrılır. Her process uygulamanın başlangıç noktasını göz önüne alan en az bir iş parçacığı içerir.
- Ortak dil çalışma zamanı (CLR), iş parçacıklarının yaşam döngüsünün oluşturulmasında ve yönetilmesinde önemli bir rol oynar. CLR, main metodunu kullanarak uygulama kodunu yürütmek için tek bir ön plan iş parçacığı oluşturur. Bu iş parçacığına birincil veya main iş parçacığı denir. Bu ana iş parçacığı ile birlikte, bir işlem kodun bir bölümünü yürütmek için bir veya daha fazla iş parçacığı oluşturabilir. CLR tarafından yönetilen çalışan iş parçacıklarında kodu yürütmek için ThreadPool sınıfını kullanabilir. Bir C # programının giriş noktası Main yöntemde başlar ve bu birincil iş parçacığının yoludur.

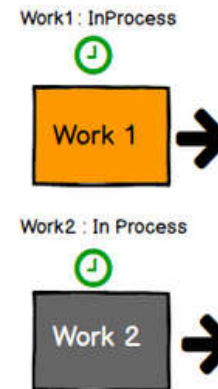
# C# ile Thread işlemleri

Synchronous way means where work multiple jobs are executed one after the other. Here Work 2 have to wait till Work 1 is completed same way the others as shown in below image.



## Synchronous way

Asynchronous means multiple work has been executed simultaneously like doing multitask at a same time.

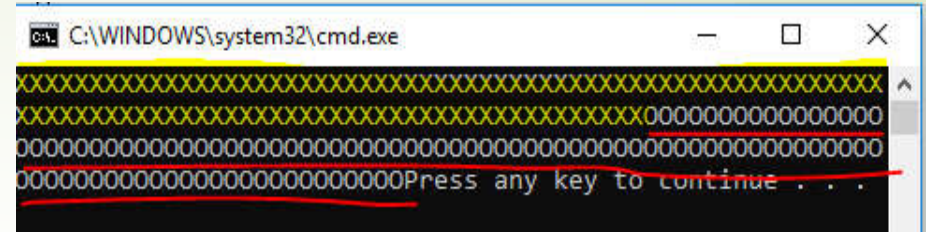


## Asynchronous way

# C# ile Thread işlemleri

- Adım 1: Senkron çalışma

```
static void Main(string[] args)
{
    Islem1(); //senkron işlem sonucu Islem1 bitmeden islem2
    baslamayacaktır.
    Islem2();
}
static public void Islem1()
{
    for (int i = 0; i < 100; i++)
    {
        Console.Write("X");
    }
}
static public void Islem2()
{
    for (int i = 0; i < 100; i++)
    {
        Console.Write("O");
    }
}
```



# C# ile Thread işlemleri-Start()

Yeni bir iş parçacığı yaratmak için Thread sınıfından bir nesne türetmemiz gerekiyor. Bunun için de thread sınıfının yaratıcısına, iş parçacığı başladığında çalıştırılacak metodu gösteren bir delege geçmemiz gerekiyor.

```
Thread th = new Thread (new ThreadStart(CalistirilacakMetod));
```

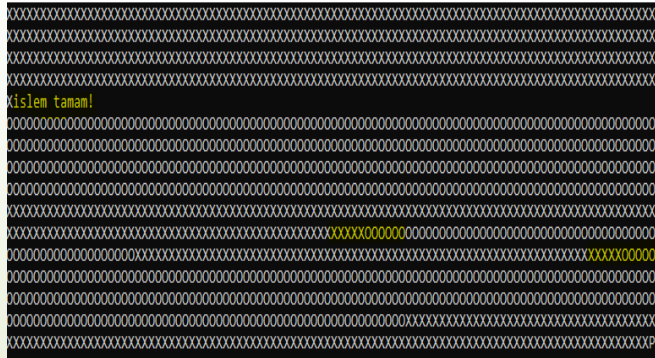
iş parçacığımızı yarattıktan sonra, iş parçacığımızın **start** metodunu kullanarak, iş parçacığımızı çalışmaya başlatıyoruz.

```
th.Start();
```

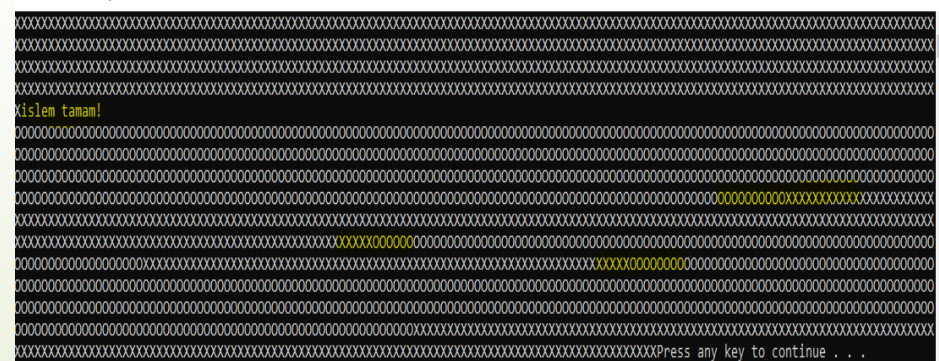
# C# ile Thread işlemleri-Start()

- Adım 2: Senkron çalışma

```
static void Main(string[] args)
{
    Thread tIslem1 = new Thread(Islem1); //asenkron olarak thread
    tanımlanıyor
    Thread tIslem2 = new Thread(Islem2);
    tIslem1.Start(); //iş parçacığı başlatıldı
    tIslem2.Start();
    Console.WriteLine("islem tamam!");
}
static public void Islem1()
{
    for (int i = 0; i < 1000; i++)
    {
        Console.Write("X");
    }
}
static public void Islem2()
{
    for (int i = 0; i < 1000; i++)
    {
        Console.Write("O");
    }
}
```



6





# C# ile Thread işlemleri-Join()

- Bir threadin başka bir thread bitmeden çalışmamasını istiyorsak threadleri birleştirmeliyiz. Bunun için Thread sınıfının **Join** metodunu kullanmalıyız.

```
static void Main(string[] args)
```

```
{
```

```
    Thread tIslem1 = new Thread(Islem1); //asen kron olarak thread
```

```
    tanımlanıyor
```

```
    Thread tIslem2 = new Thread(Islem2);
```

```
    tIslem1.Start(); //iş parçacığı başlatıldı
```

```
    tIslem1.Join(); //iş parçacığı birleştirildi. İşlem bitene kadar
```

```
    sonraki threadler bekleyecek
```

```
    tIslem2.Start();
```

```
    tIslem2.Join();
```

```
    Console.WriteLine("işlem tamam!");
```

```
}
```

```
static public void Islem1()
```

```
{
```

```
    for (int i = 0; i < 100; i++)
```

```
    {
```

```
        Console.Write("X");
```

```
    }
```

```
}
```

```
static public void Islem2()
```

```
{
```

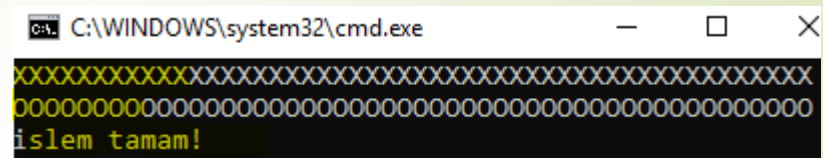
```
    for (int i = 0; i < 100; i++)
```

```
    {
```

```
        Console.Write("O");
```

```
    }
```

```
}
```



# C# ile Thread işlemleri-Sleep(ms)

- Geçerli iş parçacığını belirli bir süre boyunca askıya almak için kullanılan yöntem **Sleep()**'tir. Zaman **milisaniye** veya Zaman Aralığı olarak belirtilebilir. Bir Uyku modundayken bir yöntem hiçbir CPU kaynağını tüketmez, dolaylı olarak diğer iş parçacığı işlemleri için bellek tasarrufu sağlar.
- Örnekte geçen zamanı hesaplamak için **StopWatch** ve **TimeSpan** kullanılmıştır. StopWatch ile geçen zaman hesaplanıp, **Elapsed** değeri TimeSpan tipinde bir değere alınıp kullanılabilir. Stopwatch ve TimeSpan için **System.Diagnostics** alan adı gereklidir.



# C# ile Thread işlemleri-Sleep(ms)

- Örnekte geçen süre tespit edilmiştir. Her thread'in 2 snde bir çalıştığı görülmektedir.

```
using System.Diagnostics;
using System.Threading;
static void Main(string[] args)
{
    Stopwatch stWatch = new Stopwatch();
    stWatch.Start();
    Thread oThread = new Thread(ProcessSleep);
    oThread.Start();
    oThread.Join();
    stWatch.Stop();
    TimeSpan ts = stWatch.Elapsed;

    string gecenSure = String.Format("{0:00}:{1:00}:{2:00}", ts.Hours, ts.Minutes, ts.Seconds);
    Console.WriteLine("Toplam zaman " + gecenSure);
    Console.WriteLine("İşlem tamam..!");
}

static void ProcessSleep()
{
    for (int i = 0; i <= 5; i++)
    {
        Console.WriteLine("İşlem sürüyor..!");
        Thread.Sleep(2000); //2 saniye için işlem durduruluyor
    }
}
```

# C# ile Thread işlemleri-Priority

- Bazı durumlarda çoklu thread kullanırken öncelik vermek istenen iş parçacıkları olabilir. Birçok thread aynı anda start edildiğinde, bunlardan bir tanesi diğerlerine göre öncelikli olması gerekebilir. Bu gibi durumlarda imdadımıza yetişen **Thread.Priority** özelliğidir. Bu özellik bir enumerasyon tipindedir. Bu enum tipi, bünyesinde; *Highest*, *AboveNormal*, *Normal*, *BelowNormal*, *Lowest* gibi seçenekleri tutmaktadır.

```
public void StartMultipleWriter()  
{  
    Thread th1 = new Thread(new ThreadStart(WriteX));  
    Thread th2 = new Thread(new ThreadStart(WriteO));  
  
    th1.Priority = ThreadPriority.Lowest;  
    th2.Priority = ThreadPriority.Highest;  
  
    th1.Start();  
    th2.Start();  
}
```

10

Yani WriteX metodu WriteO metodundan daha az öncelikli olsun. Bu durumda işlemci, WriteO metodunun işlemini daha önce bitirecektir.

# C# ile Thread işlemleri-Abort()

- Tanımlı bir thread'i Abort() metodu ile durdurabiliriz. Fakat bu metodun kullanımıyla ilgili riskler hakkında çeşitli bilgilendirmeler olmaktadır

```
Thread oThread = new Thread(ProcessSleep);  
oThread.Start();  
oThread.Abort();
```

# C# ile Thread işlemleri-Name()

- **Thread.CurrentThread.Name** özelliği ile çalışan thread'in ismini öğrenebiliriz. Thread ismi ayarlamak için **threadName.name = "threadismi"** şeklinde tanımlama yaparız.

Tüm threadler default olarak "foreground (ön plan)" şeklinde tanımlanır. Uygulama tüm foreground threadler tamamlanınca biter. Thread özelliği "background (arka plan)" olarak değiştirilirse, background thread devam etse bile uygulama bitebilir. Bu durumda background thread aniden kesilmiş olacaktır. **threadName.Isbackground = true** biçiminde tanımlama yapılarak özellik değiştirilebilir. **threadName.Join()** metodunu çağırarak tüm threadlerin bitmesini bekleyip programı öyle sonlandırabiliriz.

```
Thread.Sleep (TimeSpan.FromHours (1)); // 1 saat bekle  
Thread.Sleep (5000); // 5 saniye bekle
```

# C# ile Thread işlemleri-Örnek

- Progress bar örneği;
- 3 adet progress barın 3 farklı thread ile asenkron doldurulması.
- *'Çapraz iş parçacığı işlemi geçerli değil: 'progressBar2' denetimine oluşturulduğu iş parçacığı dışında başka bir iş parçacığından erişildi.'* HATASI!
- Bu hatanın temel sebebi *Thread* çakışmaları olmasındandır. Genellikle ASenkron(MultiThread) yapısında olan programların geliştirme süreçlerinde alınan hatadır.
- Bu hatanın çözümü oldukça basittir. Programınızın yüklenme aşamasına aşağıdaki kodu eklemeniz bu sorunu halledecektir.

```
Control.CheckForIllegalCrossThreadCalls = false;
```

# C# ile Thread işlemleri-Örnek

```
using System.Diagnostics;
using System.Threading;

private void button1_Click(object sender, EventArgs e)
{
    Control.CheckForIllegalCrossThreadCalls = false;
    th1 = new Thread(islem1);
    th2 = new Thread(islem2);
    th3 = new Thread(islem3);
    th1.Priority = ThreadPriority.Highest;
    th3.Priority = ThreadPriority.Lowest;
    th1.Start();
    th2.Start();
    th3.Start();
}

public void islem1()
{
    Stopwatch st1 = new Stopwatch();
    st1.Start();
    //Thread.Sleep(1000);
    for (int i = 0; i < 100; i++)
    {
        Thread.Sleep(100);
        progressBar1.Value++;
    }
    st1.Stop();
    label4.Text = st1.Elapsed.ToString();
}
```

```
public void islem2()
{
    Stopwatch st2 = new Stopwatch();
    st2.Start();
    //Thread.Sleep(1000);
    for (int i = 0; i < 100; i++)
    {
        Thread.Sleep(100);
        progressBar2.Value++;
    }
    st2.Stop();
    label5.Text = st2.Elapsed.ToString();
}

public void islem3()
{
    Stopwatch st3 = new Stopwatch();
    st3.Start();
    //Thread.Sleep(1000);
    for (int i = 0; i < 100; i++)
    {
        Thread.Sleep(100);
        progressBar3.Value++;
    }
    st3.Stop();
    label6.Text = st3.Elapsed.ToString();
}
```



# C# ile Thread işlemleri-Örnek

Form1

	Süre
işlem1	-
işlem2	-
işlem3	-

Başlat



Form1

	Süre
işlem1	-
işlem2	-
işlem3	-

Başlat



Form1

	Süre
işlem1	00:00:10.0824625
işlem2	00:00:10.0837442
işlem3	00:00:10.1351237

Başlat

# C# ile Thread işlemleri-Link

- <http://www.csharpnedir.com/articles/read/?id=584&title=C>
- <https://www.bayramucuncu.com/c-ile-threading-islemleri/>
- <http://www.learnsharp tutorial.com/threading-and-types-of-threading-stepbystep.php>

# Thread vs Task

Async programa denilince akla ilk gelen 2 seçenek vardır.

- Task

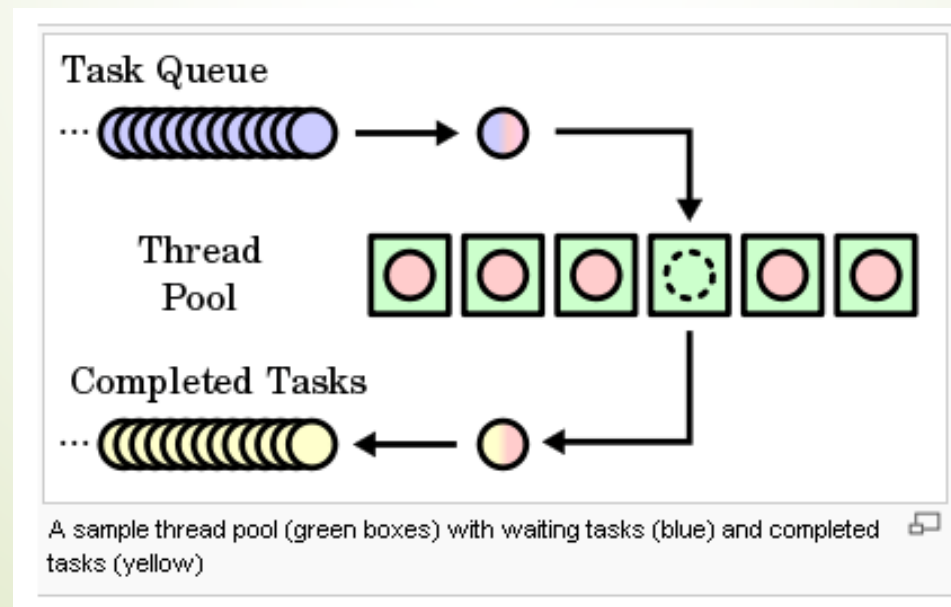
- Thread

Async programlama ile daha sıklıkla kullanılan Task yapısı thread yapısına göre üst seviyede. Task yapısını kullanarak daha gelişmiş işlemler yapabiliriz. Thread pooling yapısını otomatik olarak kullanıp birbiri ardına eklenebilecek olan işlemleri daha iyi organize etmektedir.

# Thread vs Task

## Task Nedir?

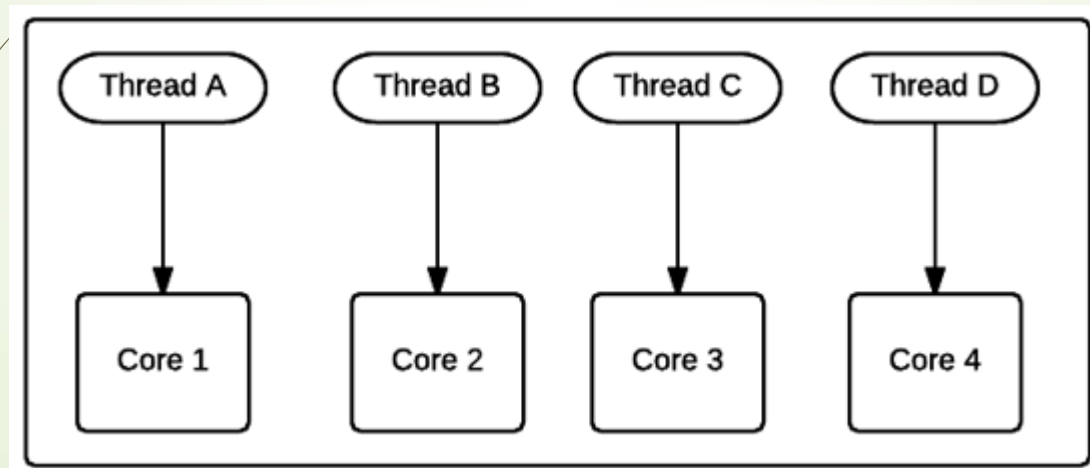
.NET framework, görevler oluşturmaya ve bunları zaman uyumsuz olarak çalıştırmaya izin veren Threading.Tasks sınıfını sağlar. Görev yapılması gereken bazı işleri temsil eden bir nesnedir. Görev, çalışmanın tamamlanıp tamamlanmadığını ve işlemin bir sonuç döndürmesi durumunda size görevin sonucunu verir.



# Thread vs Task

## Thread Nedir?

.NET Framework, System.Threading ad alanındaki konuya ilişkin sınıflara sahiptir. Bir iş parçacığı küçük bir çalıştırılabilir talimatlar kümesidir.



# Thread vs Task

## How to create a Task

```
01. static void Main(string[] args) {  
02.     Task < string > obTask = Task.Run(() => (  
03.         return "Hello"));  
04.     Console.WriteLine(obTask.result);  
05. }
```

## How to create a Thread

```
01. static void Main(string[] args) {  
02.     Thread thread = new Thread(new ThreadStart(getMyName));  
03.     thread.Start();  
04. }  
05. public void getMyName() {}
```

## Differences Between Task And Thread

Here are some differences between a task and a thread.

1. The Thread class is used for creating and manipulating a [thread](#) in Windows. A [Task](#) represents some asynchronous operation and is part of the [Task Parallel Library](#), a set of APIs for running tasks asynchronously and in parallel.
2. The task can return a result. There is no direct mechanism to return the result from a thread.
3. Task supports cancellation through the use of cancellation tokens. But Thread doesn't.
4. A task can have multiple processes happening at the same time. Threads can only have one task running at a time.
5. We can easily implement Asynchronous using 'async' and 'await' keywords.
6. A new Thread() is not dealing with Thread pool thread, whereas Task does use thread pool thread.
7. A Task is a higher level concept than Thread.