

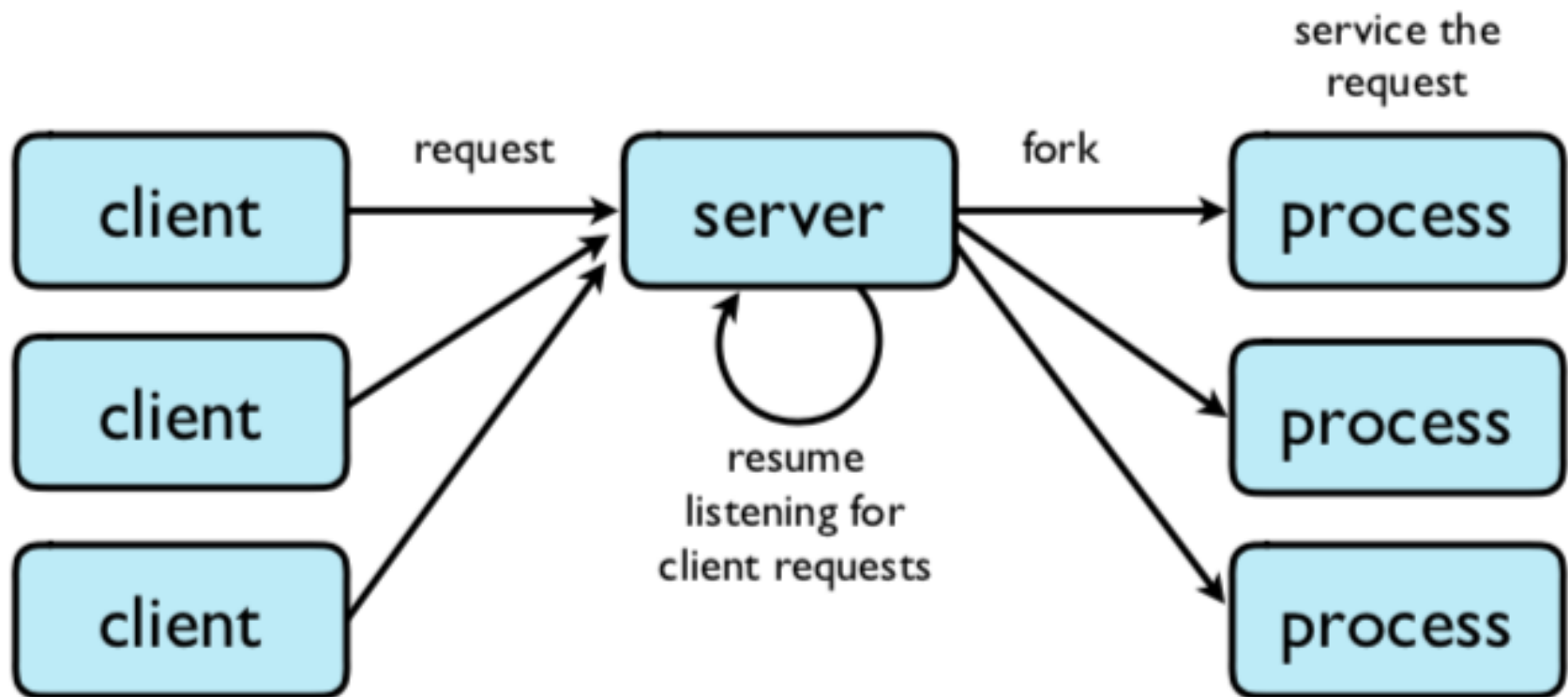
## Bölüm 4: İş Parçacıkları

# Bölüm 4: İş Parçacıkları

- Genel Bakış
- Çoklu İş Parçacığı Modelleri
- İş Parçacığı Kütüphaneleri
- İş Parçacıkları ile İlgili Meseleler
- İşletim Sistemi Örnekleri
- Windows İş Parçacıkları
- Linux İş Parçacıkları

# Hedefler

- İş parçacığı kavramını tanıtmak — çok işlemli bilgisayar sistemlerinde CPU kullanımını sağlayan temel birim
- Pthreads, Win32, ve Java iş parçacığı kütüphanelerinin tanıtımı
- Çok iş parçacıklı programlamada ortaya çıkan meselelerin irdelenmesi



Creating a new process is time consuming and resource intensive.

# Faydaları

- Cevap Verebilirlik (Responsiveness)
- Kaynak Paylaşımı (Resource Sharing)
- Ekonomi (Economy)
  - Solaris: thread creation (1/30) ve context switch (1/5)
- Ölçeklenebilirlik (Scalability)

# Process

**Stack**

**Heap**


**Static data**

**Text**

`main()`

`foo()`

`bar()`

 **PC**

**User space**

---

**Files**

**CPU context**

PC (program counter) and other registers

**Kernel space**

Şöyle Olsaydı ...



Stack

?

Heap

?

Static data

?

Text

main()

← PC<sub>1</sub>

foo()

← PC<sub>2</sub>

bar()

← PC<sub>3</sub>

If multiple program counters are used, how does this affect the use of the **stack, heap, static data, files** and **registers (context)**?

**User space**

Files

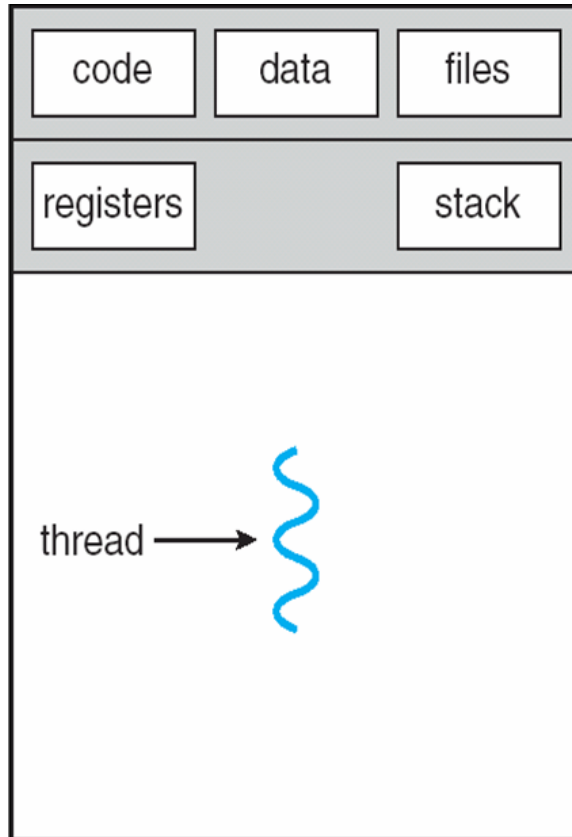
?

CPU context

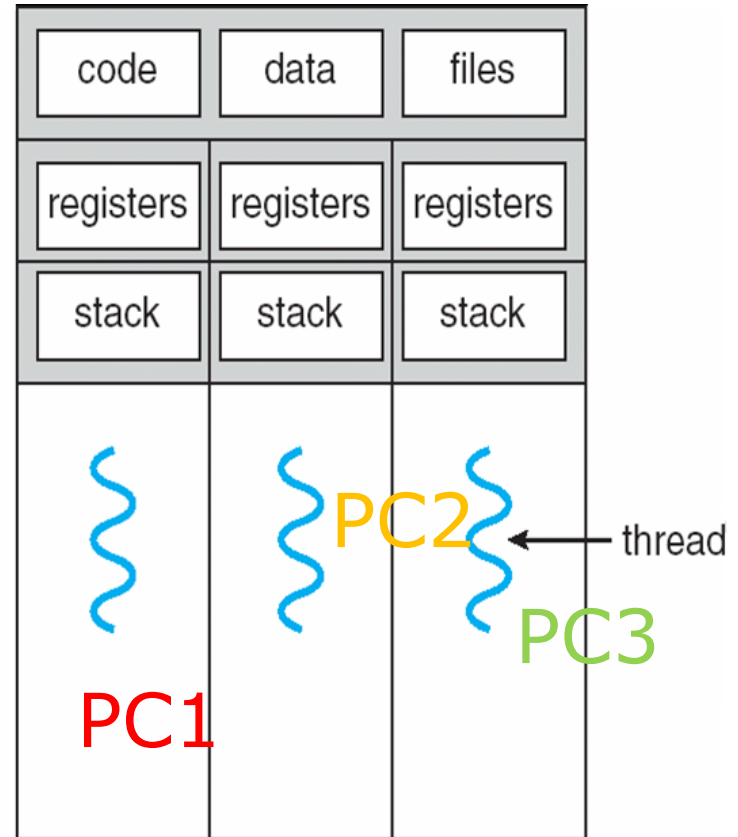
?

**Kernel space**

# Tek ve Çok İş Parçacıklı İşlemler

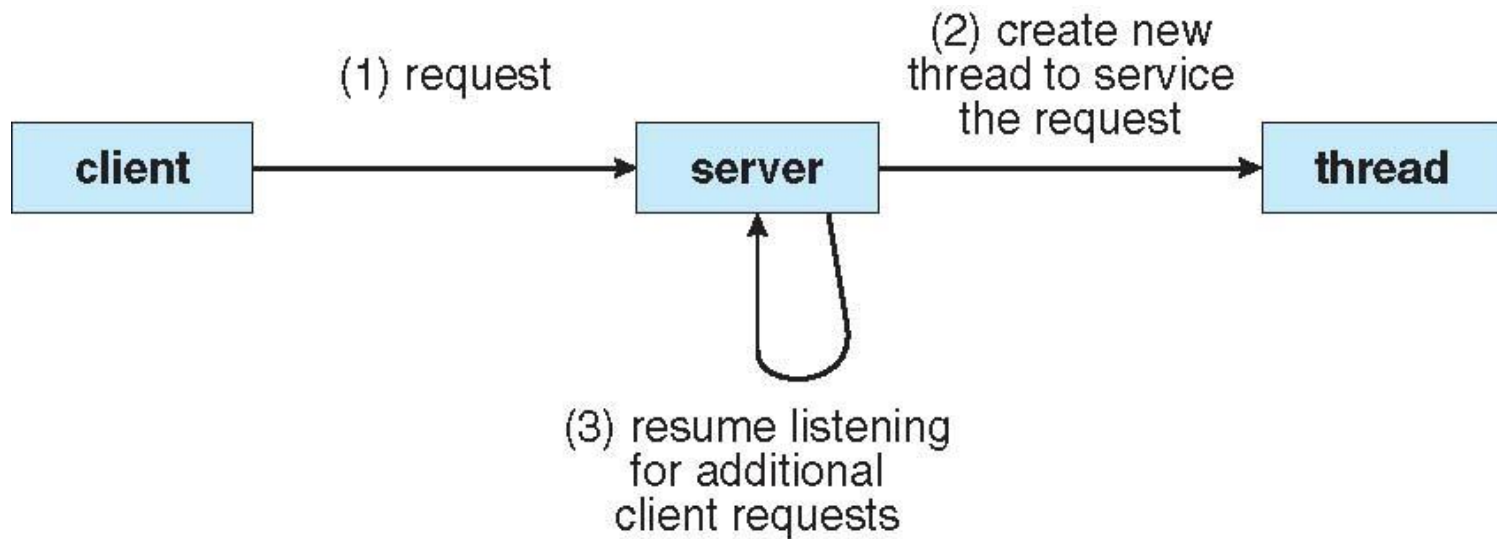


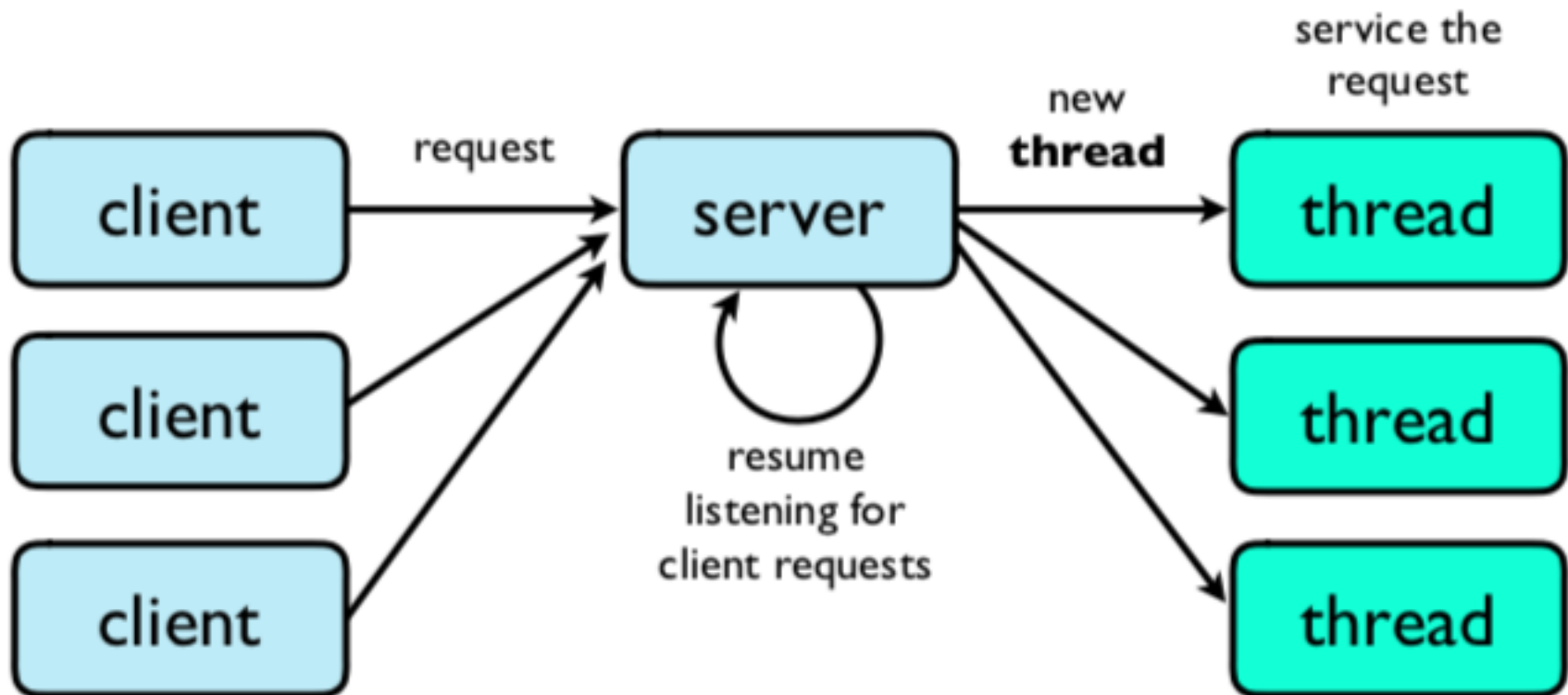
single-threaded process



multithreaded process

# Çok İş Parçacıklı Sunucu Mimarisi





# Tek core CPU'lar

- Başlangıçta CPU'lar tek core'lu üretiliyordu.

★ Tek core'da bir anda sadece 1 komut çalışabiliyor.

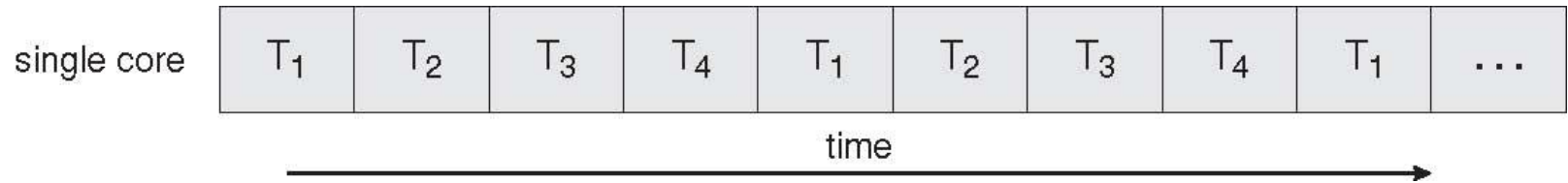


# PIPELINING

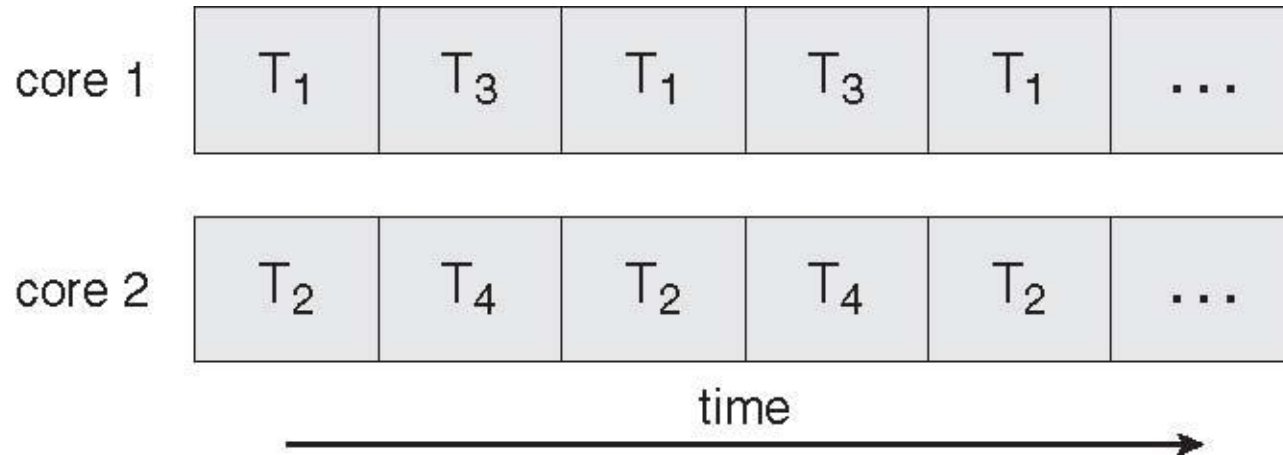


Verimi arttırabilmek için işlemciler pipelining teknolojisini kullanırlar.

# Tek Çekirdekli Sistemde Eş Zamanlı Çalıştırma



# Çok-çekirdekli Sistemde Paralel Çalıştırma





# Thread Programlama

- Çok iş parçacıklı uygulamaların zorlukları:
  - ★ Aktiviteleri bölmek (dividing activities)
  - ★ Denge (balance)
  - ★ Bilgileri Ayırmak (data splitting)
  - ★ Veri bağımlılığı (data dependency)
  - ★ Test ve Hata Ayıklama (testing and debugging)

# User Threadleri- Kullanıcı İş Parçacıkları

- İş parçacığı yönetimi kullanıcı seviyesinde tanımlı iş parçacığı kütüphaneleri ile sağlanır
- Üç ana iş parçacığı kütüphanesi:
  - POSIX **Pthreads**
  - Win32 iş parçacıkları
  - Java iş parçacıkları

# Kernel Threadleri- Çekirdek İş Parçacıkları

- Çekirdek tarafından desteklenirler
- Örnekler
  - Windows XP/2000,
  - Solaris,
  - Linux,
  - Tru64 UNIX,
  - Mac OS X

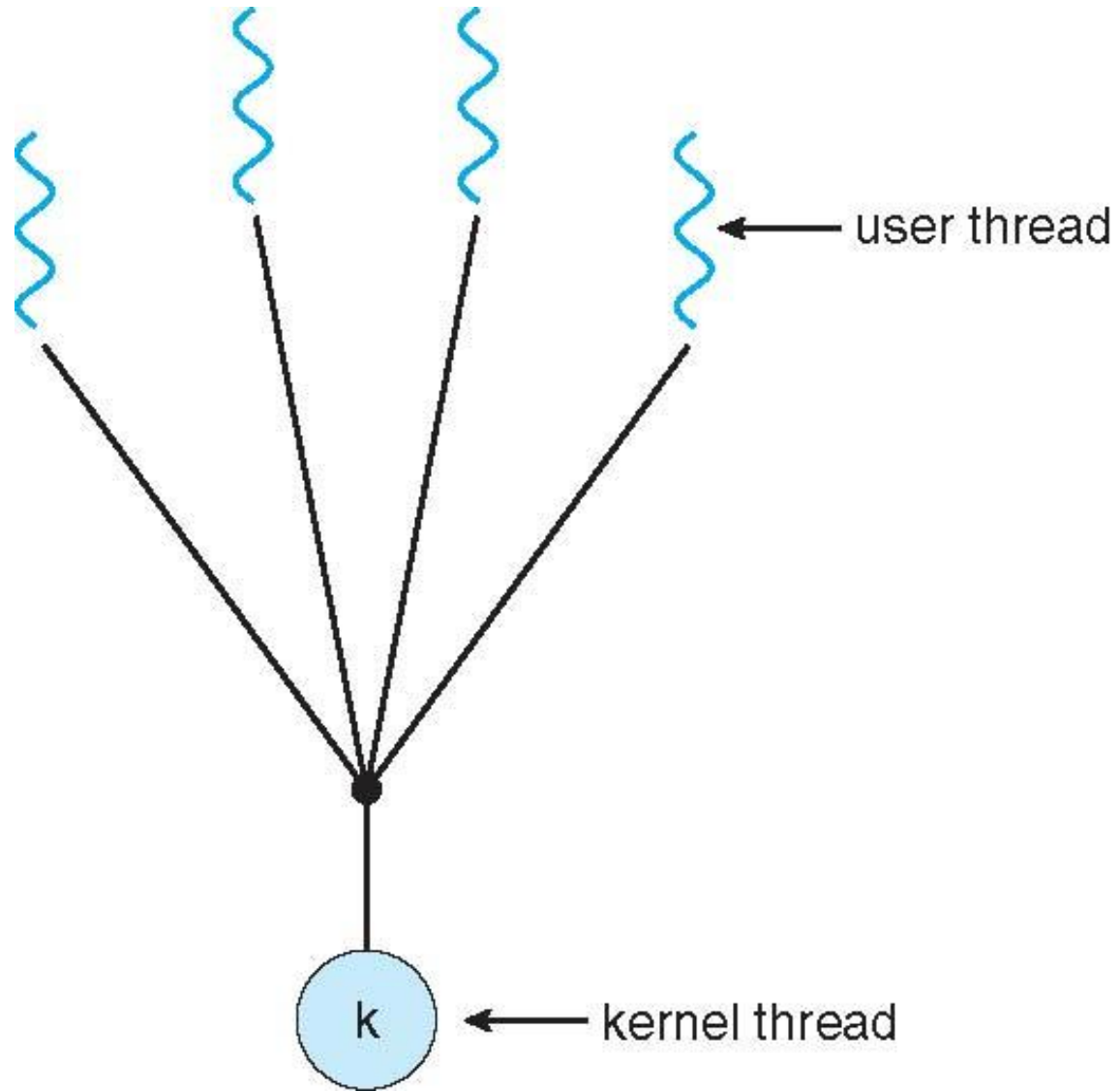
# Çoklu İş Parçacığı Modelleri

- Çoktan-Teke (Many-to-One)
- Teke-Tek (One-to-One)
- Çoktan-Çoka (Many-to-Many)

# Çoktan-Teke (Many-to-One)

- Pek çok kullanıcı seviyesindeki thread tek bir kernel threadi ile eşleşir
- Örnekler:
  - **Solaris Green Threads**
  - **GNU Portable Threads**

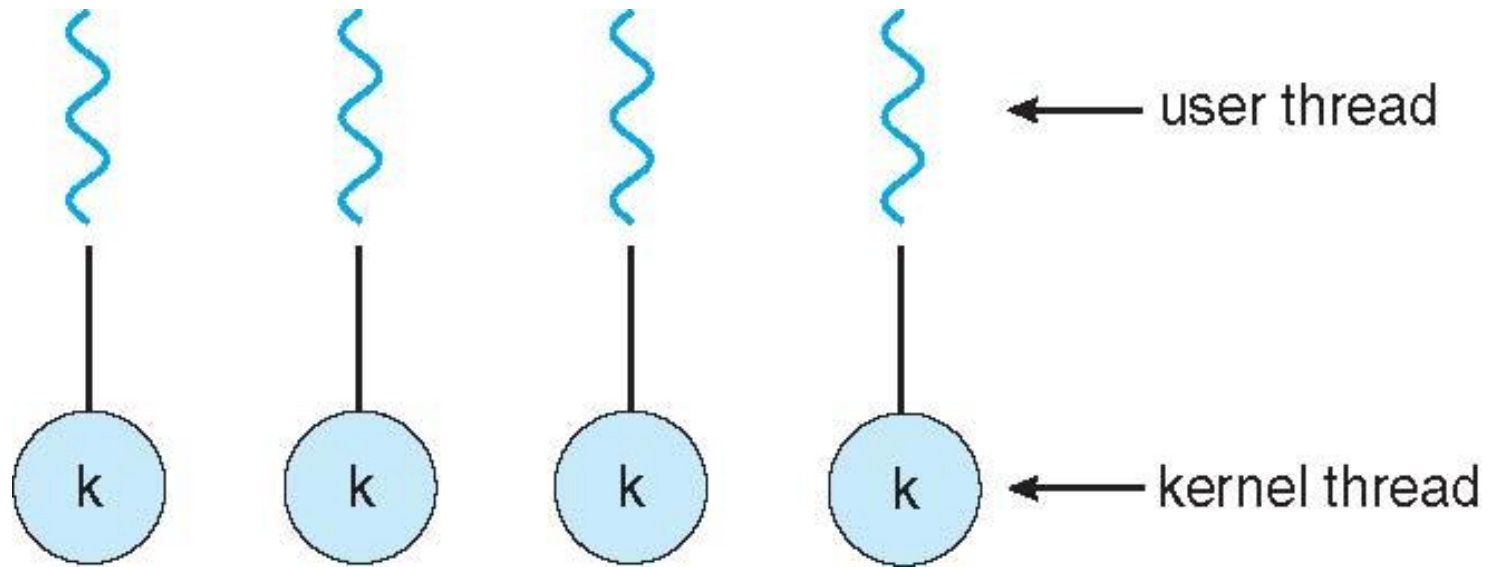
# Çoktan-Teke (Many-to-One) Model



# Teke-Tek (One-to-One)

- Her bir kullanıcı seviyesi iş parçacığı tek bir kernell threadi ile eşleşir
- Örnekler
  - Windows
  - Linux
  - Solaris 9 ve sonrası

# Teke-Tek (One-to-One) Model

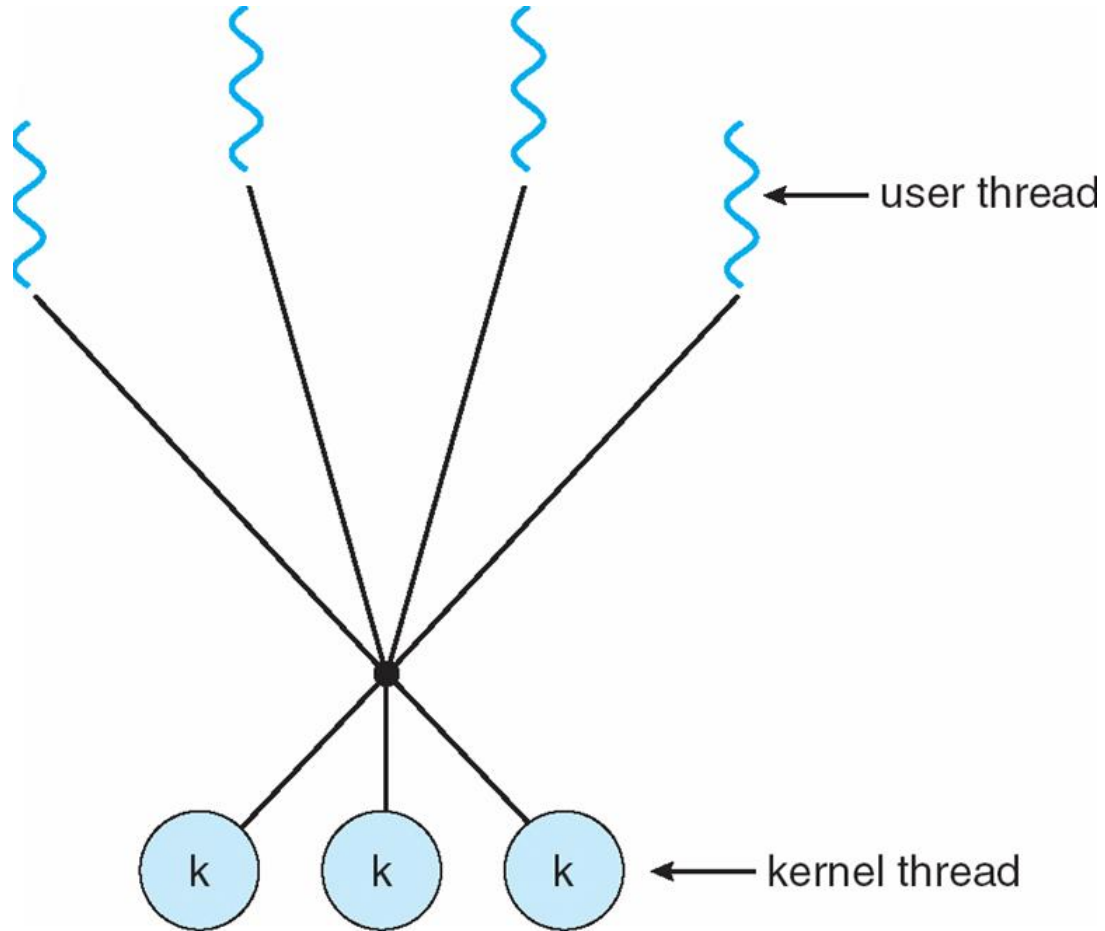




# Çoktan-Çoka (Many-to-Many)

- Pek çok kullanıcı seviyesi iş parçacığı pek çok çekirdek iş parçacığı ile eşleşir
- İşletim sisteminin yeterince çekirdek iş parçacığı oluşturmasını sağlar
- Solaris'in version 9'a kadar olan versiyonları
- *ThreadFiber* paketi ile Windows NT/2000

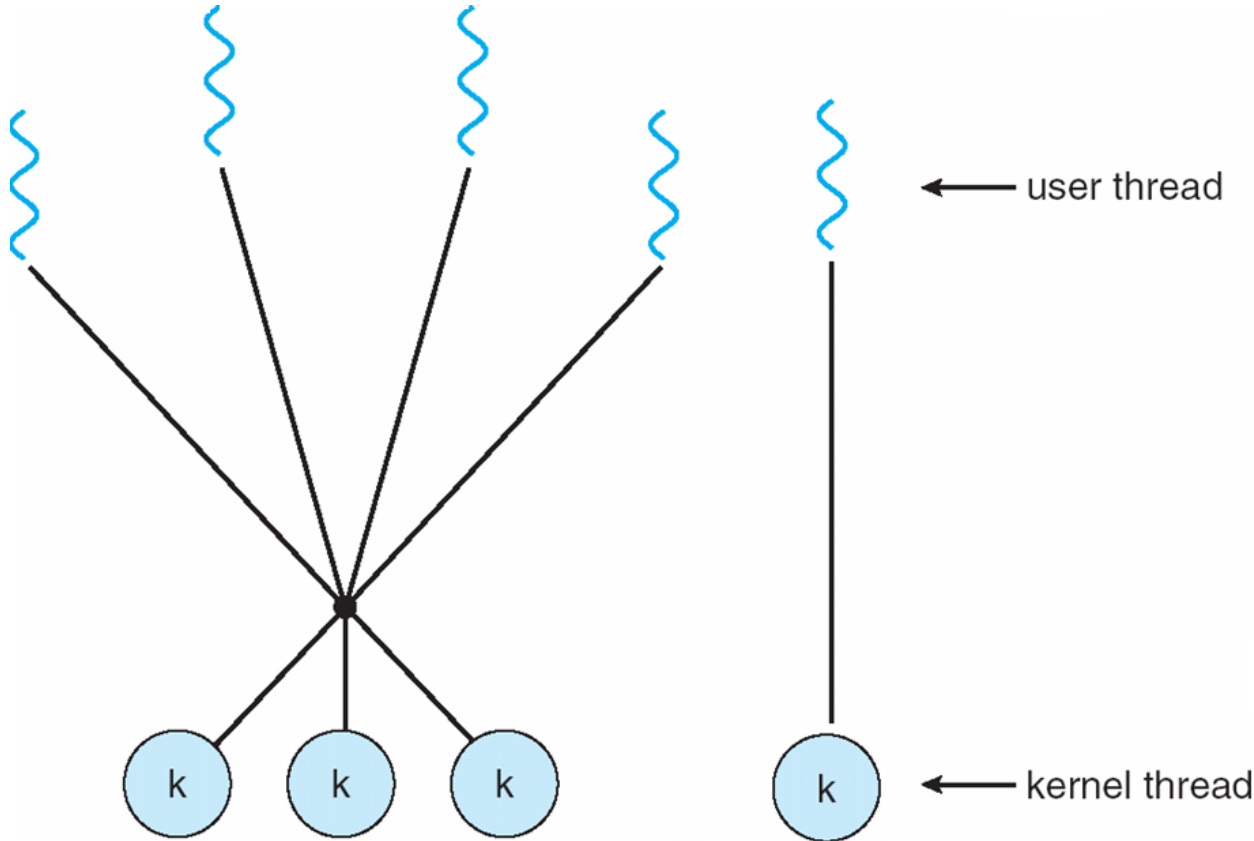
# Çoktan-Çoka (Many-to-Many) Model



# İki-seviye Model

- M:M'e benzer. Farklı olarak kullanıcı iş parçacığının çekirdek iş parçasına bağlanmasına izin verir (**bound** to kernel thread)
- Örnekler
  - IRIX
  - HP-UX
  - Tru64 UNIX
  - Solaris 8 ve öncesi

# İki-seviye Model



# İş Parçacığı Kütüphaneleri

- **İş parçacığı kütüphanesi** programcıya iş parçacıklarının oluşturulmasını ve yönetilmesini sağlayan bir API sunar.
- Gerçekleştirim için iki temel yol
  - Kütüphane tamamen kullanıcı alanında.
  - İşletim sistemi tarafından desteklenen çekirdek seviyesinde kütüphane,

# Pthreads

- Kullanıcı seviyesinde veya çekirdek seviyesinde sunulabilir,
- İş parçacığı oluşturmak ve iş parçacıklarının senkronizasyonunu sağlamak için bir POSIX standardı (IEEE 1003.1c)
- API iş parçacığı kütüphanesinin davranışını tanımlıyor.  
Gerçekleştirim kütüphanenin gerçekleştirmine bağlı
- UNIX işletim sistemlerinde genel olarak kullanılıyor (Solaris, Linux, Mac OS X)

# Java İş Parçacıkları

- Java iş parçacıkları JVM tarafından yönetilir
- Java iş parçacıkları oluşturmanın bir yolu Runnable arayüzünü gerçekleştirmektir

```
public interface Runnable
{
    public abstract void run();
}
```

# Java İş Parçacıkları – Örnek Program

```
class MutableInteger
{
    private int value;
    public int getValue() {
        return value;
    }
    public void setValue(int value) {
        this.value = value;
    }
}

class Summation implements Runnable
{
    private int upper;
    private MutableInteger sumValue;
    public Summation(int upper, MutableInteger sumValue) {
        this.upper = upper;
        this.sumValue = sumValue;
    }
    public void run() {
        int sum = 0;
        for (int i = 0; i <= upper; i++)
            sum += i;
        sumValue.setValue(sum);
    }
}
```



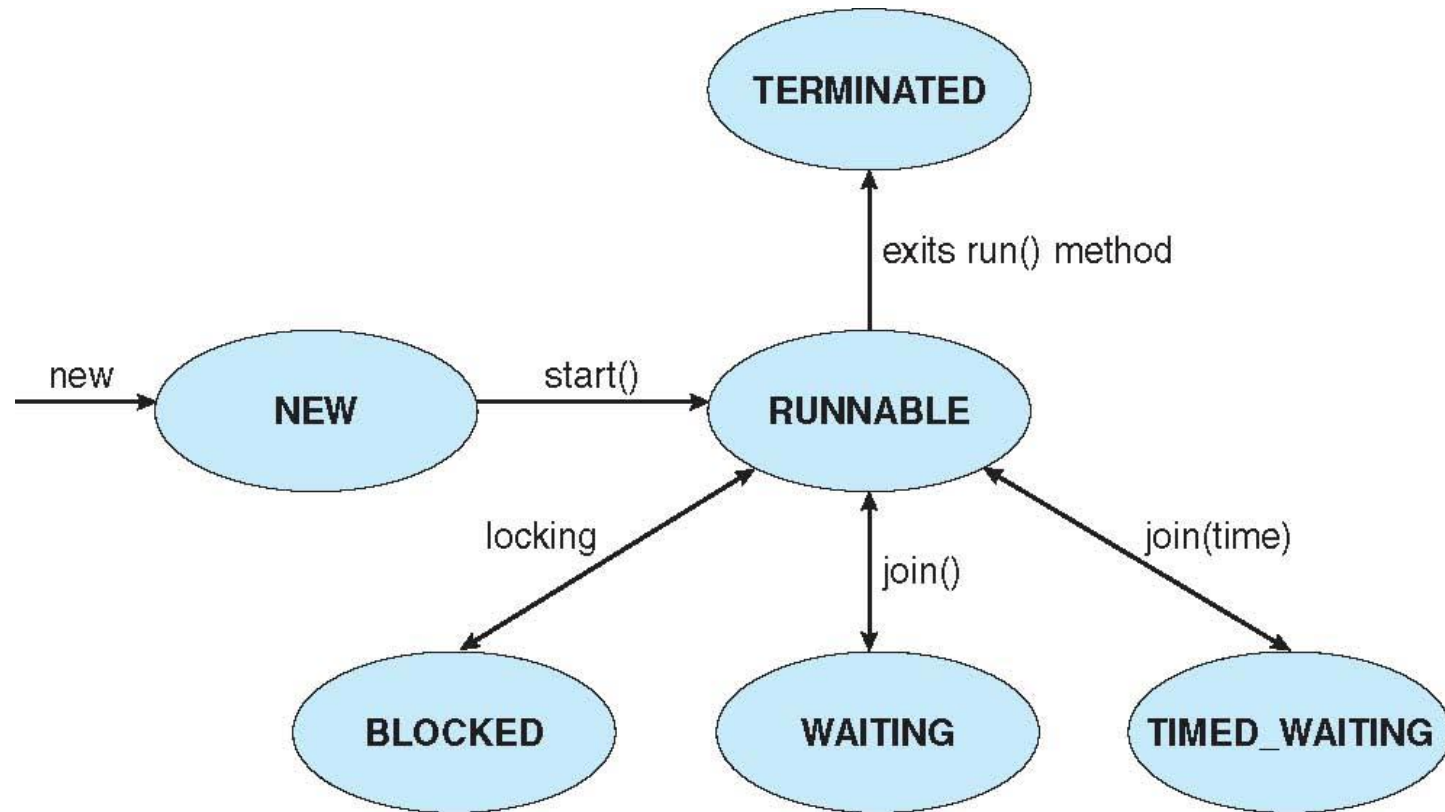
# Java İş Parçacıkları – Örnek Program

```
public class Driver
{
    public static void main(String[] args) {
        if (args.length > 0) {
            if (Integer.parseInt(args[0]) < 0)
                System.err.println(args[0] + " must be >= 0.");
            else {
                // create the object to be shared
                MutableInteger sum = new MutableInteger();
                int upper = Integer.parseInt(args[0]);
                Thread thrd = new Thread(new Summation(upper, sum));
                thrd.start();
                try {
                    thrd.join();
                    System.out.println
                        ("The sum of "+upper+" is "+sum.getValue());
                } catch (InterruptedException ie) { }
            }
        }
        else
            System.err.println("Usage: Summation <integer value>");
    }
}
```

## Diğer Yöntem

```
class ThreadTest extends Thread {  
    public ThreadTest(String str) {  
        super(str);  
    }  
  
    public void run() {  
        for (int i = 0; i < 5; i++) {  
            System.out.println("Loop " + i + ": " + getName());  
            try {  
                sleep((int) (Math.random() * 2000));  
            } catch (InterruptedException e) {  
            }  
        }  
        System.out.println("Test Finished for: " + getName());  
    }  
}
```

# Java İş Parçacığı Durumları



# Thread ile İlgili Mevzular

- **fork()** ve **exec()** sistem çağrılarının anlamı
- **Hedef iş parçacığının iptali**
  - Asenkron veya ertelenen
- **Sinyal işleme**
- **Thread havuzları (Thread pools)**
- **İş parçacığına özgü veri**

## fork() ve exec() Sistem Çağrılarının Anlamı

- **fork()** çağıran iş parçacığının mı yoksa tüm iş parçacıklarının mı kopyasını oluşturur?

# Thread İptali

- Bir threadin işi bitmeden sonlandırılması
- İki genel yaklaşım:
  - **Asenkron iptal** hedef iş parçacığını anında iptal eder
  - **Ertelenen iptal** hedef iş parçacığının düzenli olarak iptal edilmesi gerekip gerekmediğini kontrol etmesini sağlar

# Sinyal İşleme

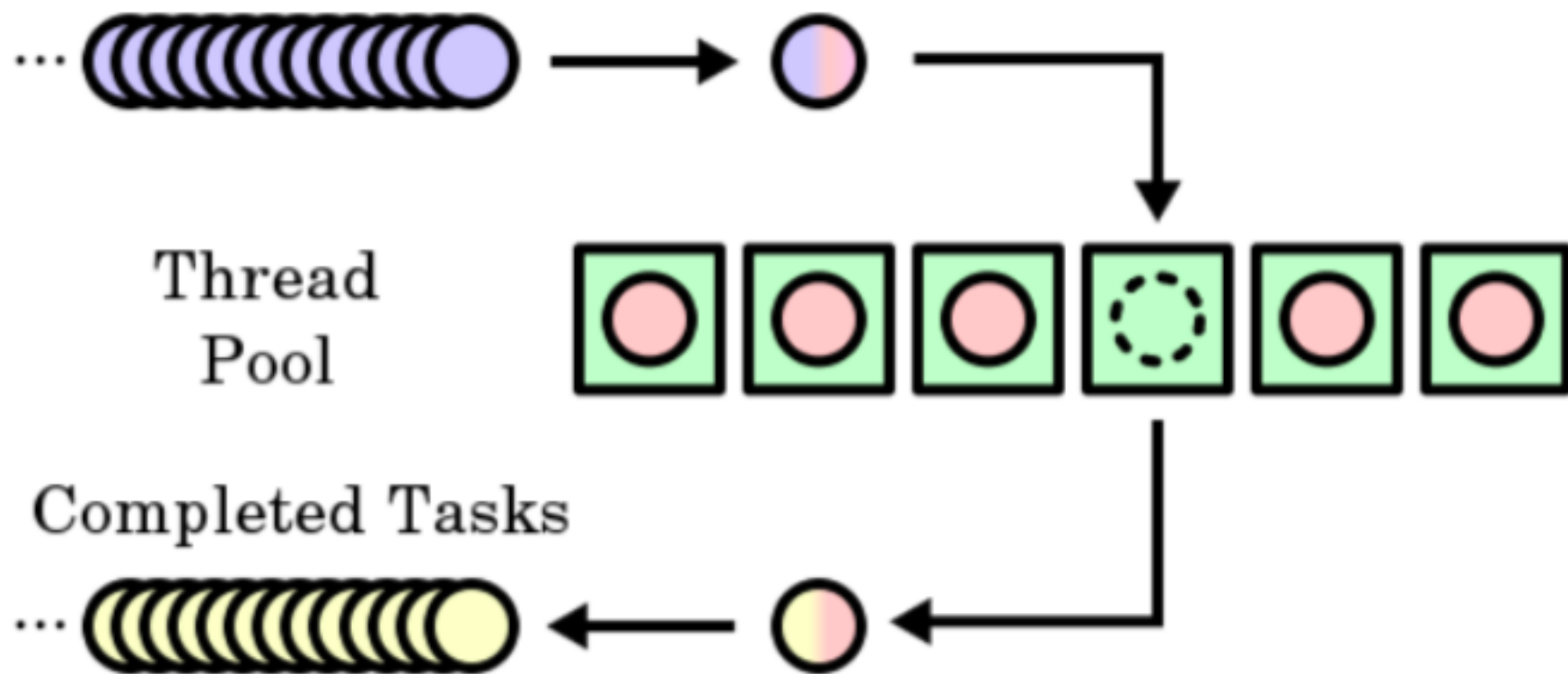
- Sinyaller UNIX sistemlerde belirli bir işlemi, bir olayın gerçekleştiğine dair bilgilendirmekte kullanılır
- Sinyalleri işlemek için bir **sinyal işleyici (signal handler)** kullanılır
  1. Belirli bir olaydan dolayı bir sinyal oluşturulur
  2. Sinyal processe iletilir
  3. Sinyal işlem tarafından işlenir
- Çok iş parçacıklı sistemlerde seçenekler:
  - Sinyali sadece ilgili thread'e ilet
  - Sinyali işlemdeki tüm threadlere ilet
  - Sinyali işlemdeki belli threadlere ilet
  - Sinyalleri işlemek için belli bir thread'i görevlendir

# Thread Pools

- Bir havuzda, kendilerine atanacak işleri beklemek üzere belli sayıda thread oluştur
- Avantajlar:
  - Genellikle varolan bir iş parçacığı ile bir isteği gerçekleştirmek, yeni bir iş parçacığı oluşturarak gerçekleştirmekten biraz daha hızlı
  - Uygulamalardaki iş parçacıklarının sayısı iş parçacığı havuzunun boyutu ile sınırlandırılır



Task Queue



# Thread Specific Data

- Her bir iş parçacığının kendi verilerine sahip olmasına izin verir
- İş parçacığı oluşturma sürecinde kontrolünüz olmadığında (örn: Java'da iş parçacığı havuzu kullanıldığında) işe yarar.

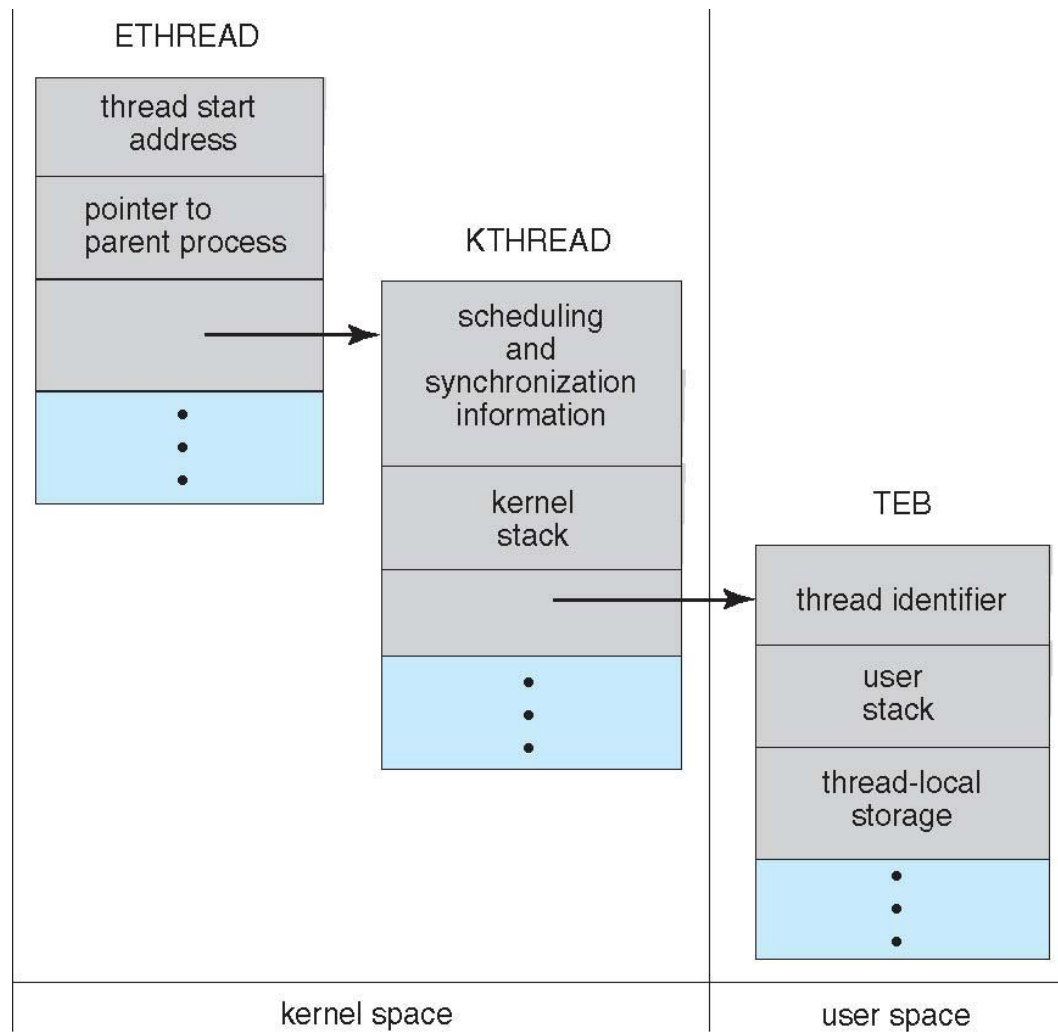
# İşletim Sistemi Örnekleri

- Windows XP threads
- Linux threads

# Windows Threadleri

- Birer-bir modeli çekirdek seviyesinde gerçekleştirir
- Her bir iş parçacığı aşağıdakilere sahiptir
  - İş parçacığı numarası (thread id)
  - Yazmaç kümesi (register set)
  - Ayrı kullanıcı ve kernel yığınları (stacks)
  - Özel veri saklama alanı
- Register kümesi, stackler ve özel veri saklama alanı iş parçacıklarının ortamı (**context**) olarak da bilinir
- Bir iş parçacığının temel veri yapıları:
  - ETHREAD (executive thread block)
  - KTHREAD (kernel thread block)
  - TEB (thread environment block)

# Windows Threads



# Linux Threads

- Linux, iş parçacığı (threads) yerine görev (task) kavramını kullanır
- Yeni thread oluşturma **clone()** sistem çağrısı ile gerçekleştirilir
- **clone()** çocuk görevin, ana görevin (process) adres uzayını kullanmasına izin verir

# Linux Threads- Flags

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.