4. HAFTA NOTU (C# ile Thread Uygulaması)

Bu not dersin 4. haftasında yapacağınız uygulamaya hazırlanmanızı sağlayacak olup temel kavramları anlatan bilgileri içermektedir.

DERSİN ÖĞRETİM ÜYESİ

Prof. Dr. Bahriye AKAY

DENEYİ YAPTIRAN ÖĞRETİM ELEMANI

Arş. Gör. Mehmet Uğur TÜRKDAMAR

2026
KAYSERİ

1-) Thread (İş Parçacığı) Nedir?

Thread, bir program (process) içerisinde bağımsız olarak çalışabilen en küçük yürütme birimidir.
Aynı process içinde birden fazla thread aynı anda (eşzamanlı) çalışabilir.

Process: Çalışan uygulamanın tamamı
Thread: O uygulama içindeki çalışma yolları

Bir process en az bir thread (main thread) içerir.

Özellik	Process	Threa
Bellek Alanı	Ayrı	Ortal
Oluşturma Maliyeti	Yüksek	Düşü
Haberleşme	Zor (IPC)	Kola
Çökme Etkisi	Sadece kendisi	Tüm pro

Thread'ler aynı bellek alanını paylaştığı için hızlıdır, ancak senkronizasyon gerektirir.

2-) Neden Thread Kullanılır?

- Paralel işlem yapmak
- Uygulamanın donmasını engellemek
- Arka planda uzun süren işlemleri çalıştırmak
- CPU çekirdeklerini verimli kullanmak

Örnek:

1. Dosya indirilirken arayüzün donmaması
2. Aynı anda birden fazla kullanıcı isteğinin işlenmesi

3-) Thread Türleri

İki tür Thread bulunmaktadır:

3.1. Foreground Thread

Program kapanana kadar çalışır

Main thread varsayılan olarak foreground'dur

3.2. Background Thread

Ana thread bittiğinde otomatik kapanır

Loglama, izleme işlemleri için kullanılır

4-) Thread Yaşam Döngüsü

Unstarted – Oluşturuldu ama başlatılmadı

Running – Çalışıyor

Wait / Sleep – Bekliyor

Stopped – Bitti

5-) Thread.Sleep() Ne Yapar?

Thread.Sleep(500);

- Thread'i 500 milisaniye askıya alır
- CPU kullanımı durur
- Diğer thread'lere çalışma şansı verilir

Sleep bloklayıcıdır → UI thread'de kullanılmamalıdır

6-)Thread Güvenliği (Thread Safety)

Birden fazla thread aynı veriye erişiyorsa sorun oluşabilir:

Race Condition

Sonucun hangi thread tarafından yazıldığı belirsizdir

Çözüm: lock

```
lock (kilit)
{
   // kritik bölge
}
```

Aynı anda sadece bir thread çalışabilir

7-)Deadlock Nedir?

İki thread'in birbirini beklemesi sonucu oluşan kilitlenme durumudur.

Sebep:

- Yanlış lock sıralaması
- İç içe kilitler

1-1) Öğrenme Hedefleri

- C# programlama dilinde thread (iş parçacığı) kavramını
- Thread'in temel çalışma mantığını
- Thread oluşturmayı
- Basit senkronizasyon problemlerini öğrenme

1-2) Uygulama Hakkında

1- Sizden C#'ta System.Threading kütüphanesi kullanarak thread oluşturması istenmektedir. Bunun için 5'e kadar sayan sayaç oluşturup, sayacın her bir adımında Thread oluşturup konsol ekranına Thread'in sırasını yazdırıp her adımda 500 ms programı uyutacak kodu yazınız.

```
using System;
using System.Threading;
class Program
{
        static void Main()
        {
                for (int i = 1; i <= 5; i++)
                {
                        int threadNumber = i;
                        Thread thread = new
Thread(() =>
{
        Console.WriteLine($"Thread
        {threadNumber} çalışıyor");
        Thread.Sleep(500); // 500 ms uyut

});
thread.Start();
  }
  Console.ReadLine(); // Program hemen kapanmasın
 }
}
```

1-3) Çoklu Thread Örneği

İki farkı thread oluşturup 300 ms arayla çalıştırınız, Thread'lerin çalışma sıralarının değiştiğini gözlemleyiniz.

```
using System;
using System.Threading;
namespace ThreadOrnek
{
   class Program
   {
      static void Main(string[] args)
      {
         Thread thread1 = new Thread(() =>
         {
            for (int i = 1; i <= 5; i++)
            {
               Console.WriteLine($"Thread 1 çalışıyor → Adım
        {i}");
               Thread.Sleep(300);
            }
         });

Thread thread2 = new Thread(() =>
         {
            for (int i = 1; i <= 5; i++)
            {
               Console.WriteLine($"Thread 2 çalışıyor → Adım
        {i}");
               Thread.Sleep(300);
            }
         });

         // Thread'leri 300 ms arayla başlat
         thread1.Start();
         Thread.Sleep(300);
         thread2.Start();

         // Thread'lerin bitmesini bekle
         thread1.Join();
         thread2.Join();

         Console.WriteLine("Tüm thread'ler tamamlandı.");
         Console.ReadLine();
      }
   }
}
```

1-4) Thread Senkronizasyonu (Giriş)

1-4-1) Race Condition

Birden fazla thread'in aynı veriye aynı anda erişmesi sonucu oluşan hatadır.

Örnek Problem:

- Ortak bir sayaç değişkeninin yanlış artması

1-4-2) Lock Kullanımı

```
static object kilit = new object();
static int sayac = 0;

static void Artir()
{
    for (int i = 0; i < 1000; i++)
    {
        lock (kilit)
        {
            sayac++;
        }
    }
}
```

lock: Aynı anda sadece bir thread'in kod bloğuna girmesini sağlar.

```
using System;
using System.Threading;

class Program
{
   static int counter = 0;
   static object lockObj = new object();
   static void Artir()
   {
      for (int i = 0; i < 1000; i++)
      {
         lock (lockObj)
         {
            counter++;
         }
      }
   }
   static void Main()
   {
      Thread t1 = new Thread(Artir);
      Thread t2 = new Thread(Artir);
      Thread t3 = new Thread(Artir);

      t1.Start();
      t2.Start();
      t3.Start();
      t1.Join();
      t2.Join();
      t3.Join();

      Console.WriteLine("Final Counter: " + counter);
      Console.ReadLine(); }}
```

Ders Katkıları

Bu uygulama kapsamında kullanılan yapılar, öğrencinin aşağıdaki teknik kazanımları edinmesini sağlar:

Paralel Programlama Mantığını Anlama

- Program içerisinde aynı anda birden fazla işlemin nasıl çalıştırılabileceğini kavrama
- İş parçacıklarının (thread) program akışına nasıl dahil edildiğini anlama
- Paralel çalışan işlemlerin program performansına etkisini gözlemleme
- Çoklu işlem mantığını algoritmik olarak planlayabilme

Thread Oluşturma ve Yönetme Yetkinliği

- C# programlama dilinde System.Threading kütüphanesini kullanma
- Thread oluşturma, başlatma ve yönetme işlemlerini öğrenme
- Birden fazla thread'in eş zamanlı çalışmasını gözlemleme
- Thread'lerin çalışma sırasının değişebileceğini deneyimleme

Program Akış Kontrolü ve Zaman Yönetimi

- Thread.Sleep() metodunun program akışına etkisini anlama
- Thread'lerin belirli sürelerle çalışmasını kontrol edebilme
- Zaman gecikmesi kullanarak thread davranışlarını gözlemleme
- Programın eş zamanlı işlem yaparken nasıl çalıştığını analiz etme

Thread Yaşam Döngüsünü Anlama

- Thread'in oluşturulması ve başlatılması süreçlerini öğrenme
- Running, Waiting ve Stopped durumlarını kavrama
- Thread'lerin çalışma sürecini ve tamamlanmasını takip edebilme

5. Senkronizasyon ve Thread Güvenliği

- Birden fazla thread'in aynı veriye erişmesi durumunda oluşabilecek sorunları fark etme
- Race Condition kavramını öğrenme
- lock mekanizması ile kritik bölgeleri koruma
- Thread güvenliği sağlayarak doğru sonuç elde etmeyi öğrenme

6. Çoklu Thread Davranışlarını Gözlemleme

- Thread'lerin farklı zamanlarda çalışabileceğini deneyimleme
- Join() metodu ile thread'lerin tamamlanmasını bekleme
- Paralel çalışan programların davranışını analiz etme
- Çoklu thread yapısının gerçek sistemlerde nasıl kullanılabileceğini kavrama