Bölüm 4: İş Parçacıkları

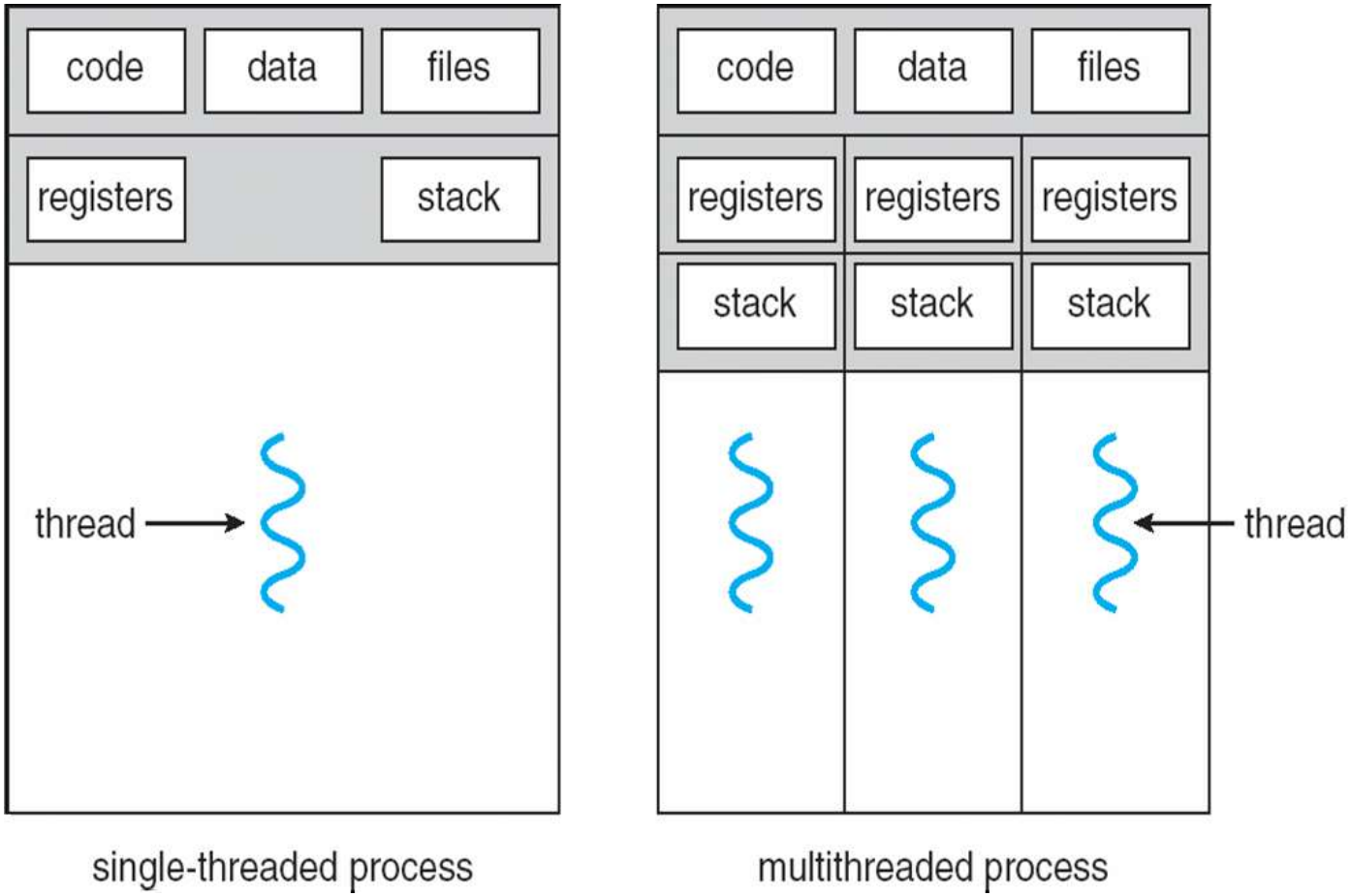
Bölüm 4: İş Parçacıkları

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

Tek ve Çok İş Parçacıklı İşlemler



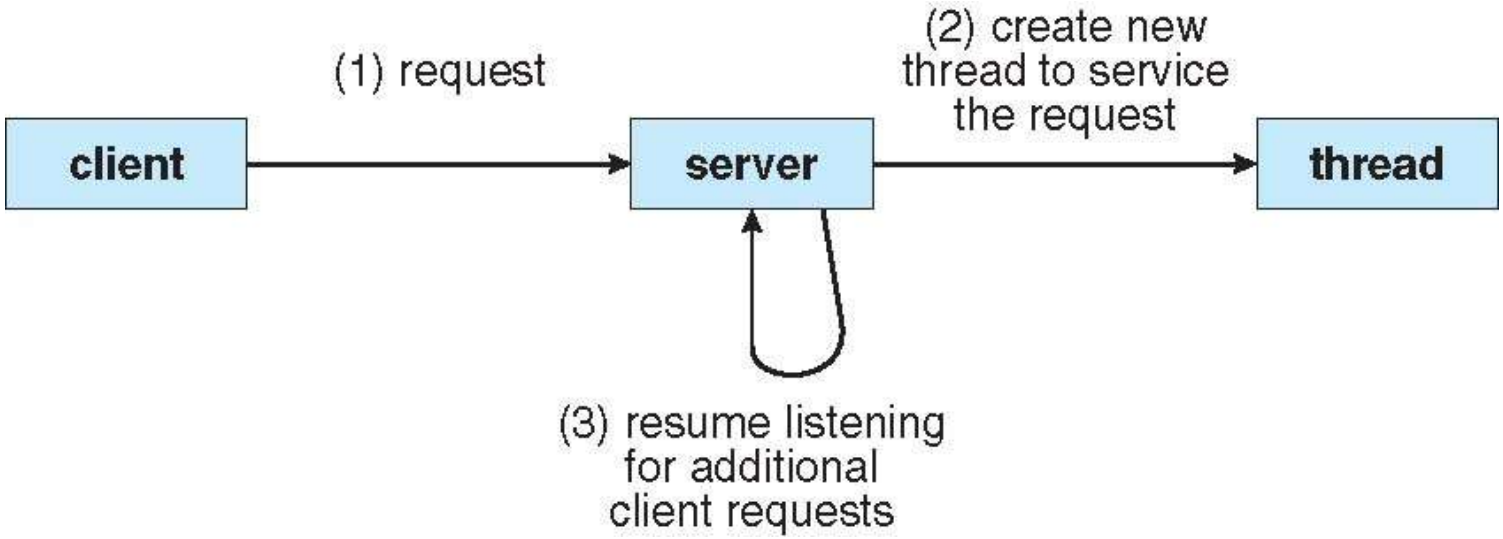
Faydalar

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

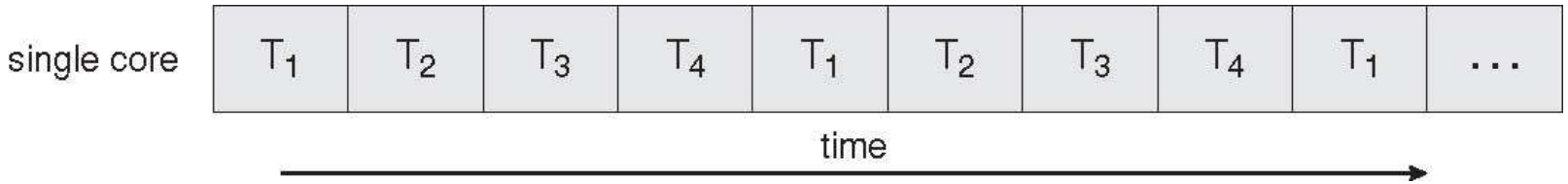
Çok-çekirdekli Programlama

- Çok-çekirdekli sistemler programcıları çok iş parçacıklı uygulamalar yazmaya zorluyor. Bu konudaki zorluklar:
  - **Aktiviteleri bölmek (dividing activities)**
  - **Denge (balance)**
  - **Bilgileri Ayırmak (data splitting)**
  - **Veri bağımlılığı (data dependency)**
  - **Test ve Hata Ayıklama (testing and debugging)**

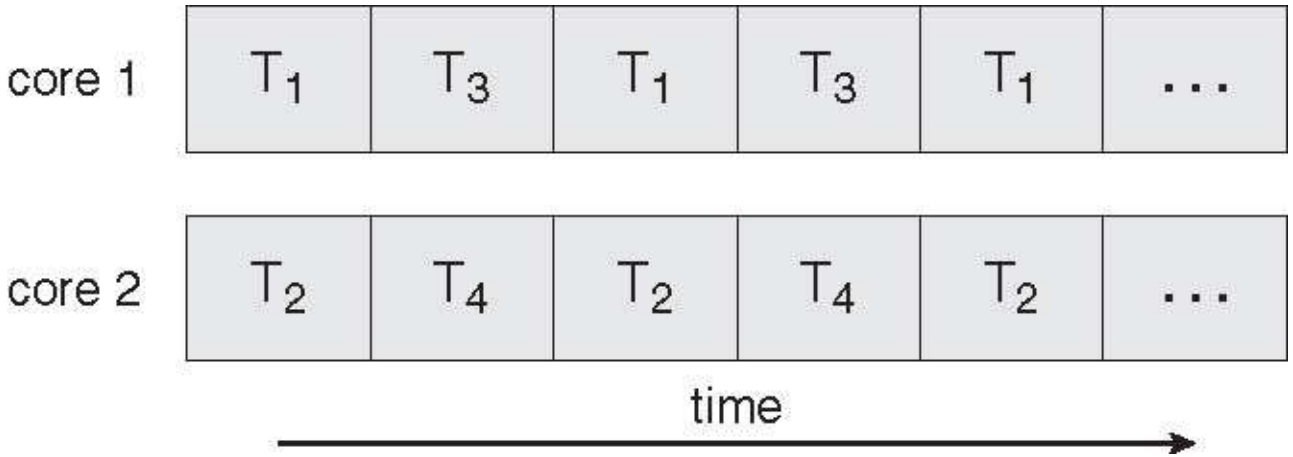
Çok İş Parçacıklı Sunucu Mimarisi



Tek Çekirdekli Sistemde Eş Zamanlı Çalıştırma



Çok-çekirdekli Sistemde Paralel Çalıştırma



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

Çekirdek İş Parçacıkları

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

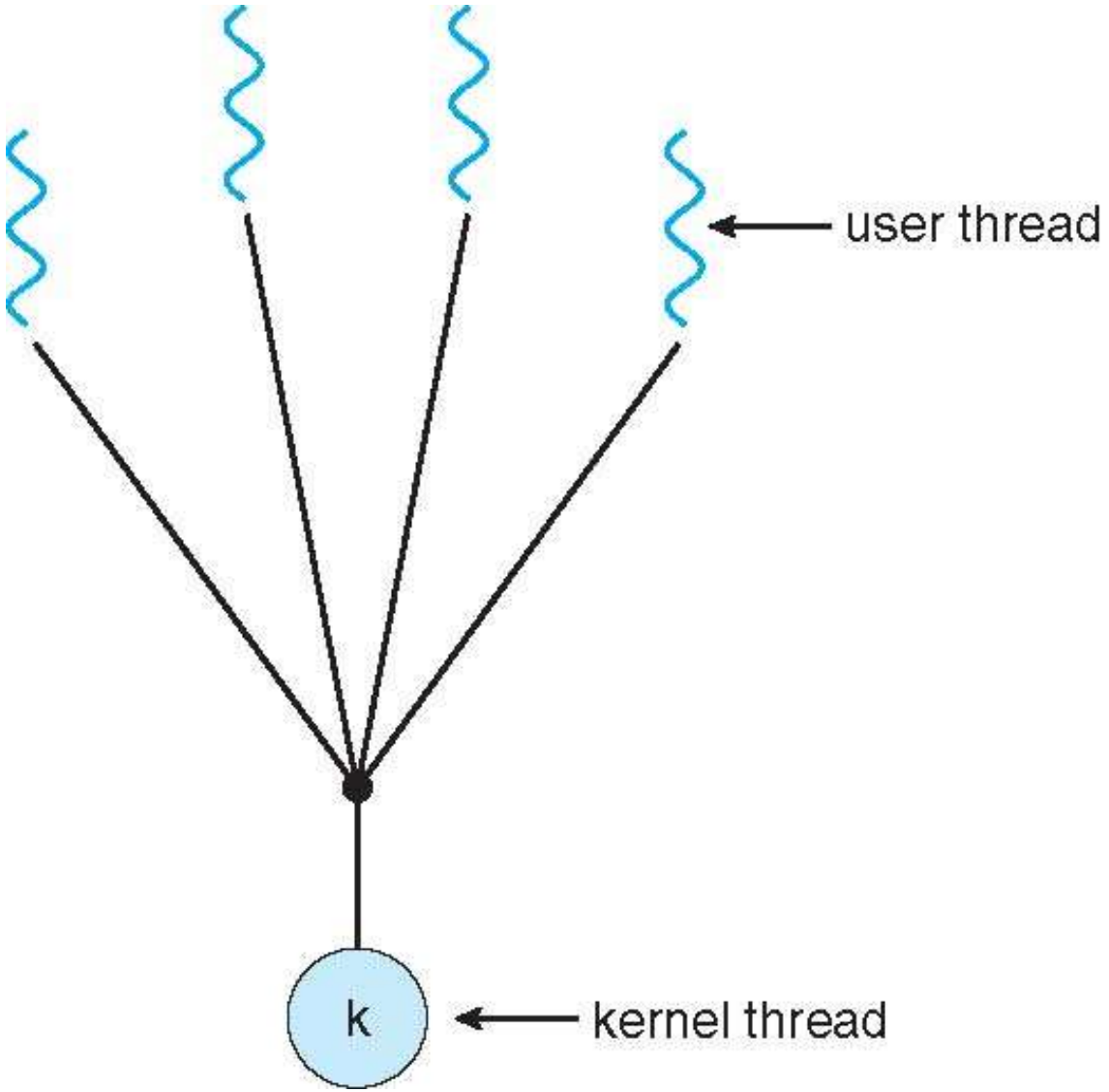
Çoklu İş Parçacığı Modelleri

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

Çoktan-Teke (Many-to-One)

- Pek çok kullanıcı seviyesindeki iş parçacığı tek bir çekirdek iş parçacığı ile eşleşir
- Örnekler:
  - **Solaris Green Threads**
  - **GNU Portable Threads**

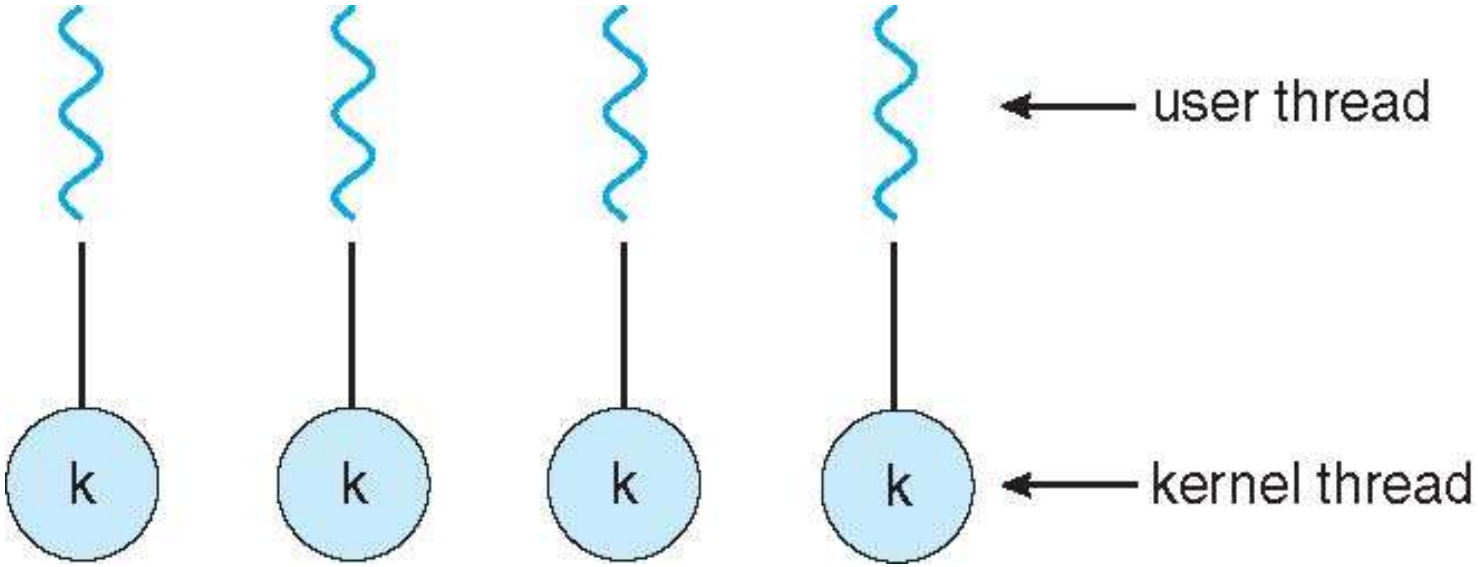
Çoktan-Teke (Many-to-One) Model



Teke-Tek (One-to-One)

- Her bir kullanıcı seviyesi iş parçacığı tek bir çekirdek iş parçacığı ile eşleşir
- Örnekler
  - Windows NT/XP/2000
  - Linux
  - Solaris 9 ve sonrası

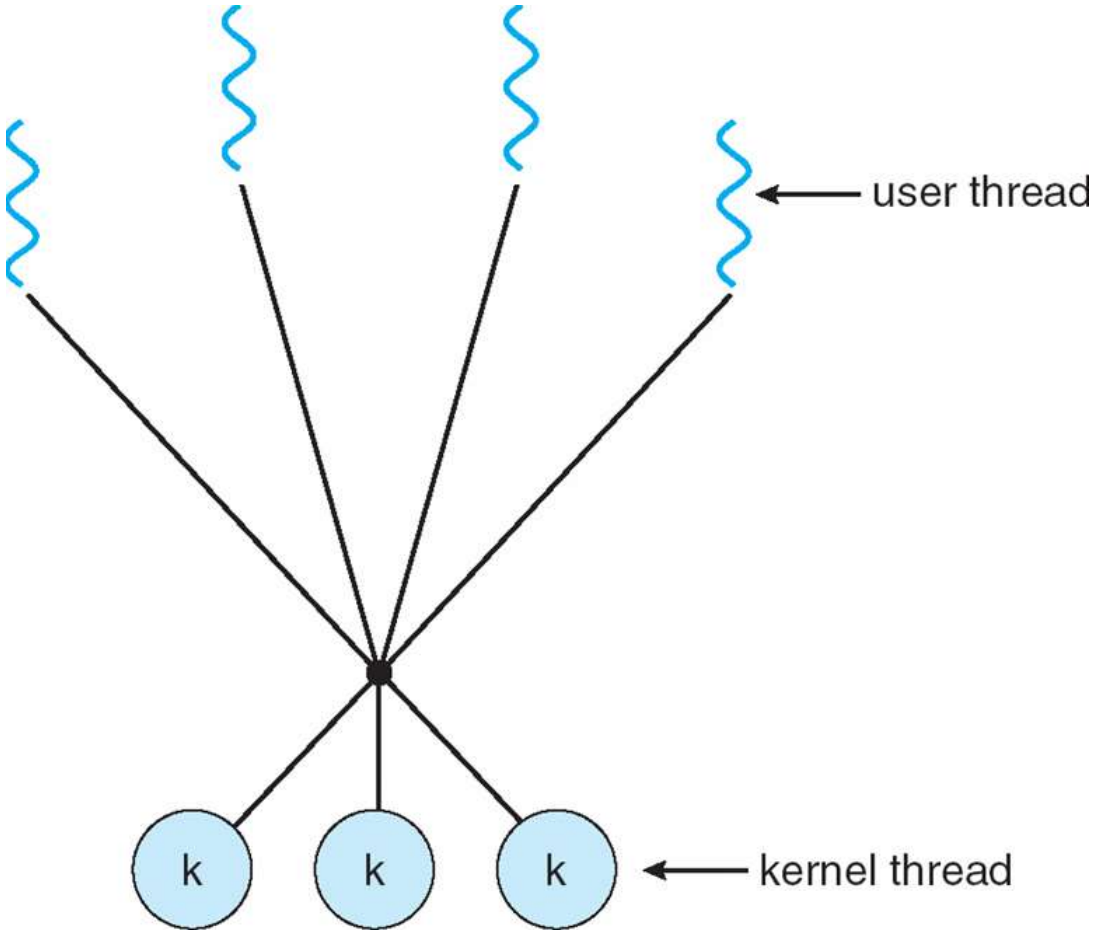
Teke-Tek (One-to-One) Model



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

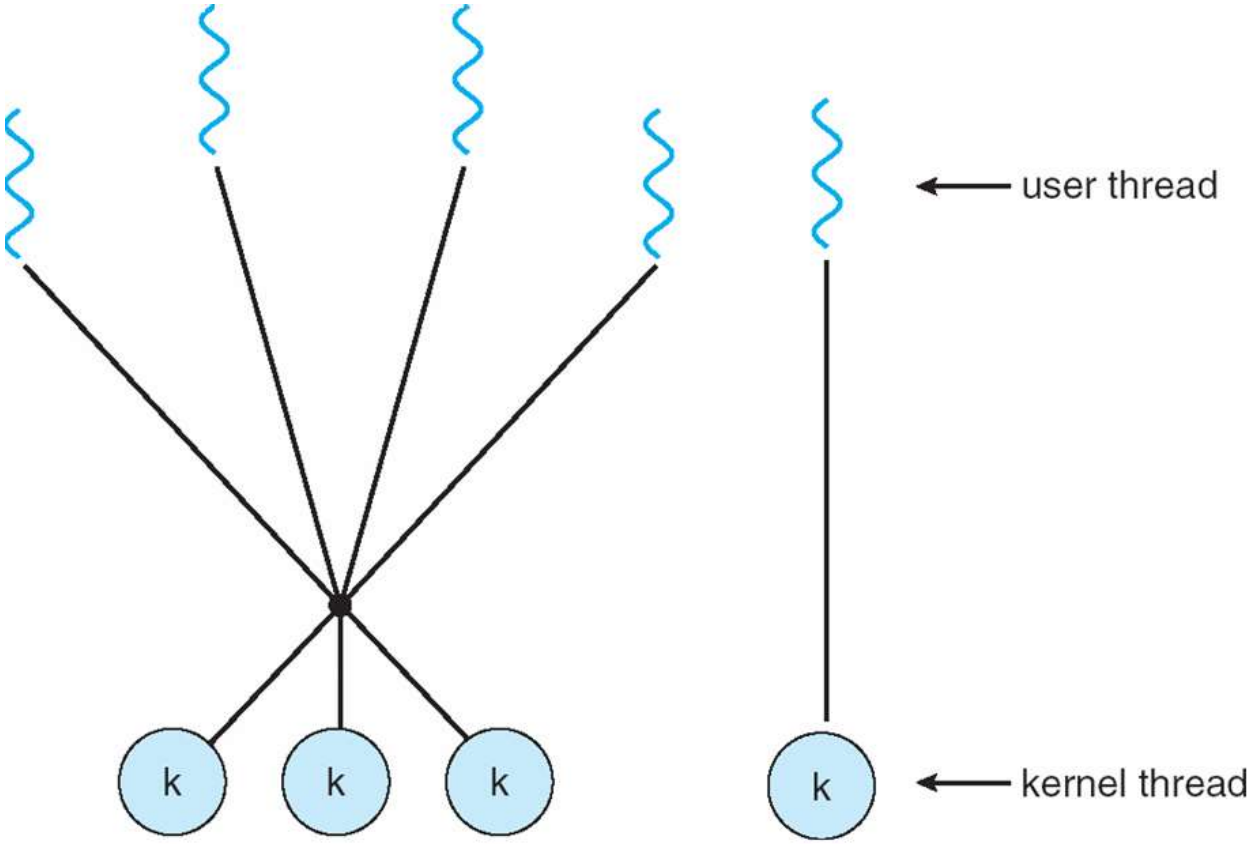
Çoktan-Çoka (Many-to-Many) Model



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 gerçekleştirimine 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

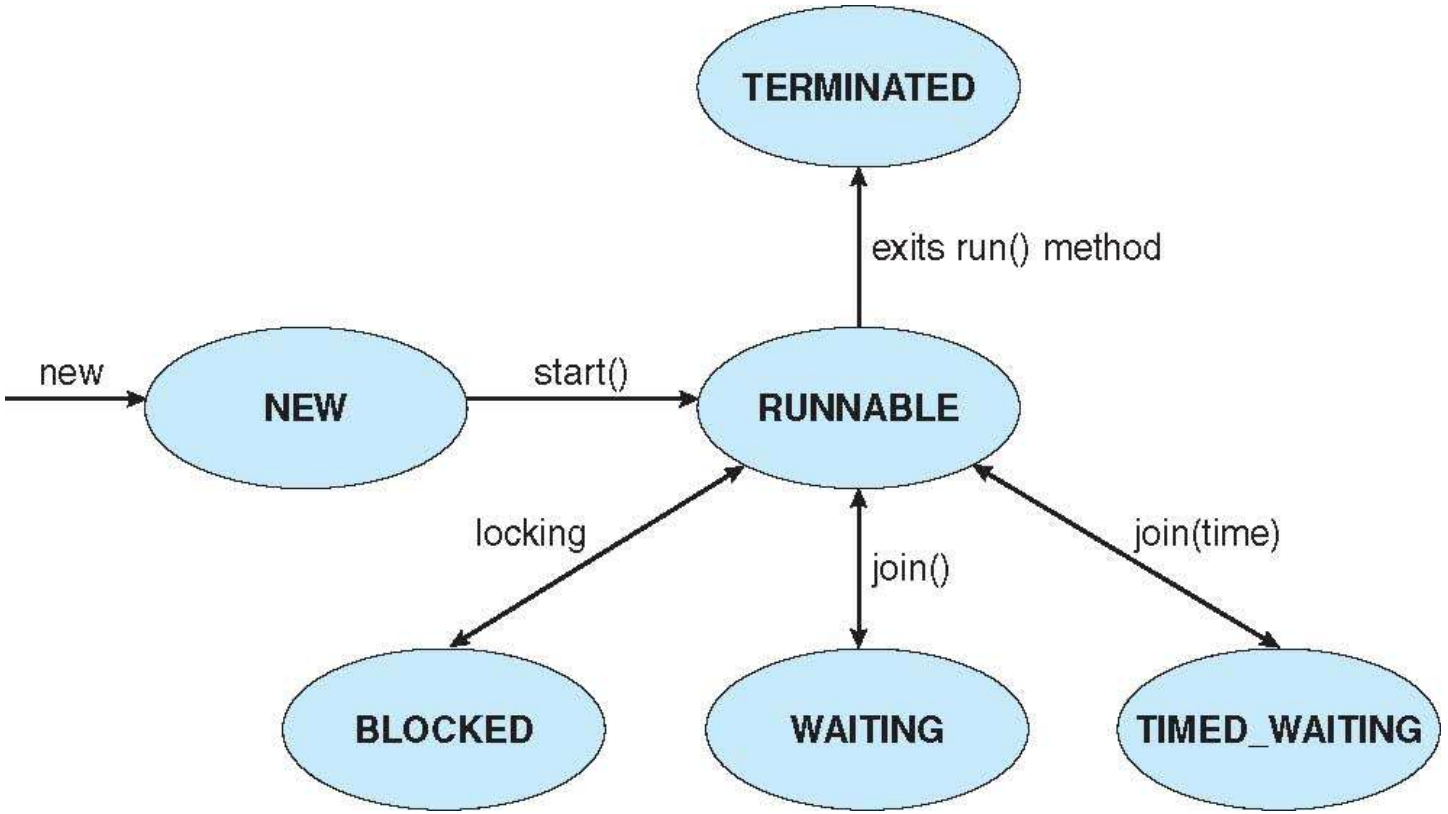
```
class MutableInteger
{
  private int value;
  public int getValue() {
    return value;
  }
  public void setValue(int value) {
    this.value = value;
  }
}

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>");
    }
}
```

Java İş Parçacığı Durumları



Üretici-Tüketici Problemi Çözümü

```
import java.util.Date;

public class Factory
{
    public static void main(String args[]) {
       // create the message queue
       Channel<Date> queue = new MessageQueue<Date>();

       // Create the producer and consumer threads and pass
       // each thread a reference to the MessageQueue object.
       Thread producer = new Thread(new Producer(queue));
       Thread consumer = new Thread(new Consumer(queue));

       // start the threads
       producer.start();
       consumer.start();
    }

}
```

Üretici İş Parçacığı

```
import java.util.Date;

class Producer implements Runnable
{
   private Channel<Date> queue;

   public Producer(Channel<Date> queue) {
      this.queue = queue;
   }

   public void run() {
      Date message;

      while (true) {
         // nap for awhile
         SleepUtilities.nap();

         // produce an item and enter it into the buffer
         message = new Date();
         System.out.println("Producer produced " + message);
         queue.send(message);
      }
   }
}
```

Tüketici İş Parçacığı

```
import java.util.Date;

class Consumer implements Runnable
{
   private Channel<Date> queue;

   public Consumer(Channel<Date> queue) {
      this.queue = queue;
   }

   public void run() {
      Date message;

      while (true) {
         // nap for awhile
         SleepUtilities.nap();

         // consume an item from the buffer
         message = queue.receive();

         if (message != null)
            System.out.println("Consumer consumed " + message);
      }
   }
}
```

İş Parçacıkları ile İlgili Mevzular

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

fork() ve exec() Sistem Çağrılarının Anlamı

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

İş Parçacığı İptali

- Bir iş parçacığının işi bitmeden sonlandırılması
- İki genel yaklaşım:
  - **Asenkron iptal** hedef iş parçacığını anında iptal eder
  - **Ertelenen iptal** hedef iş parçacığının düzenli olarak iptal edilmesi gerekip gerkmediğini 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 işleme iletilir
  3. Sinyal işlem tarafından işlenir
- Çok iş parçacıklı sistemlerde seçenekler:
  - Sinyali sadece ilgili iş parçacığına ilet
  - Sinyali işlemdeki tüm iş parçacıklarına ilet
  - Sinyali işlemdeki belli iş parçacıklarına ilet
  - Sinyalleri işlemek için belli bir iş parçacığını görevlendir

İş Parçacığı Havuzları

- Bir havuzda, kendilerine atanacak işleri beklemek üzere belli sayıda iş parçacığı 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

İş Parçacığına Özgü Veri

- 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

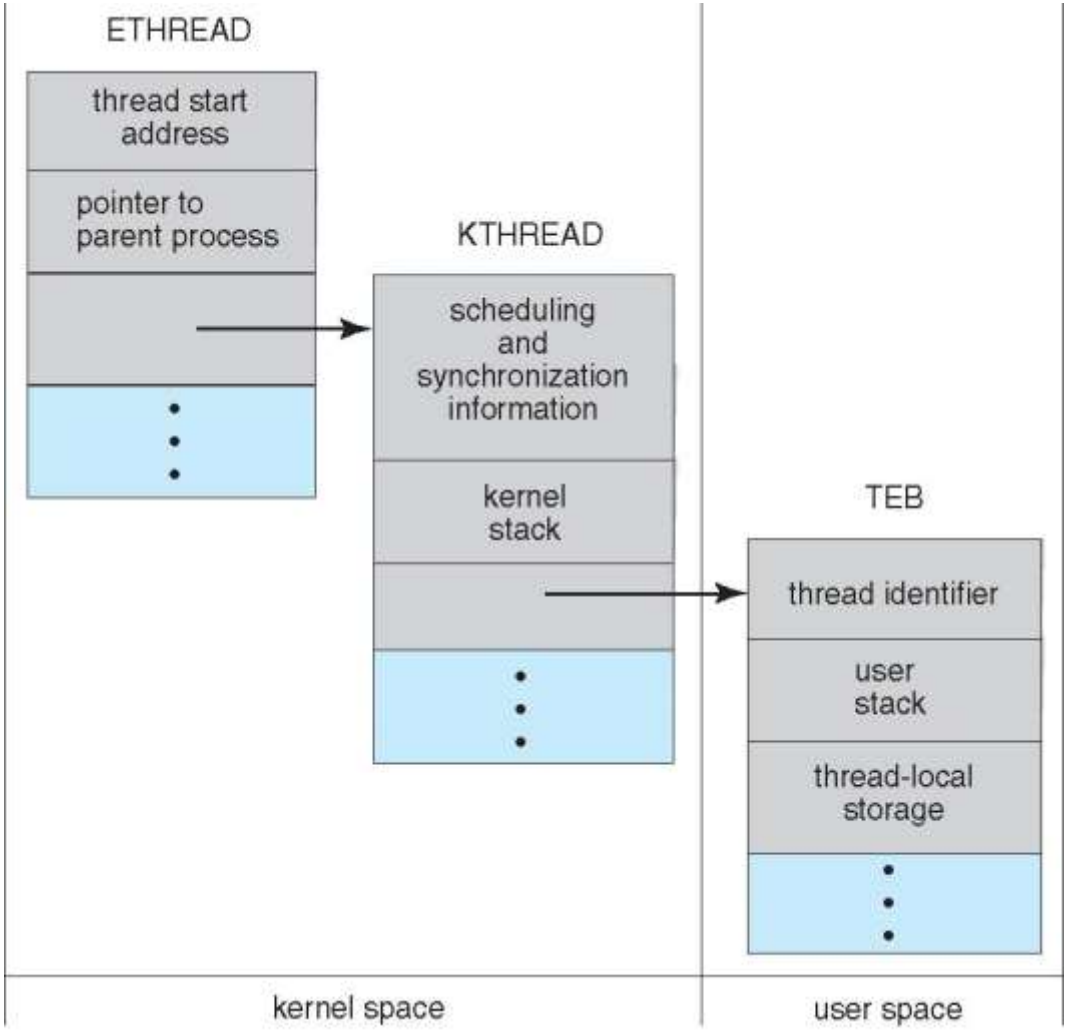
İşletim Sistemi Örnekleri

- Windows XP iş parçacıkları
- Linux iş parçacıkları

Windows XP İş Parçacıkları

- Birer-bir modeli çekirdek seviyesinde gerçekleştirir
- Her bir iş parçacığı aşağıdakilere sahiptir
  - İş parçacığı numarası (thread id)
  - Yazmaç kümesi (register set)
  - Ayrı kullanıcı ve çekirdek yığınları (stacks)
  - Özel veri saklama alanı
- Yazmaç kümesi, yığınlar 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 XP İş Parçacıkları



Linux İş Parçacıkları

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

Linux İş Parçacıkları - 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.