# .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?

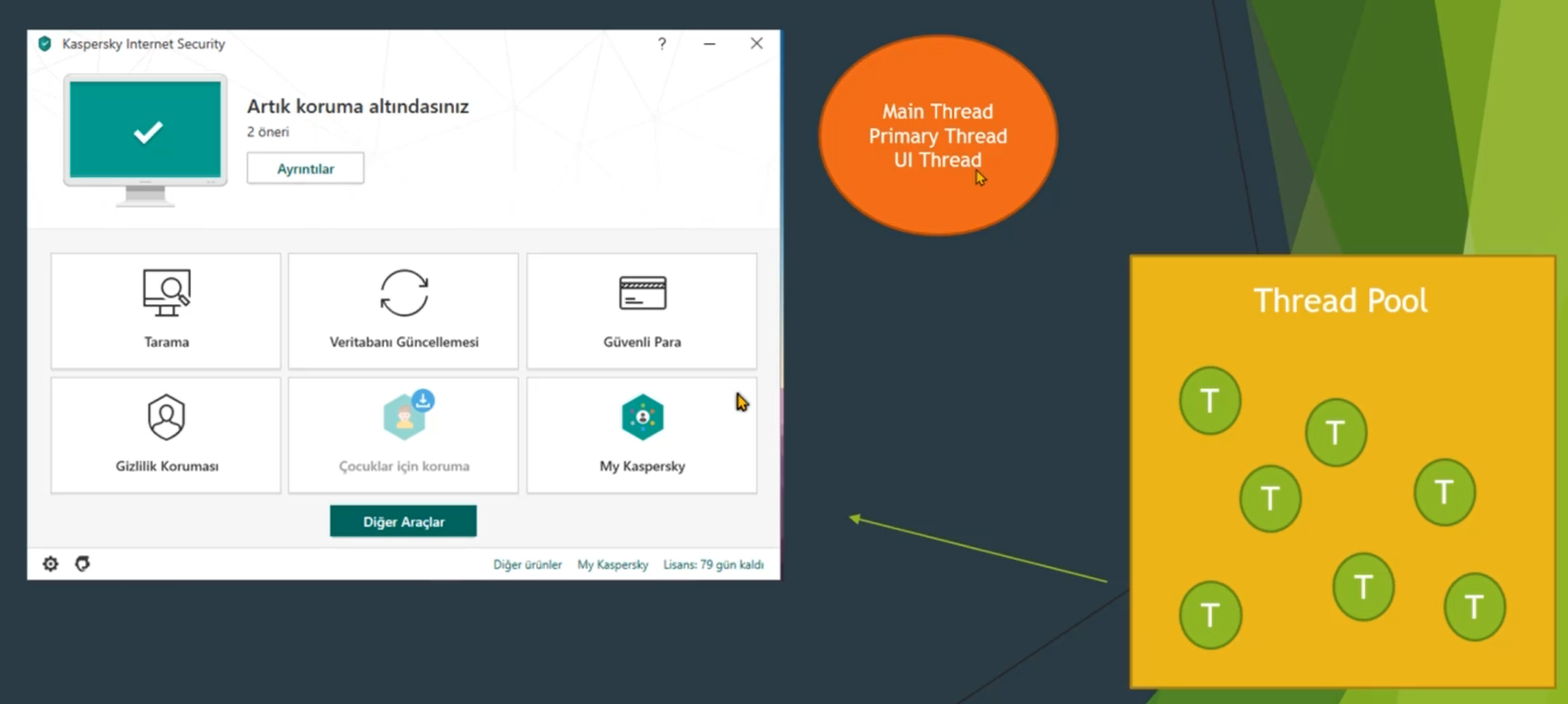
Non-blocking ya da Thread ‘in bloklanmadığı programlardır. Asenkron programlamada illa ki birden fazla Thread kullanılacağı anlamına gelmez. Birden fazla Thread ‘de kullanılabilir.

#### Multi Thread Programlama nedir?

İşlerin birden fazla Thread tarafından eş zamanlı olarak gerçekleştirilmesidir.

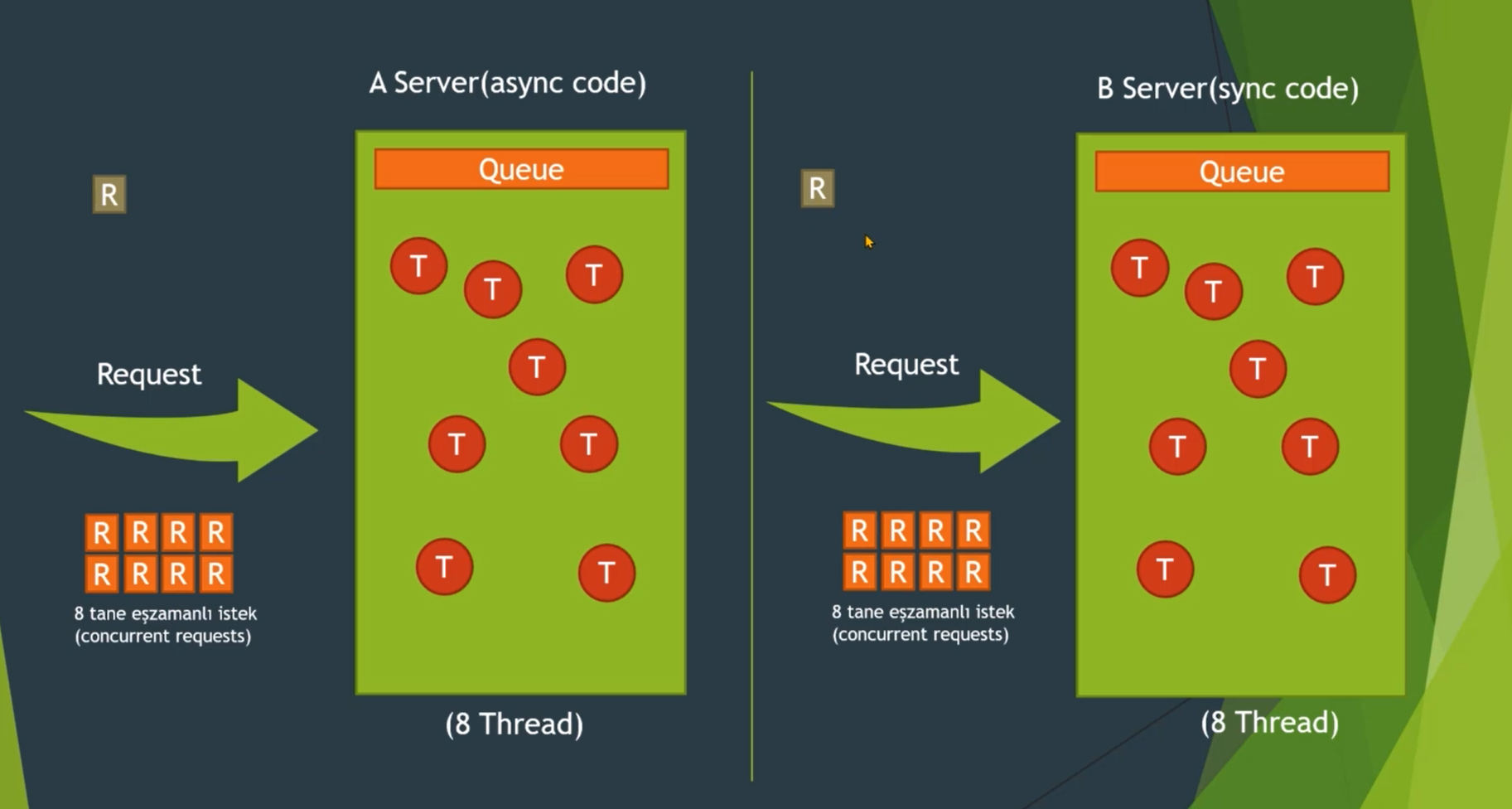
Thread: İş yapan parça.

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);

}

}

### 6. Task Method: ContinueWith

Task sınıfından sonra ContinueWith metodu ile Task işleminden sonra çalışacak kodlarınızı ekleyebilirsiniz. CallBack meganizması var.

#### 1. Yöntem

private async static Task Main(string[] args) {

Console.WriteLine("Hello World");

var mytask = new HttpClient().GetStringAsync("https://www.google.com").ContinueWith

(data => {

Console.WriteLine($"Uzunluk: {data.Result.Length}");

});

Console.WriteLine("Arada yapılacak işler");

await mytask;

}

#### 2. Yöntem

private async static Task Main(string[] args) {

Console.WriteLine("Hello World");

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

Console.WriteLine("Arada yapılacak işler");

var data = await mytask;

Console.WriteLine($"Uzunluk: {data.Length}");

}

#### 3. Yöntem

public static void calis(Task<string> data) {

// 100 satırlık bir kod

Console.WriteLine($"Uzunluk: {data.Result.Length}");

}

private async static Task Main(string[] args) {

Console.WriteLine("Hello World");

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

Console.WriteLine("Arada yapılacak işler");

await mytask;

}

Graphical user interface, text, application

Description automatically generated

### 7. Task Method: WhenAll(task1,task2,task3,task4,….)

Tüm tasklar bittikten sonra işlem yapmak için. Task array alır.

private async static Task Main(string[] args) {

Console.WriteLine($"Main Thread:{Thread.CurrentThread.ManagedThreadId}");

List<string> urlsList = new List<string>() {

"https://www.google.com",

"https://www.amazon.com/",

"https://www.microsoft.com",

"https://www.yahoo.com/",

"https://www.mynet.com/"

};

List<Task<Content>> taskList = new List<Task<Content>>();

urlsList.ToList().ForEach(url => {

taskList.Add(GetContentAsync(url));

});

var contentList = Task.WhenAll(taskList.ToArray());

Console.WriteLine("WhenAll metodundan sonra başka işler yaptım");

var data = await contentList;

data.ToList().ForEach(content => {

Console.WriteLine($"Site:{content.Site} Length:{content.Length}");

});

}

public static async Task<Content> GetContentAsync(string url) {

Content content = new Content();

var data = await new HttpClient().GetStringAsync(url);

content.Site = url;

content.Length = data.Length;

Console.WriteLine($"GetContentAsync thread: {Thread.CurrentThread.ManagedThreadId}");

return content;

}

public class Content {

public string Site { get; set; }

public int Length { get; set; }

}

Text

Description automatically generated

### 8. Task Method: WhenAny(task1,task2,task3,task4,….)

### Task array i parametre olarak alır. İlk biten Task i geriye döner.

private async static Task Main(string[] args) {

Console.WriteLine($"Main Thread:{Thread.CurrentThread.ManagedThreadId}");

List<string> urlsList = new List<string>() {

"https://www.google.com",

"https://www.amazon.com/",

"https://www.yahoo.com/",

"https://www.mynet.com/",

"https://www.microsoft.com",

};

List<Task<Content>> taskList = new List<Task<Content>>();

urlsList.ToList().ForEach(url => {

taskList.Add(GetContentAsync(url));

});

var firstData = await Task.WhenAny(taskList);

Console.WriteLine($"Site:{firstData.Result.Site} Length:{firstData.Result.Length}");

}

public static async Task<Content> GetContentAsync(string url) {

Content content = new Content();

var data = await new HttpClient().GetStringAsync(url);

content.Site = url;

content.Length = data.Length;

Console.WriteLine($"GetContentAsync thread: {Thread.CurrentThread.ManagedThreadId}");

return content;

}

public class Content {

public string Site { get; set; }

public int Length { get; set; }

### }

### 9. Task Method: WaitAll(task1,task2,task3,task4,….)

WhenAll ile aynı ama WhenAll kilitlemiyor. WaitAll blokluyor. İçerisinde milisaniye cinsinden zaman alıyor. Belirtilen sürede tamamlanırsa true, tamamlanmazsa false dönüyor. Senkron çalışıyor.

private async static Task Main(string[] args) {

Console.WriteLine($"Main Thread:{Thread.CurrentThread.ManagedThreadId}");

List<string> urlsList = new List<string>() {

"https://www.google.com",

"https://www.amazon.com/",

"https://www.yahoo.com/",

"https://www.mynet.com/",

"https://www.microsoft.com",

};

List<Task<Content>> taskList = new List<Task<Content>>();

urlsList.ToList().ForEach(url => {

taskList.Add(GetContentAsync(url));

});

Console.WriteLine("WaitAll metodundan önce");

// Task.WaitAll(taskList.ToArray());

bool result = Task.WaitAll(taskList.ToArray(), 100);

Console.WriteLine("3 saniyede sonçular geldi mi:" + result);

Console.WriteLine("WaitAll metodundan sonra");

Console.WriteLine($"Site:{taskList.First().Result.Site} Length:{taskList.First().Result.Length}");

}

public static async Task<Content> GetContentAsync(string url) {

Content content = new Content();

var data = await new HttpClient().GetStringAsync(url);

content.Site = url;

content.Length = data.Length;

Console.WriteLine($"GetContentAsync thread: {Thread.CurrentThread.ManagedThreadId}");

return content;

}

public class Content {

public string Site { get; set; }

public int Length { get; set; }

}

### 10. Task Method: WaitAny(task1,task2,task3,task4,….)

Task.WaitAny çağırıldığı yerdeki Thread’i bloklar. İlk biten task’ın Index ‘ini döner.

private async static Task Main(string[] args) {

Console.WriteLine($"Main Thread:{Thread.CurrentThread.ManagedThreadId}");

List<string> urlsList = new List<string>() {

"https://www.google.com",

"https://www.amazon.com/",

"https://www.yahoo.com/",

"https://www.mynet.com/",

"https://www.microsoft.com",

};

List<Task<Content>> taskList = new List<Task<Content>>();

urlsList.ToList().ForEach(url => {

taskList.Add(GetContentAsync(url));

});

Console.WriteLine("WaitAny metodundan önce");

var firstTaskIndex = Task.WaitAny(taskList.ToArray());

Console.WriteLine("WaitAny metodundan sonra");

Console.WriteLine($"Site:{taskList[firstTaskIndex].Result.Site} Length:{taskList[firstTaskIndex].Result.Length}");

}

### 11. Task Method: Delay()

Asenkron şekilde bir delay(gecikme) gerçekleştirir ama bu geciktirmeyi gerçekleştirirken de güncel thread’i bloklamaz.

Thread.Sleep UI thread ‘i ya da Main Thread ‘i bloklar.

private async static Task Main(string[] args) {

var stopwatch = new Stopwatch();

Console.WriteLine($"Main Thread:{Thread.CurrentThread.ManagedThreadId}");

List<string> urlsList = new List<string>() {

"https://www.google.com",

"https://www.amazon.com/",

"https://www.yahoo.com/",

"https://www.mynet.com/",

"https://www.microsoft.com",

};

List<Task<Content>> taskList = new List<Task<Content>>();

urlsList.ToList().ForEach(url => {

taskList.Add(GetContentAsync(url));

});

stopwatch.Start();

var contents = await Task.WhenAll(taskList.ToArray());

contents.ToList().ForEach(task => {

Console.WriteLine(task.Site + task.Length);

});

stopwatch.Stop();

Console.WriteLine($"Elapsed time: {stopwatch.ElapsedMilliseconds} ms");

}

public static async Task<Content> GetContentAsync(string url) {

Content content = new Content();

var data = await new HttpClient().GetStringAsync(url);

await Task.Delay(5000);

content.Site = url;

content.Length = data.Length;

Console.WriteLine($"GetContentAsync thread: {Thread.CurrentThread.ManagedThreadId}");

return content;

Text

Description automatically generated }

### 12. Task Method: Run()

Asenkron metodlar illaki Thread kullanmaz. Kullanaçağı durumlar da olabilir, kullanmayacağı durumlarda. Asıl önemli olan async metodu çağırdığımız yerde, bizim o anki thread’imizin bloklanmaması.

Task.Run() CPU bazlı işlemler.

Task.Run() ile yazılan kodlar ayrı bir thread ‘te çalışır.

public partial class Form1 : Form {

public static int Counter { get; set; } = 0;

public Form1() {

InitializeComponent();

}

//private void btnStart\_Click(object sender, EventArgs e) {

// Go(progressBar1);

// Go(progressBar2);

//}

//public void Go(ProgressBar progressBar) {

// Enumerable.Range(1, 100).ToList().ForEach(x => {

// Thread.Sleep(100);

// progressBar.Value = x;

// });

//}

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

var aTask = Go(progressBar1, textBox1);

var bTask = Go(progressBar2, textBox2);

await Task.WhenAll(aTask, bTask);

}

public async Task Go(ProgressBar progressBar, TextBox textBox) {

await Task.Run(() => {

Enumerable.Range(1, 100).ToList().ForEach(x => {

Thread.Sleep(100);

progressBar.Invoke((MethodInvoker)delegate {

progressBar.Value = x;

progressBar.Text = x.ToString();

textBox.Text = x.ToString() + " ManagedThreadId:" + Thread.CurrentThread.ManagedThreadId;

if (progressBar.Value == 10) {

MessageBox.Show($"{progressBar.Name} is {progressBar.Value}");

} else if (progressBar.Value == 50) {

MessageBox.Show($"{progressBar.Name} is {progressBar.Value}");

}

});

});

});

}

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

btnCounter.Text = Counter++.ToString();

}

}

### 13. Task Method: StartNew()

İçerisinde yazılan kodları, Task.Run() metodunda olduğu gibi ayrı bir Thread ‘te çalıştırıyor.

Task.Run() ‘a obje geçiremiyoruz ama Task.StartNew() ‘e obje geçirip, işlem bitince alabiliyoruz.

public class Status {

public int ThreadId { get; set; }

public DateTime DateTime { get; set; }

}

internal class Program {

private async static Task Main(string[] args) {

var myTask = Task.Factory.StartNew((Obj) =>

{

Console.WriteLine("myTask çalıştı");

var status = Obj as Status;

status.ThreadId = Thread.CurrentThread.ManagedThreadId;

},new Status () {

DateTime = DateTime.Now,

});

await myTask;

Status s = myTask.AsyncState as Status;

Console.WriteLine(s.DateTime);

Console.WriteLine(s.ThreadId);

}

}

### 14. Task Method: FromResult()

Cache’lenmiş datayı dönmek için kulanılır.

internal class Program {

public static string CachedData { get; set; }

private async static Task Main(string[] args) {

CachedData = await GetDataAsync();

Console.WriteLine(CachedData);

}

public static Task<string> GetDataAsync() {

//Task.Run(() => {

// return 5;

//});

//Task.Run<string>(() => {

// return 5;

//});

if (String.IsNullOrEmpty(CachedData))

return File.ReadAllTextAsync(@"C:\Users\MehmetYagci\source\repos\.Net-Core-Asenkron-Multithread-Programlama\TaskExamples\TaskFormApp\dosya.txt");

else

return Task.FromResult<string>(CachedData);

}

}

### 15. CancellationToken -1

Uzun süren async istekleri iptal etmek isteyebiliriz. 30 dak.süreçek bir işlemi iptal etmek isteyebiliriz.

Async işlem başlatılırkenbir Cancelation Token veriliyor. İşlem iptal edilmek istenildiğinde bu Token kullanılıyor.

public partial class Form1 : Form {

CancellationTokenSource ct = new CancellationTokenSource();

public Form1() {

InitializeComponent();

}

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

Task<HttpResponseMessage> myTask;

try {

myTask = new HttpClient()

.GetAsync(@"https://localhost:7178/api/Home/getcontent",

ct.Token

);

await myTask;

var content = await myTask.Result.Content.ReadAsStringAsync();

richTextBox1.Text = content;

}

catch (TaskCanceledException tcex) {

MessageBox.Show(tcex.Message);

}

catch (Exception ex) {

throw;

}

// Console.WriteLine($"myTask.IsCanceled:{myTask.IsCanceled.ToString()}" );

}

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

ct.Cancel();

}

}

### 16. CancellationToken -2

[Route("api/[controller]")]

[ApiController]

public class HomeController : ControllerBase {

private readonly ILogger<HomeController> \_logger;

public HomeController(ILogger<HomeController> logger) {

\_logger = logger;

}

[HttpGet("getcontent")]

public async Task<IActionResult> GetContent(CancellationToken ct) {

try {

\_logger.LogInformation("İstek başladı");

await Task.Delay(5000, ct); // 10 sn. bekle

// Manuel Throw fırlatma

ct.ThrowIfCancellationRequested();

// Başarısız Yöntemdi, Encoding.RegisterProvider ile çalışıyor.

Encoding.RegisterProvider(CodePagesEncodingProvider.Instance);

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

\_logger.LogInformation("İstek bitti");

return Ok(data);

}

catch (Exception ex) {

\_logger.LogInformation("İstek iptal edildi.Exception details are:" + ex.ToString());

return BadRequest();

}

// Başarılı Yöntem 1.

//byte[] responseBytes = new HttpClient().GetByteArrayAsync("https://www.google.com.tr/").Result;

//string responseString = Encoding.UTF8.GetString(responseBytes);

//return Ok(responseString);

// Başarılı Yöntem 2.

//var client = new HttpClient();

//client.DefaultRequestHeaders.Accept.Clear();

//client.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("text/html"));

//client.DefaultRequestHeaders.Add("User-Agent", "Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.2; WOW64; Trident/6.0)");

//client.DefaultRequestHeaders.TryAddWithoutValidation("Content-Type", "text/html; charset=utf-8");

//string responseString = client.GetStringAsync("https://www.google.com.tr/").Result;

//return Ok(responseString);

}

[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);

}

[HttpGet("getcontentasync2")]

public async Task<IActionResult> GetContentAsync2(CancellationToken ct) {

try {

\_logger.LogInformation("İstek başladı");

Enumerable.Range(1, 10).ToList().ForEach(x => {

Thread.Sleep(1000);

ct.ThrowIfCancellationRequested();

});

\_logger.LogInformation("İstek bitti");

return Ok("işler bitti");

}

catch (Exception ex) {

\_logger.LogInformation("İstek iptal edildi.Exception details are:" + ex.ToString());

return BadRequest();

}

}

}

### 17. Task Instance Result Property

Senkron metod içerisinden asenkron metodun değerini almak için kullanılır.

O anki Thread ‘i bloklar.

internal class Program {

private async static Task Main(string[] args) {

Console.WriteLine($"Main1->ManagedThreadId:{Thread.CurrentThread.ManagedThreadId.ToString()}");

Console.WriteLine($"GetData():{GetData().Substring(0,10)}");

Console.WriteLine($"Main2->ManagedThreadId:{Thread.CurrentThread.ManagedThreadId.ToString()}");

var asyncResult = await GetDataAsync();

Console.WriteLine($"Main3->ManagedThreadId:{Thread.CurrentThread.ManagedThreadId.ToString()}");

Console.WriteLine($"asyncResult:{asyncResult.Substring(0, 10)}");

Console.WriteLine($"Main4->ManagedThreadId:{Thread.CurrentThread.ManagedThreadId.ToString()}");

}

public static string GetData() {

Console.WriteLine($"GetData1->ManagedThreadId:{Thread.CurrentThread.ManagedThreadId.ToString()}");

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

Console.WriteLine($"GetData2->ManagedThreadId:{Thread.CurrentThread.ManagedThreadId.ToString()}");

return task.Result; // Thread 'i bloklar. Winform' da Pencereyi taşımayı, başka bir button 'a tıklamayı vs. engeller.

}

public static Task<string> GetDataAsync() {

Console.WriteLine($"GetDataAsync1->ManagedThreadId:{Thread.CurrentThread.ManagedThreadId.ToString()}");

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

Console.WriteLine($"GetDataAsync2->ManagedThreadId:{Thread.CurrentThread.ManagedThreadId.ToString()}");

return task;

}

}

### 18. Task Instance Result Property-2

internal class Program {

private async static Task Main(string[] args) {

/// Yöntem 1

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

await task; // Thread bloklanmadan işlem yapıldı.

var result = task.Result; // Burada data elimizde artık. İstedğimiz gibi kullanabiliriz

Console.WriteLine($"result:{result.Substring(0, 100)}");

/// Yöntem 1

var task2 = new HttpClient().GetStringAsync("https://www.google.com/")

.ContinueWith((data) => {

var result2 = data.Result;

Console.WriteLine($"result2:{result2.Substring(0, 100)}");

});

await task2;

}

private static string GetData() {

return string.Empty;

}

}

### 19. Task Instance Properties

### 

### 20. ValueTask nedir ?

### 21. Task Akış durumu