YZM 3102
İşletim Sistemleri

Yrd. Doç. Dr. Deniz KILINÇ
Celal Bayar Üniversitesi
Hasan Ferdi Turgutlu Teknoloji Fakültesi
Yazılım Mühendisliği

BÖLÜM - 5

Bu bölümde,

- Thread (İş Parçacığı)
- Thread'lerin Faydaları
- Multi-core Programlama
- Amdahl Yasası
- Paralel Çalışma Türleri
- Multi-threading Modelleri
- Thread Kütüphaneleri
- Pthreads
- .NET Threads
- Dolaylı thread

konularına değinilecektir.

Thread (İş Parçacığı)

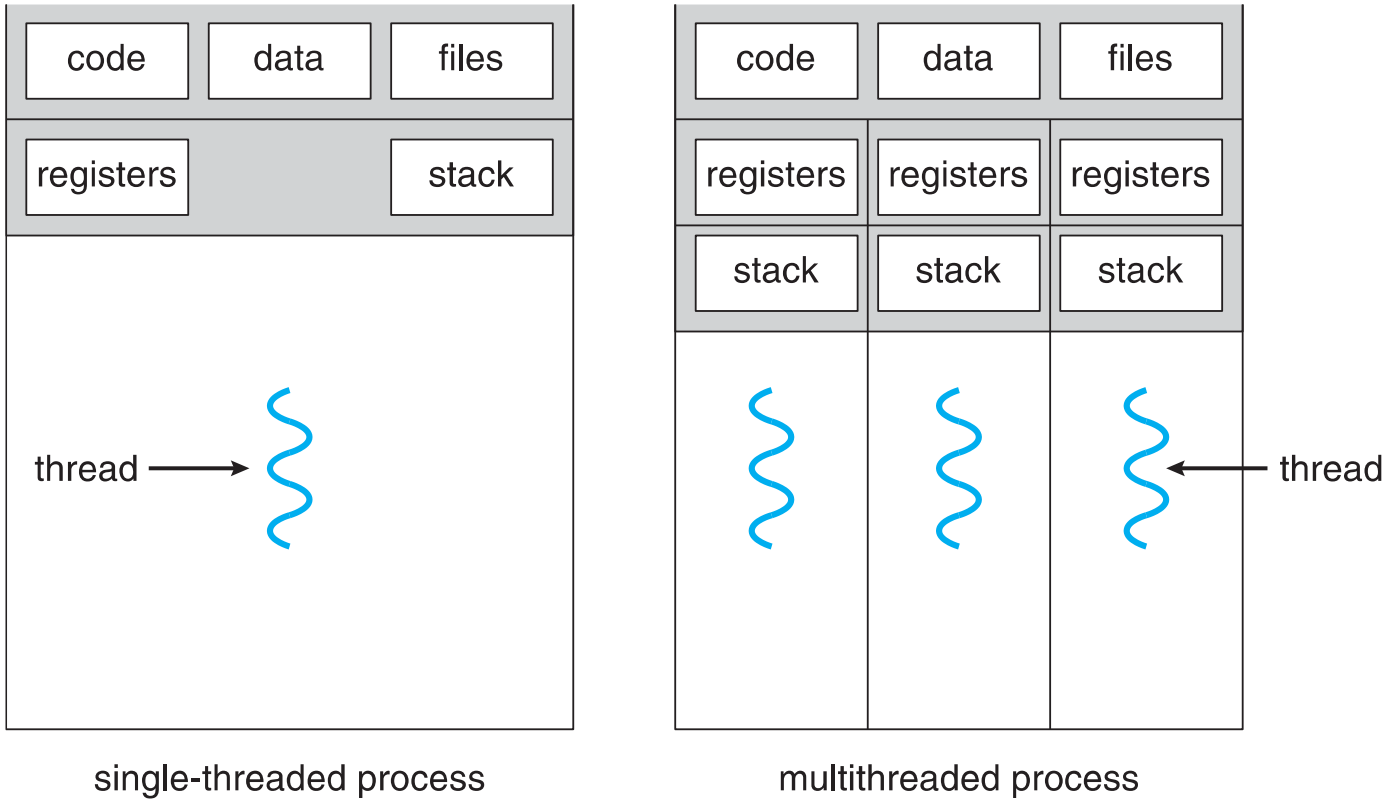
- Bir thread,
  - Thread Id
  - Program sayacı,
  - Bir grup register ve
  - Bir stack yapısını içerir.
- Thread'ler, program kodunu, data kısmını, dosyalar gibi işletim sistemi kaynaklarını **ortak kullanır**.
- *Klasik prosesler* **tek thread'e** sahiptirler.
- Eğer bir proses, **birden fazla thread'e sahipse** birden fazla görevi **eşzamanlı** yapabilir.
- Günümüzdeki modern bilgisayarlarda çalışan yazılım uygulamalarının **çoğu multithread** çalışırlar.

Thread (İş Parçacığı) (devam…)

- Uygulamalar, çok sayıda thread'e sahip **tek proses** şeklinde geliştirilirler.
- Bir **web tarayıcı**,
  - bir thread ile **veri aktarımı** yapabilir,
  - başka thread ile **verileri ekranda görüntüleyebilir**.
- Bir **kelime işleme** (***Microsoft Word***) uygulaması,
  - bir thread ile **klavyeden giriş alabilir**,
  - bir thread ile **spell check yapabilir** ve
  - başka bir thread ile **ekran görüntüsünü** düzenleyebilir.
- Çoğu işletim sistemi kernel'ı multhreaded yapıdadır.

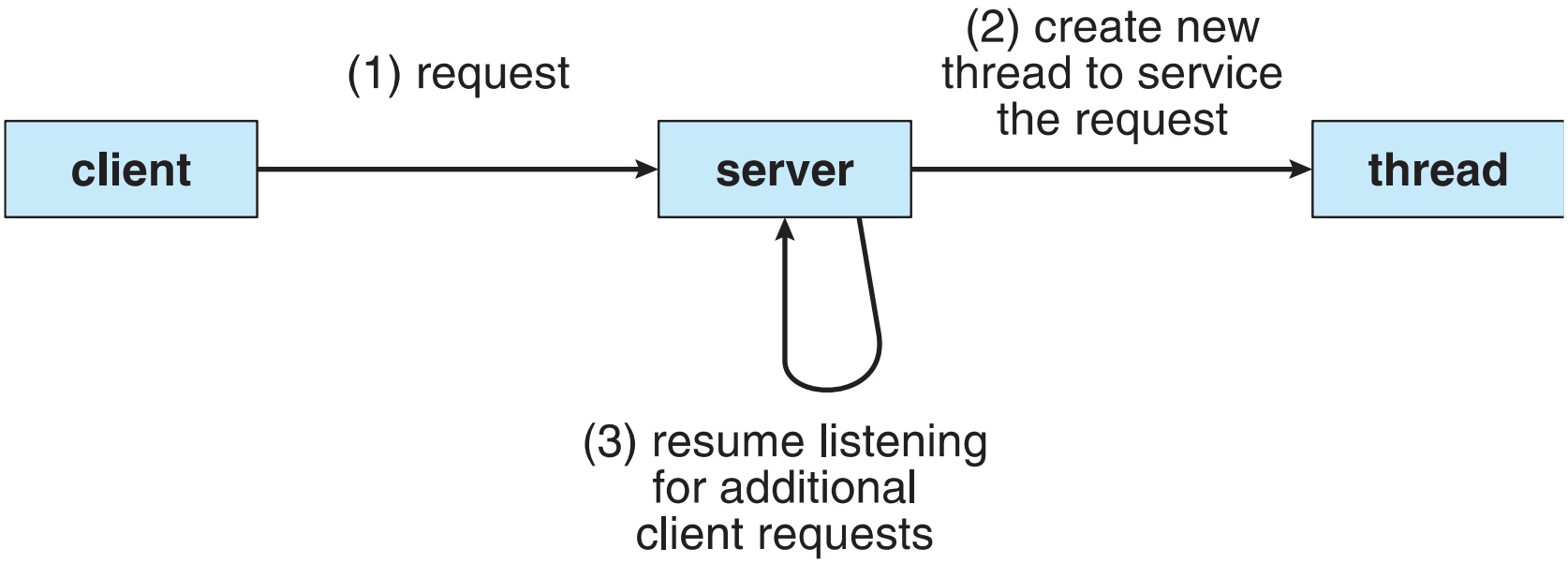
Thread (İş Parçacığı) (devam…)

- Her thread, **paylaşmadan kullandığı** kendisine ait bileşenlere sahiptir. **Single ve multi-threaded proses:**



Multi-thread Sunucu Mimarisi

- Bir Web sunucu prosesi **multithreaded çalışırsa**, her gelen istek için **ayrı bir thread oluşturulur** ve proses portu dinlemeye devam eder.
- Her istek için ayrı bir proses oluşturmak **hem zaman alır** hem de **maliyetlidir**.



Thread'lerin Faydaları

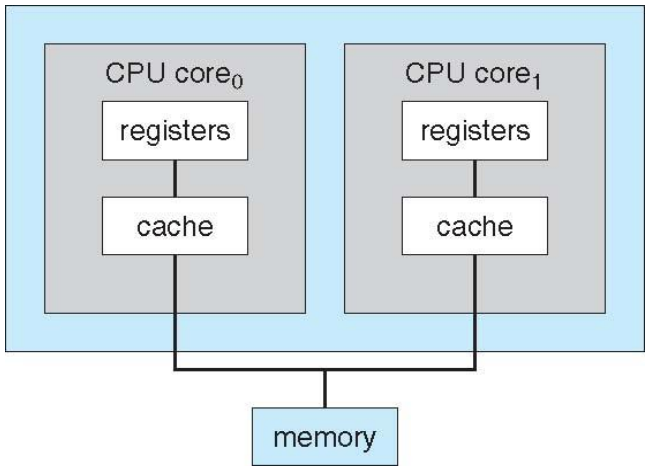
- Thread kullanımının faydalarını 4 ana başlıkta toplayabiliriz:
  1. **Responsiveness:** Kullanıcı etkileşimli uygulamalarda, bir kısım bloklanmış, kilitlenmiş veya uzun süren işlem **yürütülse bile** kullanıcı ile etkileşim yapan kısmı çalışmasını sürdürür. **Sistemin cevap verebilirlik özelliği artmış olur.**
  2. **Resource sharing:** Prosesler kaynaklarını shared memory veya message passing teknikleri aracılığıyla paylaşabilirler. **Thread'ler ait oldukları prosesin sahip olduğu hafıza alanını ve diğer kaynakları paylaşabilirler.**

Thread'lerin Faydaları (devam…)

3. **Economy:** Bir process oluştururken hafıza ve kaynak tahsis edilmesi maliyeti yüksek bir iştir. **Thread'ler ait oldukları prosesin kaynaklarını paylaştıklarından dolayı context switch daha düşük maliyetle yapılır.** (Solaris işletim sisteminde, thread oluşturma 30 kat daha hızlıdır ve thread'lerde context switch 5 kat daha hızlıdır.)
4. **Scalibility: Çok işlemcili mimarilerde thread'ler farklı core'lar üzerinde eşzamanlı çalışabilir.** Ancak, tek thread yapısına sahip process sadece bir işlemci üzerinde çalışabilir.

Multi-core Programlama

- Bilgisayar mimarisindeki en önemli gelişmelerden birisi, **çok işlemcili** *sistemlerin geliştirilmesidir*.
- Son zamanlarda, **tek chip içerisine birden fazla core yerleştirilmektedir** ve bu tür sistemler **multi-core** veya **multi-processor** olarak adlandırılır.
- Her bir core **işletim sistemi için** ayrı bir işlemci olarak görünür.

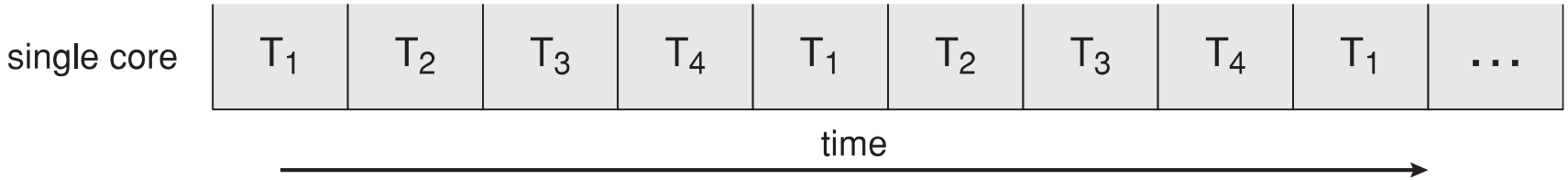


Multi-core Programlama (devam…)

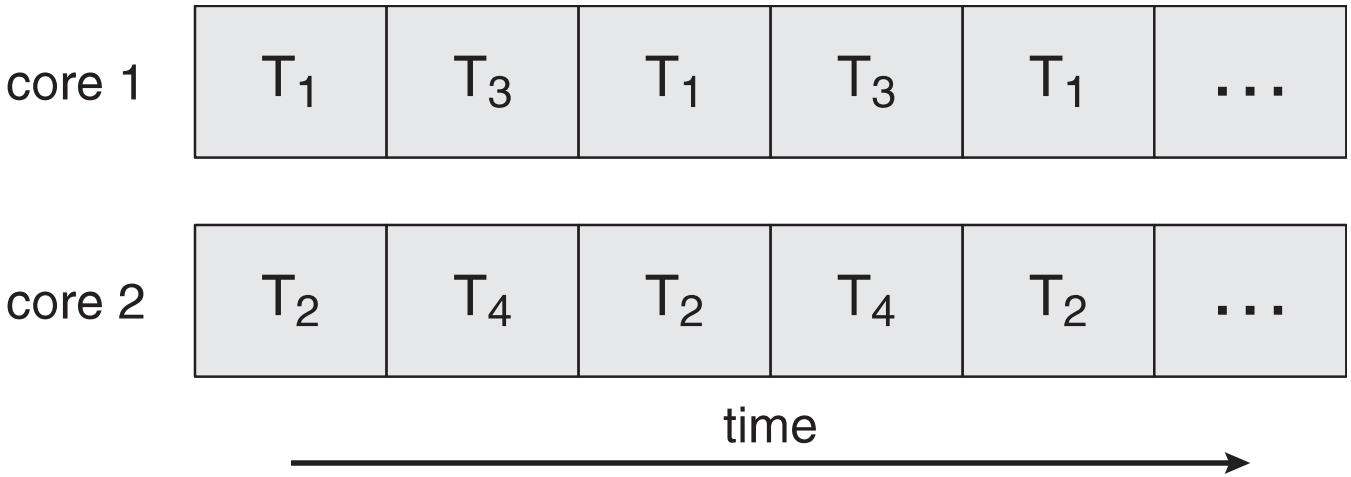
- Bir core üzerinde çalışan **4 thread'e sahip** bir uygulama için **eşzamanlı çalışma**, **thread'lerin belirli aralıklarla çalıştırılmasını ifade eder.**
- Çok core'a sahip sistemlerde **eşzamanlı çalışma**, **her core'a bir thread atanarak thread'lerin paralel çalışmasını ifade eder.**
- **Parallelism,** birden fazla görevin **eşzamanlı** yapılmasını ifade eder.
- **Concurrency,** birden fazla görev arasında **kısa aralıklarla geçiş yaparak** *birlikte ilerletilmesini sağlar*.

Multi-core Programlama (devam…)

Concurrent execution on single-core system:



Parallelism on a multi-core system:



Multi-core Programlama (devam…)

- **CPU planlayıcıları (schedulers)**
  - Prosesler arasında ***sürekli switch ederek*** progress'i (devamlılığı) sağlarlar ve
  - ***Parallelism yapılıyormuş*** **izlenimi verirler**.
- Aslında bu prosesler
  - **Concurrent** çalışmaktadırlar.
  - **Paralel** çalışmamaktadırlar.

Amdahl Yasası

- Amdahl kuralı: **core sayısına göre bir sistemdeki performans artışı** aşağıdaki gibi ifade edilir:
- *S:* uygulamada **seri çalışması zorunlu** olan kısmın oranını (**paralel olamayan**), *N:* ise **core sayısını** ifade eder.

$$speedup \leq \frac{1}{S + \frac{(1-S)}{N}}$$

- Bir uygulamada, %75 paralel ve %25 seri çalışıyorsa (S=0,25),
  - 2 core'a (N=2) sahip sistemde bu uygulamayı çalıştırınca **1,6 kat hız artar**.
  - Core sayısı 4 olduğunda, **2.28 kat hız artışı** sağlanır.
- **Core sayısı** sonsuza giderken hız artışı **(1/S) 'e doğru gider**

Multi-core Programlama Zorlukları

- Programlama zorluğu 2 taraftan değerlendirilebilir.

1. **İşletim Sistemi Tasarımı Açısından**
   - İşletim sistemi tasarımcıları multi-core sistemlerin performansını artırmak için **scheduling algoritmaları** yazmak zorundadır.
2. **Uygulama Yazılımı Açısından**
   - Uygulama geliştiricilerinin **mevcut programları değiştirmeleri** ve yeni programları **multi-hreaded şekilde tasarlamaları** gerekmektedir.

Multi-core Programlama Zorlukları (devam…)

- Multicore programlamada 5 önemli zorluk vardır:
  - **Identifying tasks:** Uygulamalarda eşzamanlı çalışabilecek ayrı alanların bulunması gereklidir. Bu alanlar farklı core'lar üzerinde paralel çalışacaktır.
  - **Balance:** Programcıların görevleri ayrıştırırken iş yükünü eşit dağıtmaları gereklidir.
  - **Data splitting:** Verilerin farklı core'lar üzerinde çalışan görevler tarafından erişilecek ve işlem yapılacak şekilde ayrıştırılması gereklidir.
  - **Data dependency:** Bir görevin erişeceği verinin diğer görevlerle bağımlılığının incelenmesi gereklidir.
  - **Testing and debugging:** Multi-hreaded çalışan programların test ve debug işlemi daha zordur.

Paralel Çalışma Türleri

- Temel olarak
  - **Data parallelism** ve
  - **Task parallelism** olarak

iki farklı paralel çalışma türü vardır.

1. **Data Parallelism:** Aynı veri kümesine ait parçaların core'lara dağıtılması ve *aynı tür işlemin eşzamanlı yürütülmesine* odaklanır.
   - **Örneğin:** N elemanlı bir dizi toplamı için **2 core kullanılacaksa**, [0]..[(N/2)-1] eleman 1.coreda [N/2]..[N-1] eleman 2.coreda toplanır.

Paralel Çalışma Türleri (devam…)

2. **Task Parallelism:** Core'lara görevlerin (thread'ler) dağıtılmasına odaklanır.
   - **Her thread ayrı bir işlemi gerçekleştirir.** Farklı thread'ler aynı veride veya farklı veride çalışabilir.
   - **Örneğin:** Aynı dizi elemanları üzerinde farklı istatistiksel hesaplamalar yapan thread'ler aynı veriyi kullanır farklı core'larda çalışır.

- Çoğu pratik uygulamada, tasarlanan uygulamalar *hem data hem de task paralel çalışma türlerini* **hibrit** olarak kullanır.

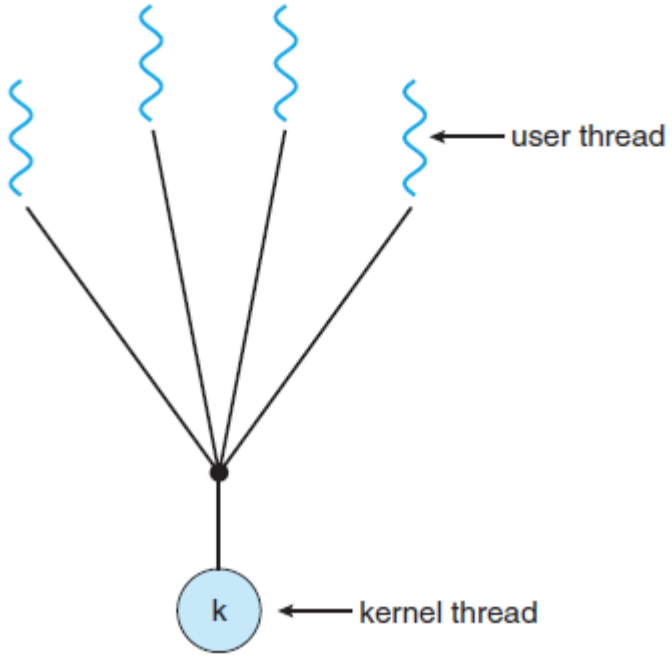
Multi-threading Modelleri

- Thread desteği, kullanıcı seviyesinde **user thread'ler** için veya kernel seviyesinde **kernel thread'ler** için sağlanabilir.
- User thread'leri kullanıcı uygulamaları tarafından, kernel thread'leri ise işletim sistemi tarafından gerçekleştirilir.
- Windows, Linux, Unix, Mac OS X ve Solaris gibi işletim sistemleri **kernel thread'leri destekler**.
- Kernel thread'leri ile user thread'leri **arasında** *aşağıdaki* ***ilişkilendirme modellerinden birisinin oluşturulması zorundadır***.
  - Many-to-one model
  - One-to-one model
  - Many-to-many model

Many-to-one Model

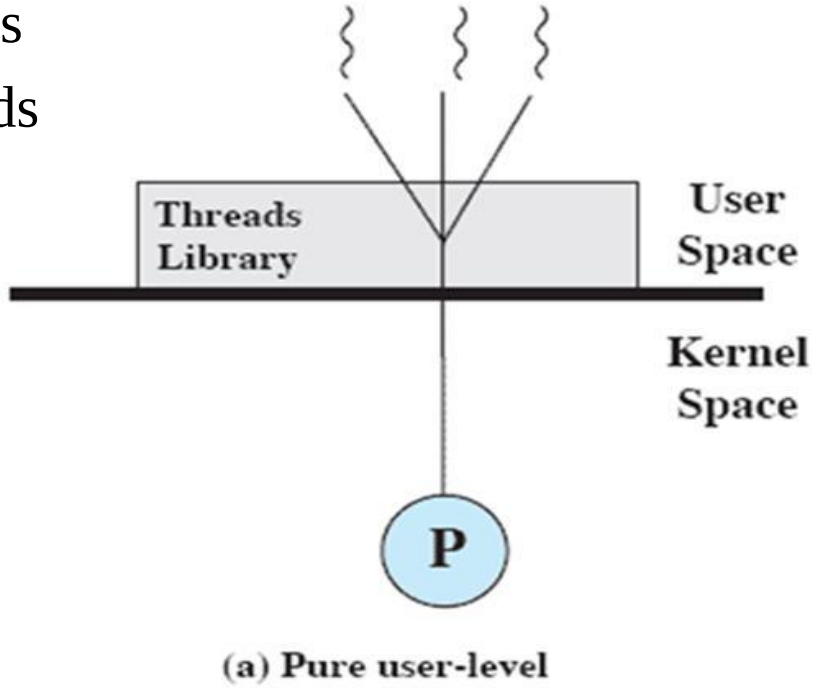
- Many-to-one modelinde, **çok sayıda kullanıcı thread'i bir tane kernel thread'i ile eşleştirilir.**
- Eğer **bir thread sistem çağrısını bloklarsa** tüm process bloklanmış olur.

- Aynı anda sadece bir tane kullanıcı thead'i kernel thread'e erişebilir.
- Sadece bir kernel thread'i kullanıldığı için **multicore sistemlerde birden fazla thread için eşzamanlı çalışma yapılamaz.**



Many-to-one Model (devam…)

- **User-level thread** modeli olarak da geçer.
- Kernel, **thread'lerin varlığından haberdar değildir**.
- Az sayıda sistem bu model kullanmaktadır.
  - Solaris Green Threads
  - GNU Portable Threads



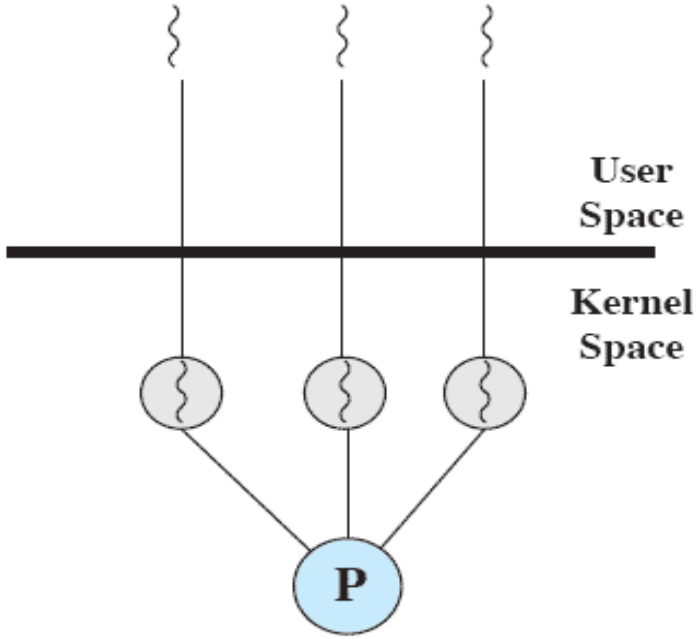
(a) Pure user-level

One-to-One Model

- One-to-one modelinde,
  - bir kullanıcı thread'i
  - bir kernel thread'i ile **eşleştirilir**.
- Eğer **bir thread sistem çağırısını bloklarsa** diğer thread'ler çalışmasına devam eder.
- Birden fazla kernel thread'inin **multicore sistemlerde eşzamanlı çalışmasına izin verir.**
- **Bir user thread için ←→ Bir kernel thread oluşturulması gereklidir**.

One-to-One Model (devam…)

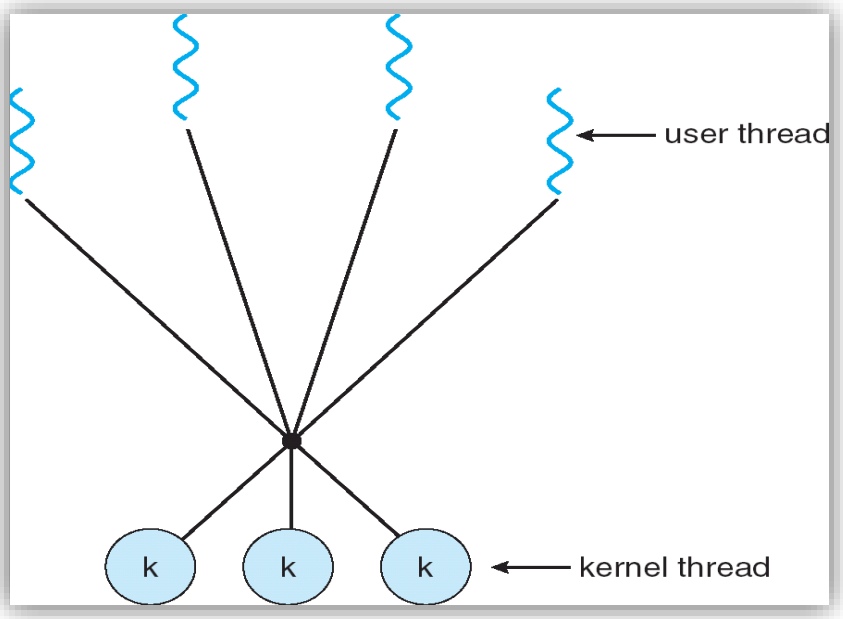
- **Kernel-level thread** modeli olarak da geçer.
- Uygulama tarafından thread yönetimi yapılmaz.
- **Örnekler:** Windows NT/XP/2000, Linux, Solaris 9 ve sonrası



(b) Pure kernel-level

Many-to-Many Model

- Many-to-many modelinde, **çok sayıda kullanıcı thread'i** ile aynı sayıdaki veya daha az sayıdaki kernel thread'i **eşleştirilir** (Solaris 9, Unix işletim sistemleri kullanır.).
- Bir thread sistem çağrısını bloklarsa, kernel başka bir thread'i çalıştırır.



Thread Kütüphaneleri

- Thread kütüphanesi, programcıya **thread oluşturmak** ve **yönetmek** için API sağlar.
- Thread kütüphanesi oluşturulurken **iki farklı yaklaşım** kullanılır:
  - Tüm thread kütüphanesi **kullanıcı alanında** oluşturulur ve kernel desteği yoktur.
  - İşletim sisteminin doğrudan desteklediği **kernel seviyesinde** kütüphane oluşturulur.

Thread Kütüphaneleri (devam…)

- Çoklu thread oluşturmak için **iki farklı strateji** kullanılmaktadır:
  - **Asenkron threading:** Parent, yeni bir child thread oluşturduğunda **eşzamanlı** olarak çalışmasını sürdürür. Web server mimarisi…
  - **Senkron threading:** Parent, child thread oluşturduğunda çalışmasını durdurur ve tüm child threadler sonlandığında çalışmasına devam eder (fork-join strategy).
- **Asenkron threading:**
  - Threadler arasında **veri paylaşımı az** olduğunda,
- **Senkron threading:**
  - Threadler arasında **veri paylaşımı çok** olduğunda kullanılır.

Thread Kütüphaneleri (devam…)

- Günümüzde **3 tane temel thread kütüphanesi** kullanılmaktadır:
  1. **POSIX Pthreads:** User-level veya kernel-level thread kütüphanesi sağlar.
  2. **Windows threads:** Kernel-level thread kütüphanesi sağlar.
  3. **Java threads:** User-level thread kütüphanesi sağlar.

Sonraki Örnekler: Kütüphaneler anlatılırken 1-N arasındaki sayıların toplamını ayrı hesaplayan thread yaratılacaktır.

$$sum = \sum_{i=0}^{N} i$$

Pthreads

- Pthreads, **IEEE1003.1c** standardıyla thread oluşturma ve yönetmek için tanımlanan API'dir.
- Linux, Unix, Mac OS X ve Solaris işletim sistemleri *Pthreads standardını kullanır*.
- **Windows** Pthreads standardını **desteklemez**.
- Pthreads standardında thread'lerin **hepsi ayrı fonksiyonlar** halinde oluşturulur.
- Tüm thread'ler **global scope'ta** tanımlanan ***verileri paylaşırlar***.

Pthreads Örnek 1

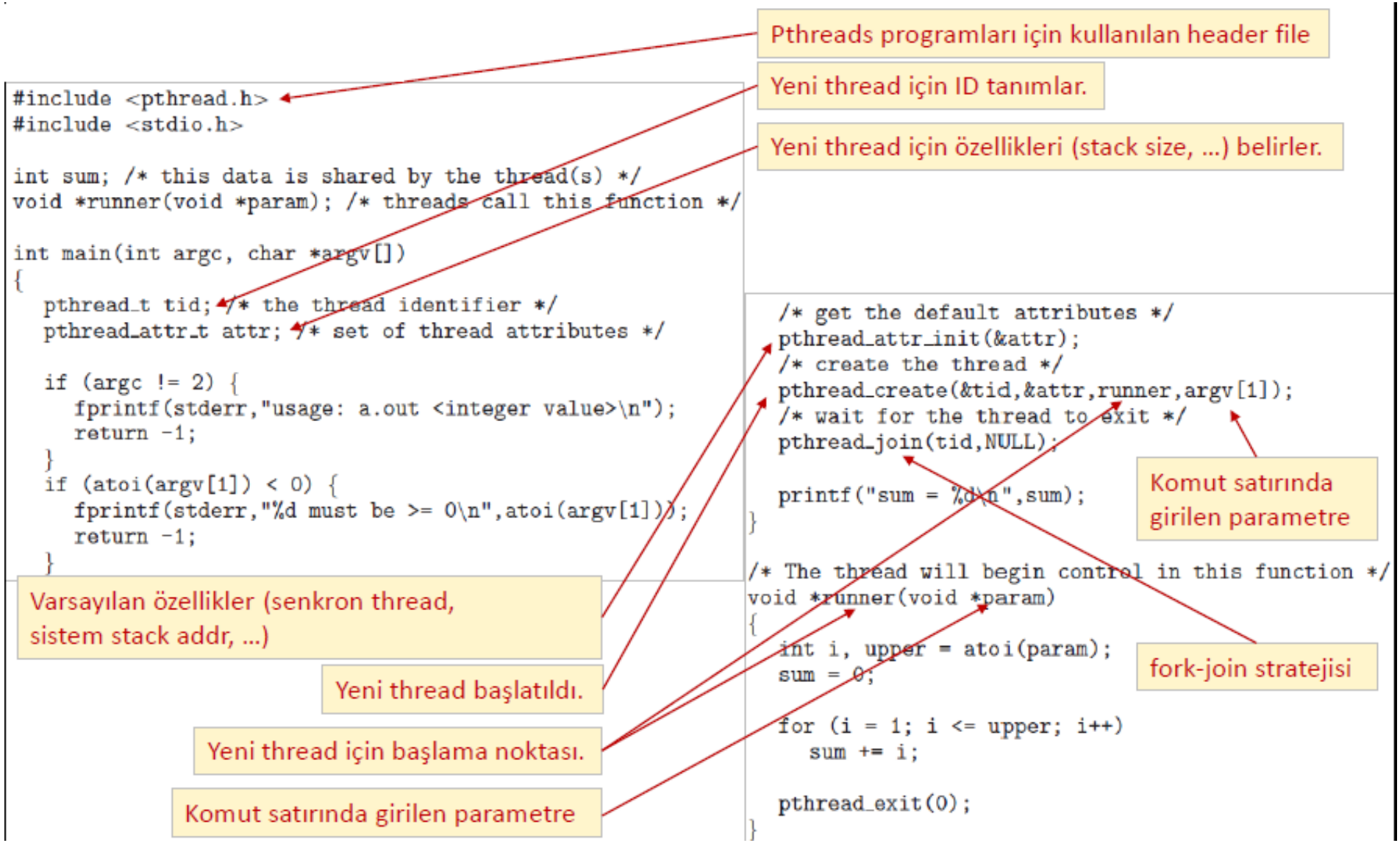
```
#include <pthread.h>
#include <stdio.h>

int sum; /* this data is shared by the thread(s) */
void *runner(void *param); /* threads call this function */

int main(int argc, char *argv[])
{
   pthread_t tid; /* the thread identifier */
   pthread_attr_t attr; /* set of thread attributes */

   if (argc != 2) {
      fprintf(stderr,"usage: a.out <integer value>\n");
      return -1;
   }
   if (atoi(argv[1]) < 0) {
      fprintf(stderr,"%d must be >= 0\n",atoi(argv[1]));
      return -1;
   }
```

Pthreads Örnek 1 (devam…)

```
   /* get the default attributes */
   pthread_attr_init(&attr);
   /* create the thread */
   pthread_create(&tid,&attr,runner,argv[1]);
   /* wait for the thread to exit */
   pthread_join(tid,NULL);

   printf("sum = %d\n",sum);
}

/* The thread will begin control in this function */
void *runner(void *param)
{
   int i, upper = atoi(param);
   sum = 0;

   for (i = 1; i <= upper; i++)
      sum += i;

   pthread_exit(0);
}
```

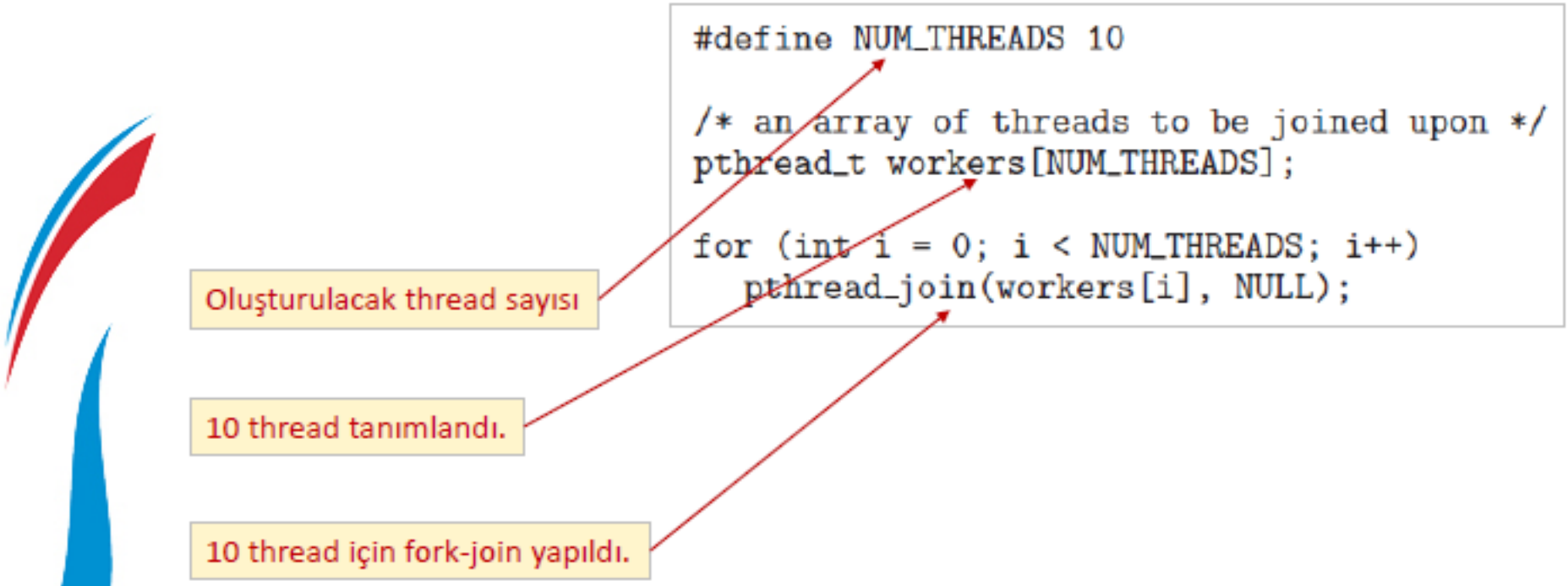
Figure 4.9 Multithreaded C program using the Pthreads API.

Pthreads Örnek 1 Detaylı Açıklama



Pthreads Örnek 2

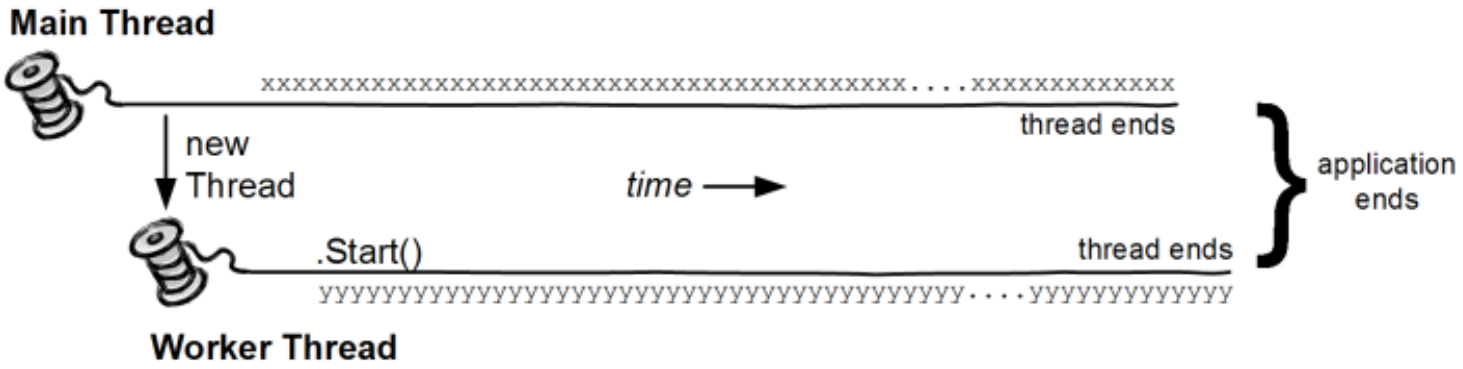
- **Önceki örnekte bir thread oluşturulmuştur**. Çok sayıda thread aşağıdaki örnekteki gibi oluşturulabilir.



.NET Threads Örnek 1

```
class Program
{
    0 references
    static void Main(string[] args)
    {
        Thread t = new Thread(WriteY);          // Kick off a new thread
        t.Start();                               // running WriteY()

        // Simultaneously, do something on the main thread.
        for (int i = 0; i < 1000; i++) {
            Thread.Sleep(10);
            Console.Write("x");
        }
    }

    1 reference
    static void WriteY()
    {
        for (int i = 0; i < 1000; i++)
        {
            Thread.Sleep(40);
            Console.Write("y");
        }

    }
}
```

.NET Threads Örnek 1



.NET Threads Örnek 2

```
public void StartWorker()
{
    Thread worker = new Thread(Calculate);
    worker.IsBackground = true;
    worker.Start();
}

private void Calculate()
{
    long sum = 0;
    for (int i = 0; i < 1000000000; i++)
    {
       sum += i;
     }
}
```

- UI desteği yoktur. Thread'in progress süreci ve tamamlanması takip edilemez veya takibi çok zordur.

.NET Threads Örnek 3(devam…)

- BackgroundWorker

```
private void BGWorker(Label label, int sure)
{
    BackgroundWorker bw = new BackgroundWorker();
    // this allows our worker to report progress during work
    bw.WorkerReportsProgress = true;
    // what to do in the background thread
    bw.DoWork += new DoWorkEventHandler(
    delegate(object o, DoWorkEventArgs args)
    {
        BackgroundWorker b = o as BackgroundWorker;
        // do some simple processing for 10 seconds
        for (int i = 1; i <= 10; i++)
        {
            // report the progress in percent
            b.ReportProgress(i * 10);
            Thread.Sleep(1000 * sure);
        }
    });
```

.NET Threads Örnek 3 (devam…)

- BackgroundWorker (devam…)

```
    // what to do when progress changed (update the progress bar for example)
    bw.ProgressChanged += new ProgressChangedEventHandler(
    delegate(object o, ProgressChangedEventArgs args)
    {
        label.Text = string.Format("{0}% Completed", args.ProgressPercentage);
    });

    // what to do when worker completes its task (notify the user)
    bw.RunWorkerCompleted += new RunWorkerCompletedEventHandler(
    delegate(object o, RunWorkerCompletedEventArgs args)
    {
        label.Text = "Finished!";
    });
    bw.RunWorkerAsync();
}
```

Dolaylı (Implicit) Thread

- Multi-core işlemcilerdeki gelişmelerle birlikte, **uygulamalar** yüzlerce hatta binlerce thread içermektedirler.
- Çok sayıda thread ile **uygulama geliştirmek** oldukça **zordur** ve hata olma olasılığı vardır.
- Thread oluşturma işinin **uygulama geliştiriciler yerine**, ***compiler tarafından yapılması*** günümüzde giderek **popüler** hale gelmektedir.
- Bu stratejiye **implicit threading** denilmektedir.

Dolaylı (Implicit) Thread (devam…)

OpenMP

- OpenMP, C, C++ ve Fortran için yazılmış bir grup **compiler direktifidir**.
- Shared memory yaklaşımını kullanılır.
- OpenMP ile paralel çalışacak kod blokları tanımlanır.
- OpenMP ile **#pragma omp parallel** direktifi paralel çalışacak bloğun hemen başında kullanılır.
- OpenMP farklı türdeki deyimler için ayrı direktifler kullanır.

```
#pragma omp parallel for
for (i = 0; i < N; i++) {
  c[i] = a[i] + b[i];
}
```

Dolaylı (Implicit) Thread (devam…)

OpenMP

```
#include <omp.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
   /* sequential code */

   #pragma omp parallel
   {
      printf("I am a parallel region.");
   }

   /* sequential code */

   return 0;
}
```

Signal Handling

- Unix sistemlerde bir **signal** belirli bir **olayın gerçekleştiğini** gösterir.
- Oluşan olaya karşılık gelen sinyal bir prosese iletilir.
- Oluşan sinyal, **senkron** veya **asenkron** alınabilir.
- Senkron sinyal, sinyalin oluşmasına neden olayı gerçekleştiren prosese iletilir.
- Senkron sinyale illegal hafıza erişimi veya 0'a bölme verilebilir.
- Asenkron sinyal, sinyali oluşturan prosesten başka bir prosese iletilir.
- Asenkron sinyale <ctrl><C> tuşlarına birlikte basmak verilebilir.

Signal Handling (devam…)

- İşletim sistemlerinde sinyaller farklı hedeflere gönderilebilir:
  - Process içerisindeki **sadece bir thread'e** gönderilebilir (Senkron).
  - Process içerisindeki **tüm thread'lere** gönderilebilir <ctrl><C>.
  - Process içerisindeki **bazı thread'lere** gönderilebilir.
  - Bir process için **tüm sinyalleri almak üzere** bir thread atanabilir.
- Senkron sinyaller **sadece oluşturan thread'e** gönderilir.
- Aşağıdaki UNİX fonksiyonu ile ***ID değeri verilen*** prosese iletilir.

```
kill(pid_t pid, int signal)
```

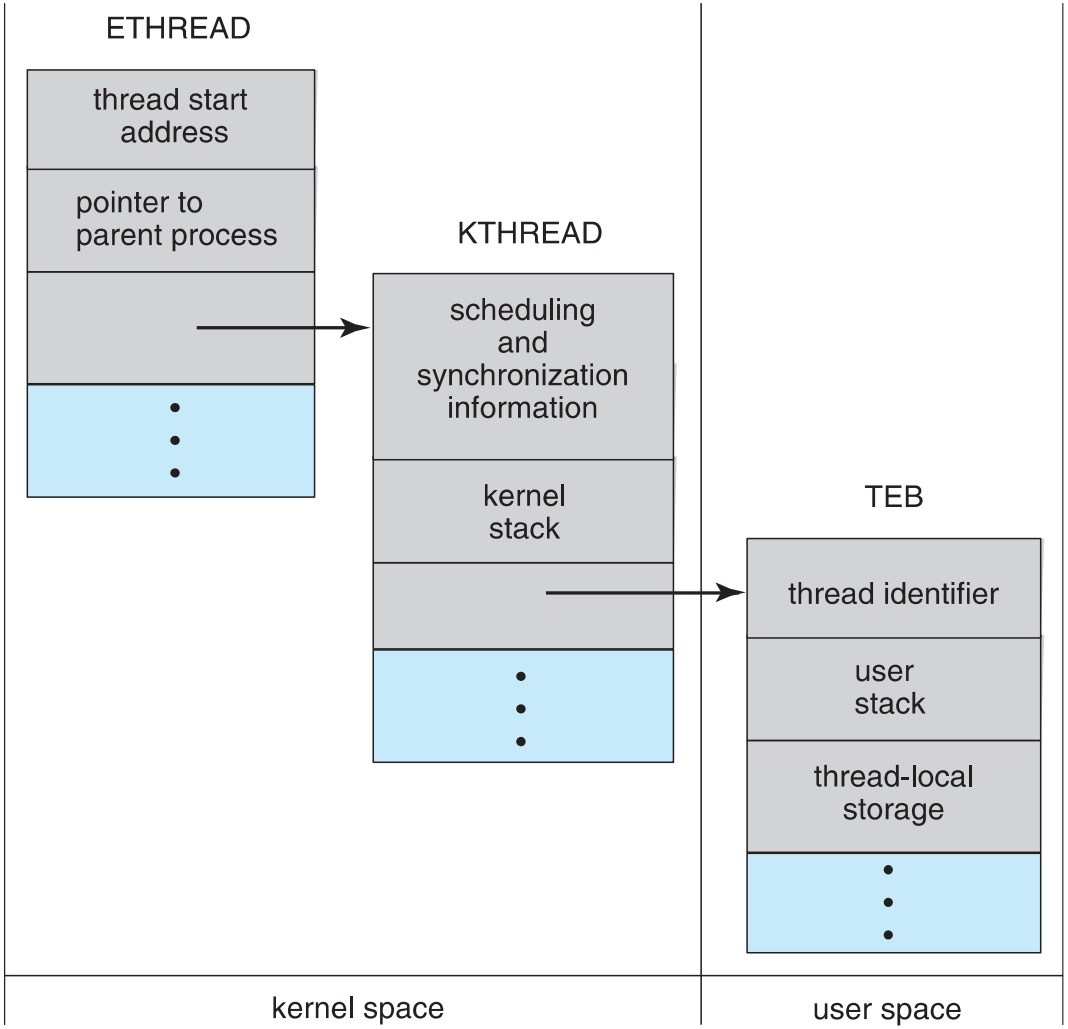
- POSIX Pthreads ile aşağıdaki fonksiyon kullanılır.

```
pthread_kill(pthread_t tid, int signal)
```

Windows Threads

- Microsoft işletim sistemleri için temel API olarak **Windows API** kullanılır.
- Bir Windows uygulaması ayrı bir proses olarak çalışır ve proses **bir veya daha fazla thread** içerebilir.
- Windows, kullanıcı thread'leri ile kernel thread'leri arasında **one-to-one eşleştirme** yapar.
- Bir thread için ayrılan register kümesi, stack, depolama alanı **context** olarak adlandırılır.
- Windows bir thread için aşağıdaki veri yapılarını kullanır:
  - **ETHREAD**: Yürütücü thread blok
  - **KTHREAD**: Kernel thread blok
  - **TEB**: Thread ortam (environment) blok

Windows Threads (devam…)



Linux Threads

- Linux, **proses ve thread arasında net bir ayrım yapmaz** ve her ikisi de **task** olarak adlandırılır.
- fork() sistem çağrısının yanı sıra **clone()** sistem çağrısı ile thread oluşturabilir.
- clone() ile yeni bir görev başlattığında,
  - Dosya sistemi, hafıza aralığı, sinyaller veya açık olan dosyalar paylaştırılabilir.
- Görevler, Linux kernel içerisinde bir veri yapısına sahiptir (**struct task-struct**) ve açık dosyalar, virtual memory ve sinyal bilgilerini gösterir.
- Linux, fork() ile yeni bir görev başlattığında parent task veri yapısı kopyalanır.

Linux Threads (devam…)

- Linux, clone() ile yeni bir görev başlattığında **flag bitlerine göre** veri yapısı oluşturulur.
- Bu bitlerden herhangi birisi set edilmeden de clone() kullanılabilir. Bu durumda clone() ve fork() **benzer işlev görür**.

flag	meaning
CLONE_FS	File-system information is shared.
CLONE_VM	The same memory space is shared.
CLONE_SIGHAND	Signal handlers are shared.
CLONE_FILES	The set of open files is shared.

Çalışma Soruları

- Signal, system call ve interrupt arasındaki fark nedir?
- Proses ve thread arasındaki fark nedir?

İYİ ÇALIŞMALAR…

Yararlanılan Kaynaklar

- **Ders Kitabı:**
  - Operating System Concepts, Ninth Edition, Abraham Silberschatz, Peter Bear Galvin, Greg Gagne
- **Yardımcı Okumalar:**
  - İşletim Sistemleri, Ali Saatçi
  - Şirin Karadeniz, Ders Notları
  - İbrahim Türkoğlu, Ders Notları
- **M. Ali Akcayol, Gazi Üniversitesi Bilgisayar Mühendisliği Bölümü**
- **http://www.albahari.com/threading/**