# .Net Core Asenkron- Multithread Programlama (TASK,TPL,PLINQ)

## Bölüm 1: Giriş

### 1. Giriş

### 2. Asenkron(Asynchronous) programlama nedir ? Multi Thread Programlama nedir ?

#### Asenkron(Asynchronous) programlama nedir?

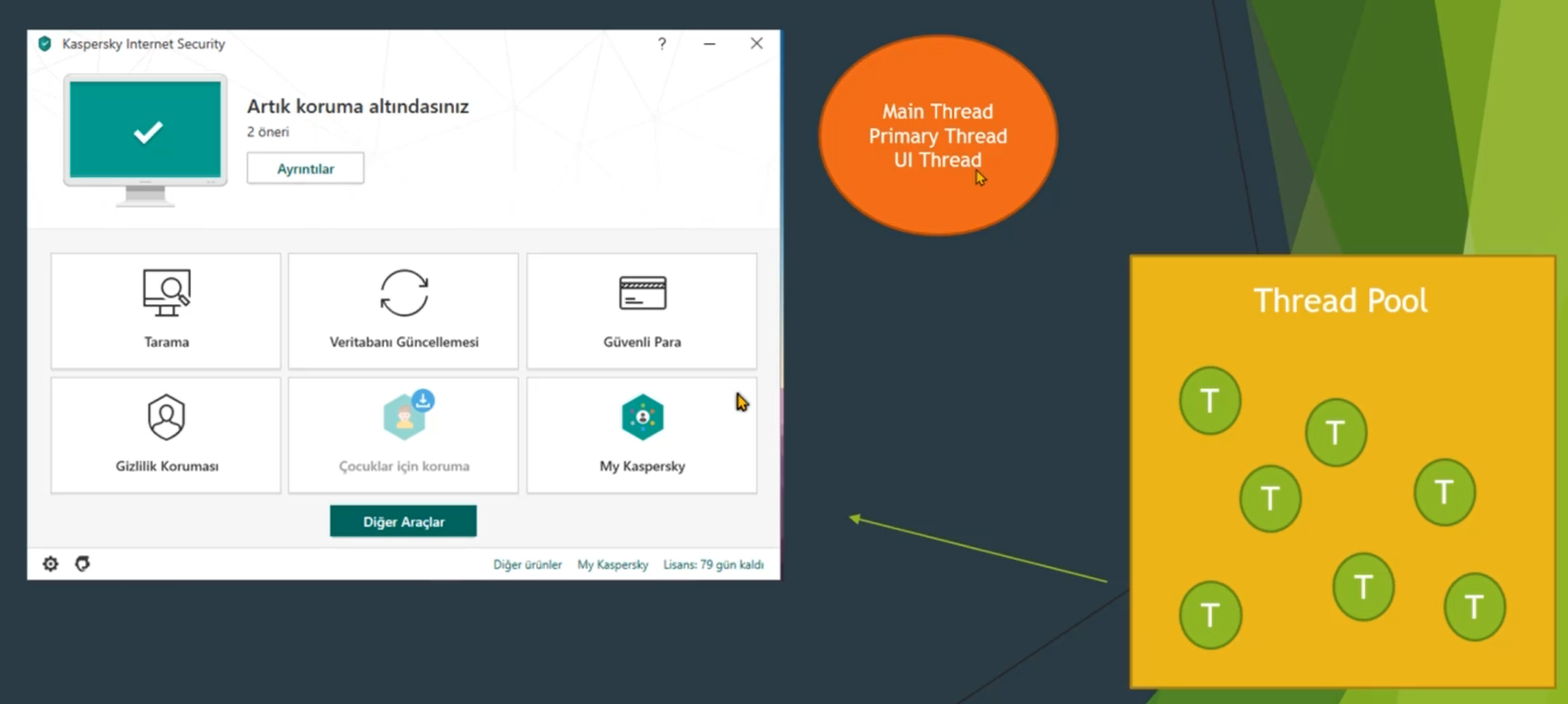
Non-blocking ya da Thread ‘in bloklanmadığı programlardır. Asenkron programlamada illa ki birden fazla Thread kullanılacağı anlamına gelmez. Birden fazla Thread ‘de kullanılabilir.

#### Multi Thread Programlama nedir?

İşlerin birden fazla Thread tarafından eş zamanlı olarak gerçekleştirilmesidir.

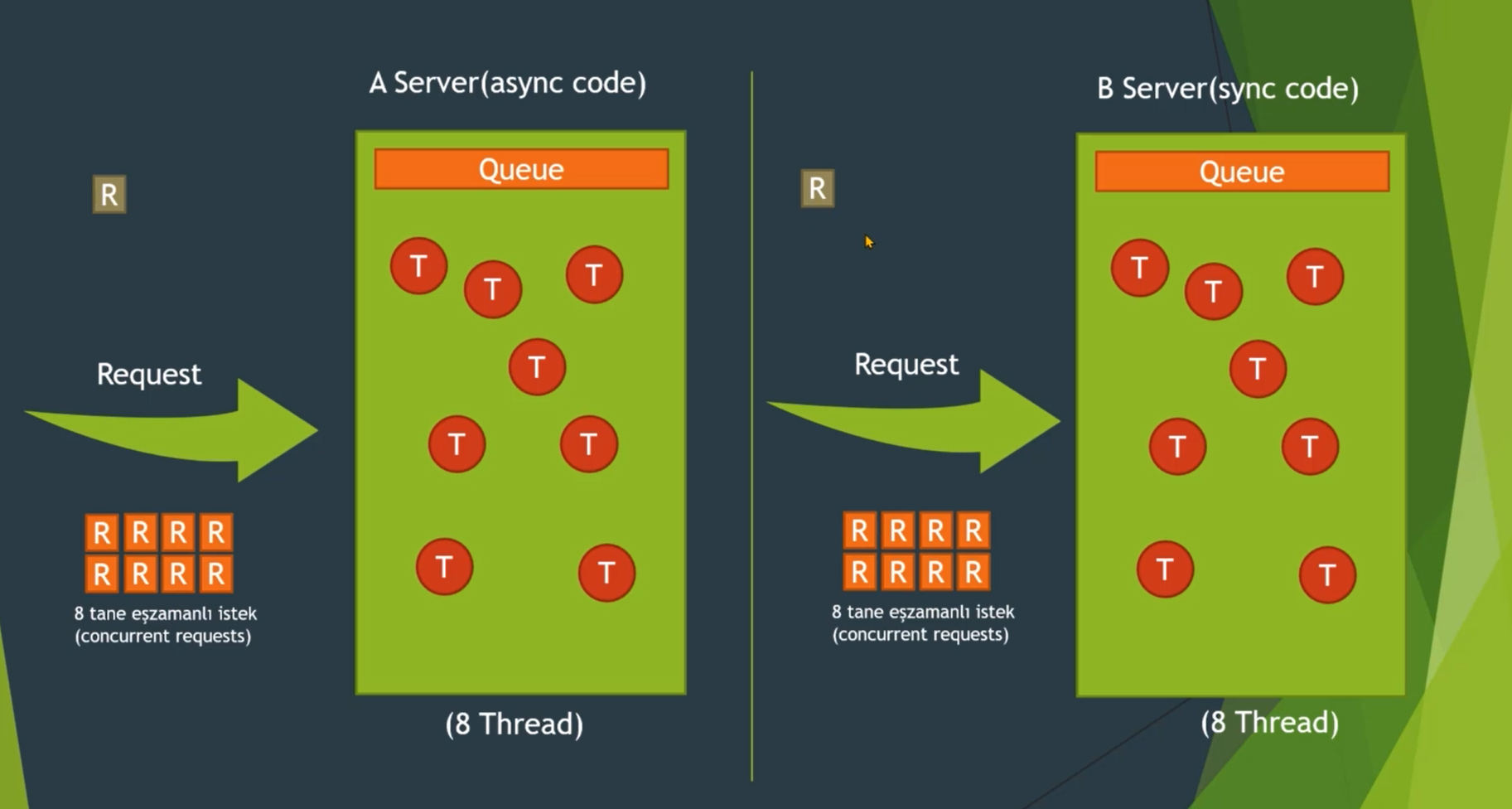
Thread: İş yapan parça.

Uygulamalar çalışmaya başladığı anda Main Thread ile ayağa kalkarlar. Windows uygulamalarında Main Thread’a UI thread denir. Web API ‘lerinde Main Thread yada Primary Thread denebilir.



Yukarıdaki Windows uygulamasında UI Thread(Main Thread) kullanıcıdan gelen istekleri karşılar.

Tarama işlemi için farklı bir Thread kullanılabilir. Thread Pool ‘dan bir ya da birden fazla Thread alınabilir.(C,D,E,F sürücülerini ayrı Thread ‘lerda tarasın.) Tarama vb. Main Thread ‘den farklı iş yapan Thread’lere Worker Thread denir.



B tarafı senkron çalışıyor. A tarafı asenkron.

B tarafına 20 istek gelirse, 8 Thread işlem yapar ve 12 tanesi kuyrukta bekler. İşlem yapan 8 Thread ‘de tüm işlem bitene kadar bloklanır.

A tarafına 20 istek gelirse, 8 Thread işlem yapar ve 12 tanesi kuyrukta bekler. Thread ‘lerden biri çalışmaya başladığında gelen isteği alır, işler ve kendinden sonraki kısma örneğin database operation kısmına iletir. Ve bir sonraki isteği alır.

Restoranda 8 garson ve 8 müşteri geldi. Garsonlar müşteriden isteklerini aldılar ve şefe gittiler.

Senkron programlamada Garsonlar Şefe gittikten sonra bekliyorlar. Şef yemeği hazır edene kadar yanında bekliyorlar. Başka bir iş yapmadan.

Garsonlar şefe siparişleri verdikten sonra masaları temizleyebilir, başka siparişleri de alabilir.

Şef 3 nolu yemek hazır dediği zaman yemeği alıp müşteriye götürebilir.

Asenkron programlamada, garsonlar şefe müşteri isteklerini ilettikten sonra başka işlerle(masa temizliği vb.) ya da yeni siparişleri de alabilirler.

## Bölüm 2: TASK

### 3. Task ( async-await)-1

Task ‘ın amacı bir söz, bir taahhüttür. Yani bir işlemi, bir görevi yerine getirmekle ilgili bir taahhüttür.

Bir metodumuz var ve geriye string dönüyor. Senkron yapıda bu metod çağrıldığı yere string döner.

Bu metod Task<string> dönüyorsa bu bir söz, taahhüt demektir. Yani çağrıldığı yerde, ben sana bir string ifade dönücem ama hemen şimdi değil(senkron ‘dan farklı olarak) eğer beklersen bir müddet sonra ben sana datanı dönücem. Task ile işlem hemen gerçekleştirilmiyor, başladığı anda bir taahhüt verilmiş oluyor, bittiği anda bir bilgi vericem denmiş oluyor. Senkron ‘da thread bekliyor.

async-await; bir metodun içerisinde asenkron bir metod çağrısı yapılacaksa bu ikili zorunlu.

Eğer bir metod, içerisinde asenkron bir çağrım yapmayacak, geriye bir Task dönecekse, bu ikilinin olmasına gerek yok.

private async Task<string> ReadFileAsync() {

}

private async ifadesi Compiler’a metod içerisinde bir async çağrı olduğunu söylüyor.

### 4. Task ( async-await)-2

private async void BtnReadFile\_Click(object sender, EventArgs e) {

string data = string.Empty;

Task<string> okuma = ReadFileAsync();

// Başka bir işlem

richTextBox2.Text = await new HttpClient().GetStringAsync("https://www.google.com.tr/");

data = await okuma; // sonucu bekle.

richTextBox1.Text = data;

}

Aşağıdaki istek async çalışıyor ve thread ‘i bloklamıyor

richTextBox2.Text = await new HttpClient().GetStringAsync("https://www.google.com.tr/");

Aşağıdaki istek sync çalışıyor ve thread ‘i blokluyor. Ama sync bir metod içerisinden async bir metod çağrılırken .Result kullanılabiliyor.

richTextBox2.Text = new HttpClient().GetStringAsync("https://www.google.com.tr/").Result;

### 5. Task ( async-await)-3

private async Task<string> ReadFileAsync() {

string data = string.Empty;

using (StreamReader s = new StreamReader("dosya.txt")) {

Task<string> myTask = s.ReadToEndAsync(); //

// Üstteki satır ile alttaki satır arasında başka işler yapılabilir.

// WebRequest atılabilir vs...

await Task.Delay(5000);

data = await myTask;

return data;

}

}

private Task<string> ReadFileAsync2() {

using (StreamReader s = new StreamReader("dosya.txt")) {

return s.ReadToEndAsync();

}

}

Yukarıdaki iki metod ‘da geriye Task<string> dönüyor. İkinci metod ‘un async yapılmasına gerek yok. Eğer bu şekilde sadece geriye Task<string> döndürüleçek ise metodun async yapılmasına gerek yok. İlk versiyonda arada başka işlemler de yapılabiliyor.

Aşağıda sync ve async Web API istekleri mevcut. Async metod ‘da ilk isteği alan thread; örneğin Thread 5, sonraki isteklerde de bloklanmaz.

public class HomeController : ControllerBase {

[HttpGet("getcontent")]

public IActionResult GetContent() {

// Başarısız Yöntemdi, Encoding.RegisterProvider ile çalışıyor.

Encoding.RegisterProvider(CodePagesEncodingProvider.Instance);

var data = new HttpClient().GetStringAsync("https://www.google.com.tr/").Result;

return Ok(data);

}

[HttpGet("getcontentasync")]

public async Task<IActionResult> GetContentAsync() {

Encoding.RegisterProvider(CodePagesEncodingProvider.Instance);

// 5. Thread aşağıdaki isteği karşılıyor olsun.

// 1. istek te Thread 5 aşağıdaki satırdan sonra bloklanmıyor.

// 2. İstek gelince Thread 5 gelen isteği karşılıyor.

var myTask = new HttpClient().GetStringAsync("https://www.google.com.tr/");

var data = await myTask;

return Ok(data);

}

}