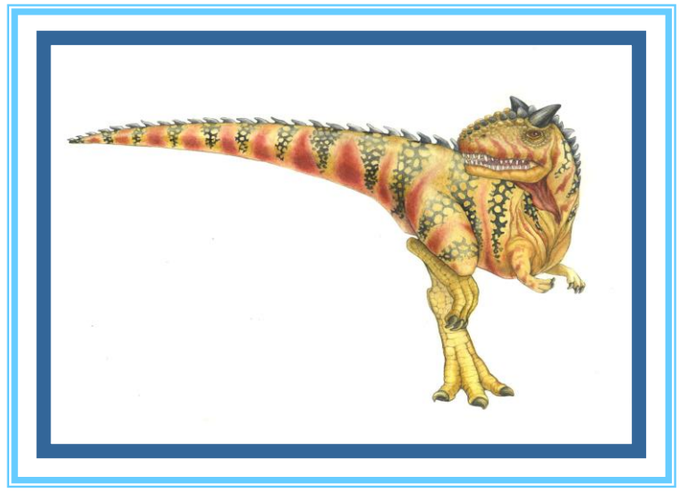
Bölüm 4: İş Parçacıkları (Threads)



Hafta 4: İş Parçacıkları

- Genel Bakış
- Çoklu iş parçacığı (Multithreading) Modelleri
- İş parçacığı Kütüphaneleri
- İş parçacığı sorunları
- İşletim Sistemi Örnekleri
- Windows XP İş parçacıkları
- Linux İş parçacıkları

Hedefler

- İş parçacığı kavramını tanıtmak — çoklu iş parçacığı bilgisayar sistemlerinin temeli olan en temel CPU birimi
- PThreads, Win32 ve Java iş parçacığı kütüphane API'lerini tanıtmak
- Çoklu iş parçacığı programlama ile ilgili sorunları incelemek

Motivasyon

- İş parçacıkları uygulama altında çalışırlar.
- Uygulama içinde birden fazla görev, ayrı iş parçacıkları tarafından gerçekleştirilebilir.
  - Güncellemeleri göstermek
  - Veri çekmek
  - Yazım denetimi
  - Ağ taleplerini yanıtlama
- Proses oluşturma fazla zaman alan ve kaynak tüketen bir işlemdir, iş parçacığı oluşturma ise daha az kaynak ve zaman tüketir.
- Kodu basitleştirmek, verimliliği arttırmak
- Çekirdekler genellikle çoklu iş parçacığı olarak çalışırlar.

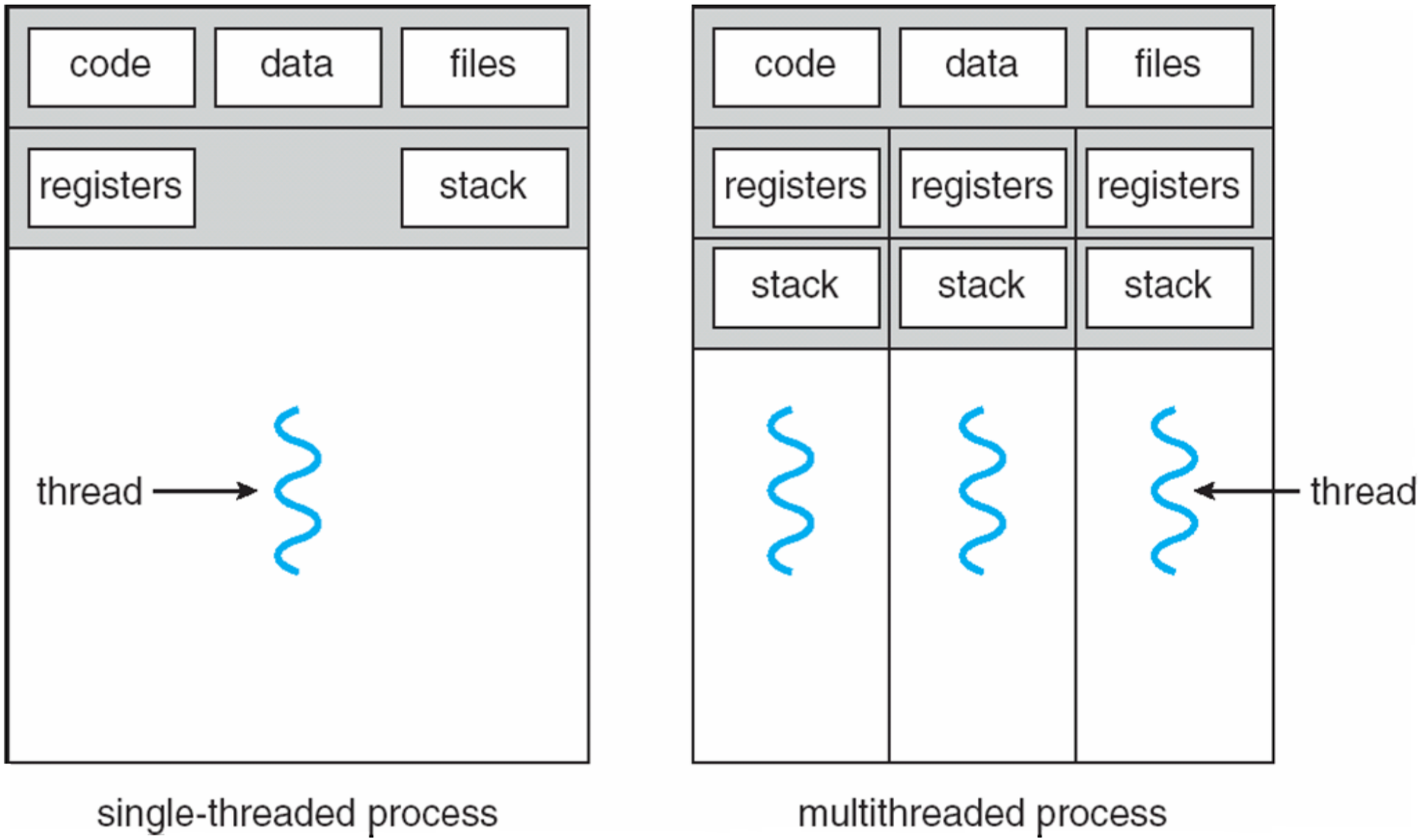
İş parçacığı ne demektir? (Bilişim Terimleri Sözlüğü-ODTU)

- *izlek - iş parçacığı*
  - Bir bilgi işleme sürecinde gerçekleştirilebilecek en küçük işlem birimi

İş parçacığı ne demektir? (wikipedia)

- **İş parçacığı (iş parçacığı)** bilgisayar biliminde, bir programın kendini eş zamanlı olarak çalışan birden fazla iş parçasına ayırabilmesinin bir yoludur.

Tekli ve Çoklu iş parçacıklı Prosesler



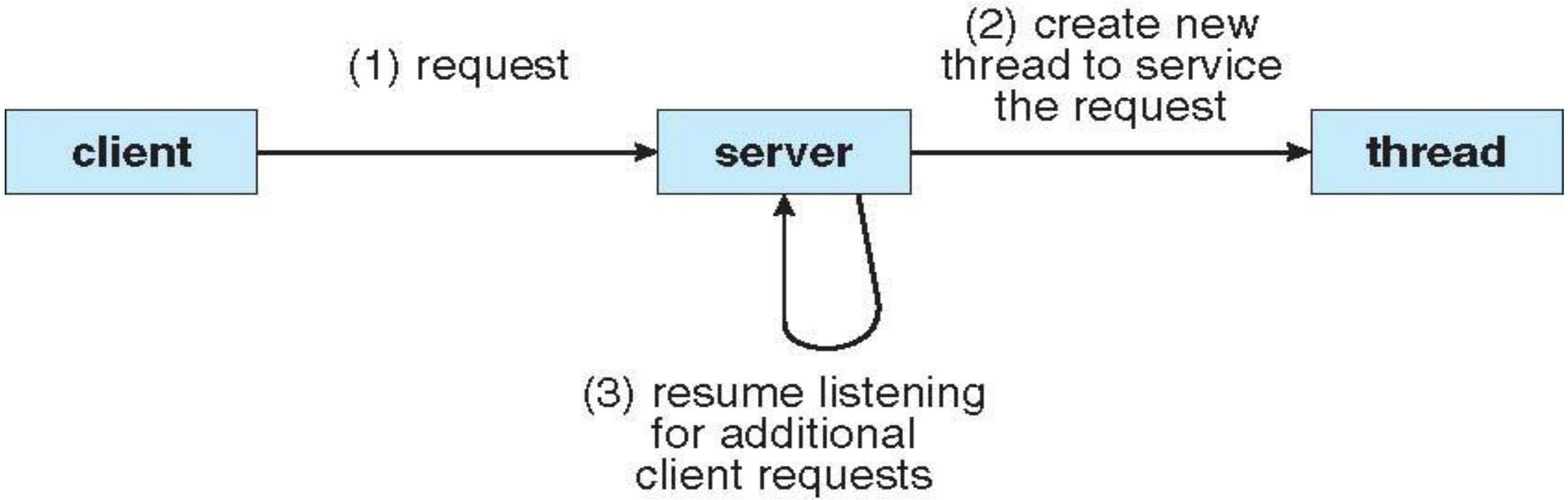
Faydaları

- Duyarlılık (Responsiveness)
- Kaynak Paylaşımı
- Tasarruf
- Ölçeklenebilirlik

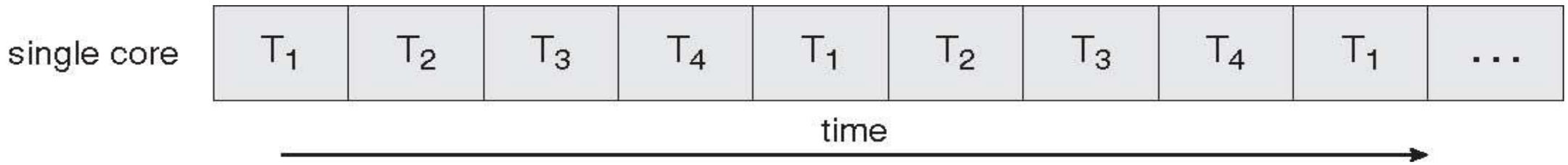
Çok Çekirdekli (Multicore) Programlama

- Çok çekirdekli sistemler programcılar için aşağıdaki zorlukları içerir :
  - **Aktiviteleri bölme**
  - **Denge**
  - **Veri Bölümleme**
  - **Veri Bağımlılığı**
  - **Test ve Hata Ayıklama**

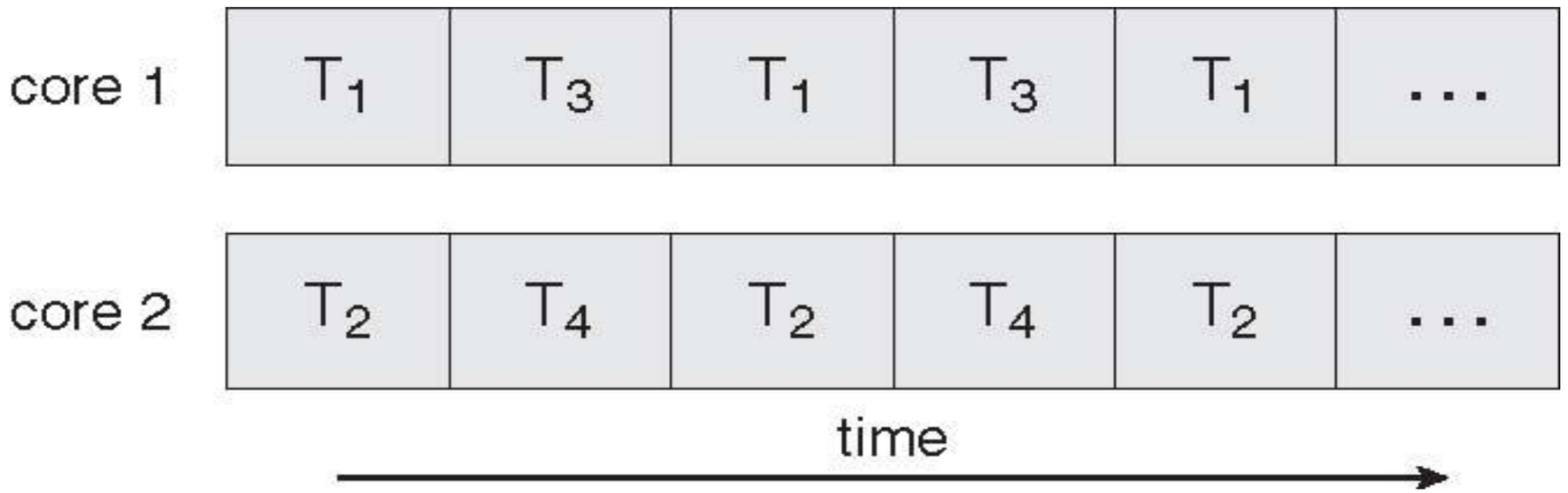
Çoklu iş parçacığı Sunucu Mimarisi



Tek Çekirdekli Sistemlerde Eşzamanlılık



Çok Çekirdekli Sistemlerde Paralel Çalışma



Kullanıcı Seviyeli İş Parçacıkları

- İş parçacığı yönetimi, kullanıcı seviyeli iş parçacığı kütüphaneleri ile yapılır.
- 3 adet temel iş parçacığı kütüphanesi vardır:
  - POSIX **Piş parçacığıs**
  - Win32 iş parçacıkları
  - Java iş parçacıkları

Çekirdek İş Parçacıkları

- Çekirdek tarafından çalıştırılır.
- Örnekler :
  - Windows XP/2000
  - Solaris
  - Linux
  - Tru64 UNIX
  - Mac OS X

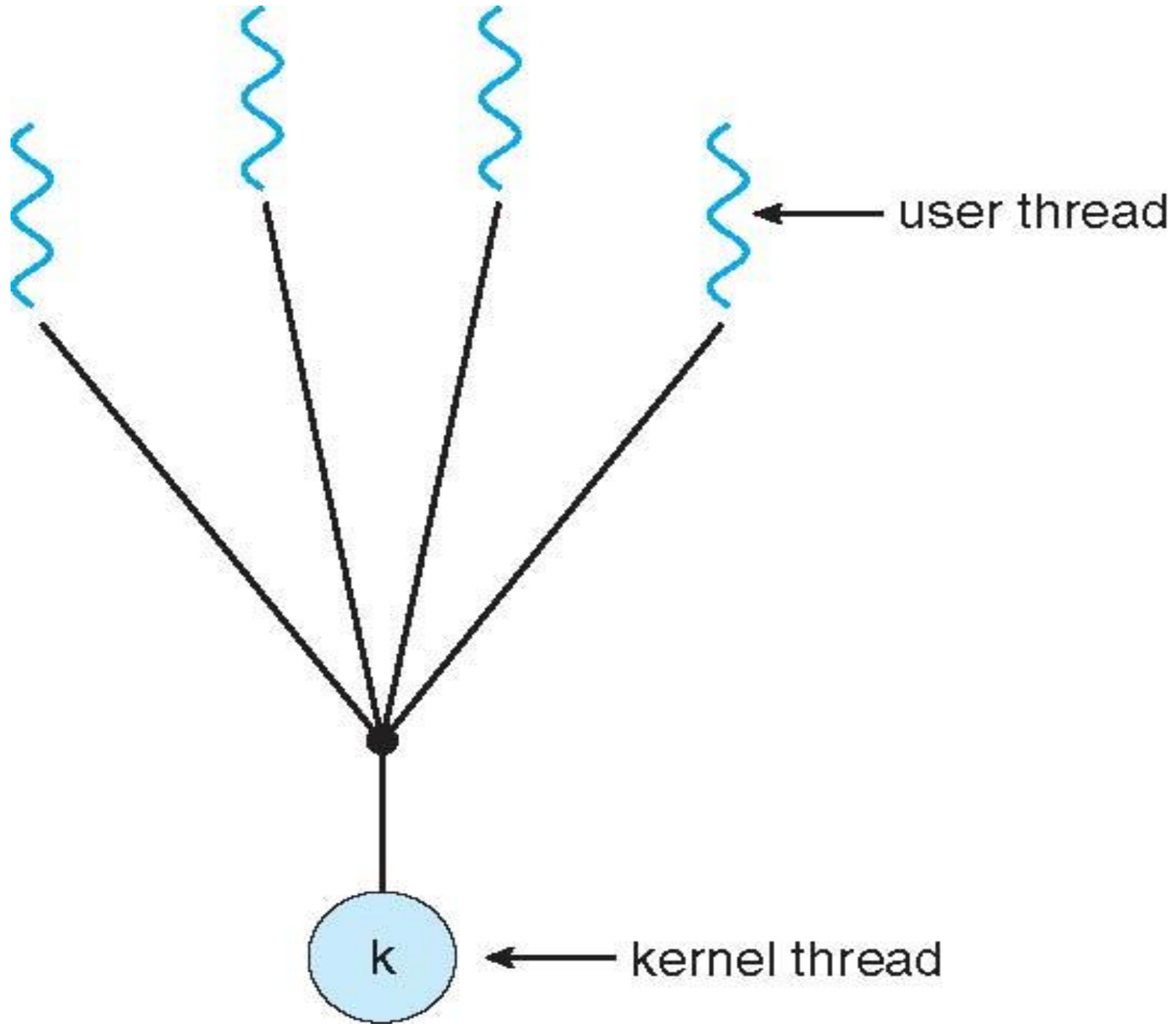
Çoklu İş Parçacığı Modelleri

- Çok-a-bir
- Bir-e-bir
- Çok-a-çok

Çok-a-tek

- Birden fazla kullanıcı-seviyeli iş parçacığı, tek bir çekirdek iş parçacığı ile ilişkilendirilir.
- Örnekler :
  - **Solaris Green iş parçacığı**
  - **GNU Portable iş parçacığı**

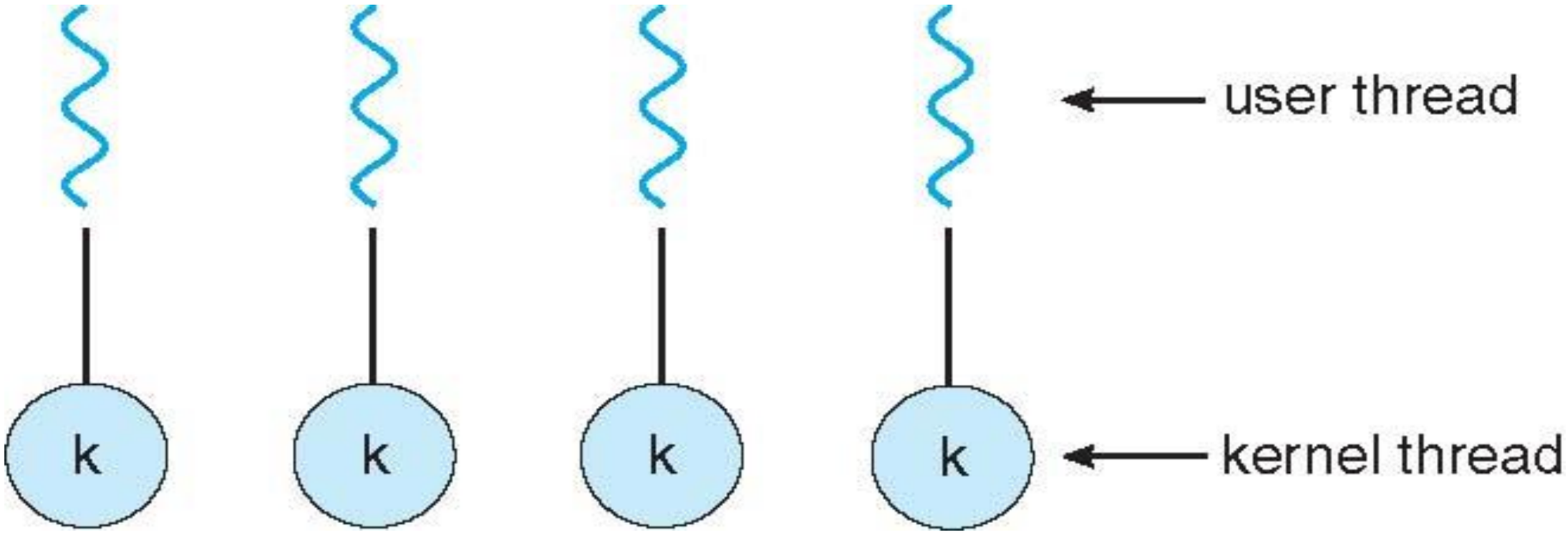
Çok-a-tek Modeli



Bir-e-Bir

- Her kullanıcı-seviyeli iş parçacığı, çekirdek iş parçacığı ile ilişkilendirilir.
- Örnekler:
  - Windows NT/XP/2000
  - Linux
  - Solaris 9 ve üstü

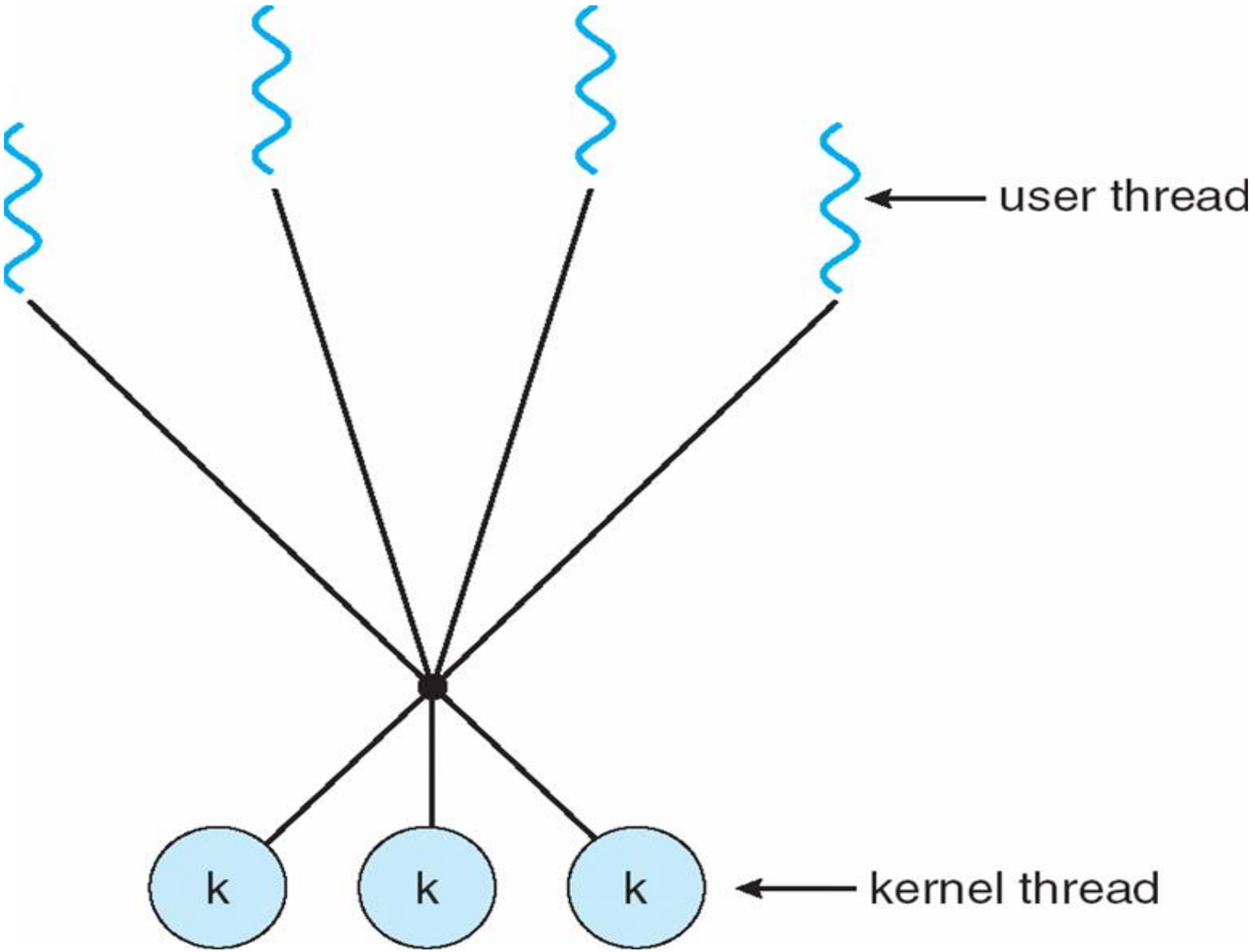
Bir-e-bir Modeli



Çok-a-Çok Modeli

- Çok sayıda kullanıcı-seviyeli iş parçacığının, yine çok sayıda çekirdek iş parçacığı ile ilişkilendirilmesine izin verir.
- İşletim sisteminin yeterli sayıda çekirdek iş parçacığı oluşturmasına izin verir.
- Solaris 9 ve önceki sürümlerinde
- *ThreadFiber* paketi ile Windows NT/2000

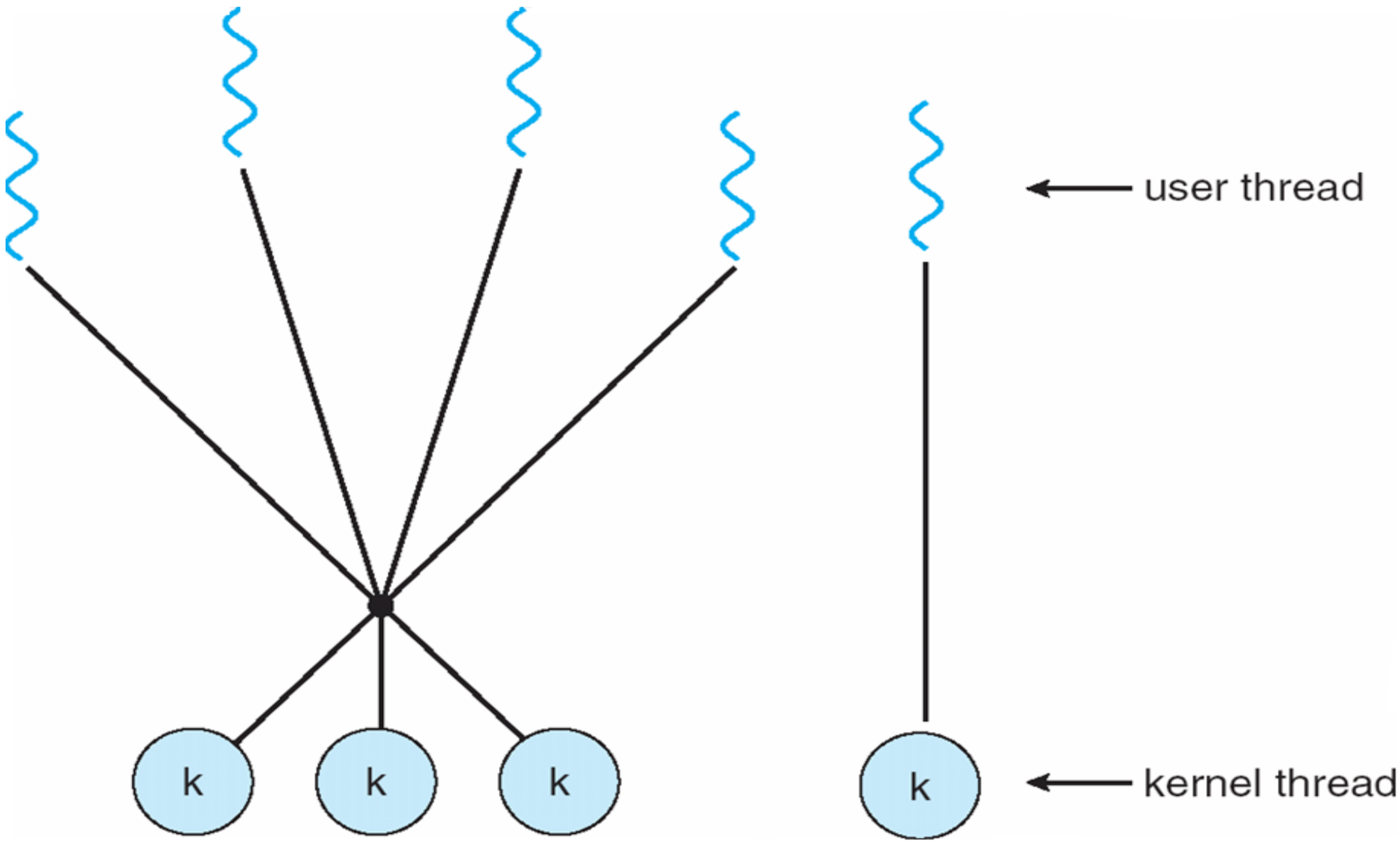
Çok-a-Çok Modeli



İki-Seviyeli Model

- Bir kullanıcı-iş parçacığının, bir çekirdek iş parçacığına bağlı olmasına izin vermesi dışında Çok-a-Çok modeli ile benzerdir.
- Örnekleri :
  - IRIX
  - HP-UX
  - Tru64 UNIX
  - Solaris 8 ve önceki sürümler

İki-Seviyeli Model



Iş Parçacığı Kütüphaneleri

- **Iş parçacığı kütüphanesi** programcılara API vasıtasıyla iş parçacıkları oluşturma ve bunları yönetme imkanı sağlar.
- İki temel uygulama yöntemi
  - Kütüphane tamamen kullanıcı tarafındadır.
  - Çekirdek taraflı kütüphane, işletim sistemi tarafından sağlanır.

Pthread İş parçacığı

- Kullanıcı-seviyesinde ya da çekirdek-seviyesinde olabilir.
- İş parçacığı yaratma ve senkronizasyon için bir POSIX standardı (IEEE 1003.1c) vardır.
- API iş parçacığı kütüphanesinin davranışını belirtir, uygulama kütüphanesinin gelişimine bağlıdır.
- UNIX işletim sistemlerinde (Solaris, Linux, Mac OS X) yaygındır.

Pthread Örneği

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

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

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;
   }
```

Pthread Örneği (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.

Win32 API Çoklu iş parçacığı C Programı

```
#include <windows.h>
#include <stdio.h>
DWORD Sum; /* data is shared by the thread(s) */
/* the thread runs in this separate function */

DWORD WINAPI Summation(LPVOID Param)
{
   DWORD Upper = *(DWORD*)Param;
   for (DWORD i = 0; i <= Upper; i++)
      Sum += i;
   return 0;
}

int main(int argc, char *argv[])
{
   DWORD ThreadId;
   HANDLE ThreadHandle;
   int Param;
   /* perform some basic error checking */
   if (argc != 2) {
      fprintf(stderr,"An integer parameter is required\n");
      return -1;
   }
   Param = atoi(argv[1]);
   if (Param < 0) {
      fprintf(stderr,"An integer >= 0 is required\n");
      return -1;
   }
```

Win32 API Çoklu İş parçacığı C Programı (Devam)

```
// create the thread
ThreadHandle = CreateThread(
   NULL, // default security attributes
   0, // default stack size
   Summation, // thread function
   &Param, // parameter to thread function
   0, // default creation flags
   &ThreadId); // returns the thread identifier

if (ThreadHandle != NULL) {
   // now wait for the thread to finish
   WaitForSingleObject(ThreadHandle,INFINITE);

   // close the thread handle
   CloseHandle(ThreadHandle);

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

Figure 4.10 Multithreaded C program using the Win32 API.

Java Iş parçacığı

- Java iş parçacıkları JVM tarafından yönetilir.
- Genelde, işletim sistemi tarafından sağlanan iş parçacığı modelleri temel alarak gerçekleştirilir.
- Java iş parçacıkları şunlar tarafından oluşturulabilir :
  - Türetilmiş iş parçacığı sınıfı
  - Runnable arayüzlerin uygulanması

Java Çoklu Iş parçacığı Programı

```
class Sum
{
   private int sum;

   public int getSum() {
     return sum;
   }

   public void setSum(int sum) {
     this.sum = sum;
   }
}

class Summation implements Runnable
{
   private int upper;
   private Sum sumValue;

   public Summation(int upper, Sum sumValue) {
     this.upper = upper;
     this.sumValue = sumValue;
   }

   public void run() {
     int sum = 0;
     for (int i = 0; i <= upper; i++)
        sum += i;
     sumValue.setSum(sum);
   }
}
```

Java Çoklu Iş parçacığı Programı (Devam)

```
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
         Sum sumObject = new Sum();
         int upper = Integer.parseInt(args[0]);
         Thread thrd = new Thread(new Summation(upper, sumObject));
         thrd.start();
         try {
            thrd.join();
            System.out.println
                    ("The sum of "+upper+" is "+sumObject.getSum());
          } catch (InterruptedException ie) { }
        }
      }
      else
        System.err.println("Usage: Summation <integer value>"); }
}
```

Figure 4.11 Java program for the summation of a non-negative integer.

İş parçacığı Sorunları

- **fork()** ve **exec()** sistem çağrılarının semantiği
- **Hedeflenen iş parçacıklarının sonlandırılması**
  - Asenkron ya da deferred (ertelenmiş)
- **Sinyal** işleme
  - Senkron ve Asenkron

İş Parçacığı Sorunları (Devam)

- İş parçacığı havuzları
- İş parçacığına özel veriler
  - iş parçacığı özel veri için ihtiyaç duyulan yapıyı oluştur
- İş Sıralayıcı Aktivasyonları

fork() ve exec()'in Semantiği

- fork() yalnızca çağırılan iş parçacığını mı kopyalar yoksa tüm iş parçacıklarını mı?

İş Parçacığı İptali

- İş parçacığı, bitmeden önce sonlandırılır.
- İki genel yaklaşım vardır:
  - **Asenkron** iptalde, hedef iş parçacığı hemen sonlandırılır.
  - **Ertelenmiş** iptal, periyodik olarak hedef iş parçacığının iptal edilmesinin gerekip gerekmediğinin kontrol edilmesi sağlanır.

Sinyal İşleme

- Sinyaller UNIX sistemlerde belirli bir olayın meydana geldiğini belirtmek için kullanılır.
- **Sinyal işleyiciler** proses sinyallerini kullanır
  1. Belirli bir olay tarafından sinyal oluşturulur.
  2. Sinyal prosese iletilir.
  3. Sinyal işlenir.
- Seçenekler:
  - Sinyali uygulayana kadar iş parçacığına sinyal gönder
  - Sinyali proses içindeki herbir iş parçacığına gönder
  - Sinyali proses içindeki belirli iş parçacığına gönder
  - Proses için tüm sinyalleri alan özel bir iş parçacığı belirle

İş Parçacığı Havuzları

- Bir havuz içerisinde, çalışmayı bekleyen iş parçacığı dizisi oluştur
- Avantajları:
  - Genellikle var olan bir iş parçacığına cevap vermek yeni bir iş parçacığı oluşturmaktan biraz daha hızlıdır.
  - Uygulama içindeki iş parçacıkları sayısının havuz boyutuyla sınırlandırılmasını sağlar

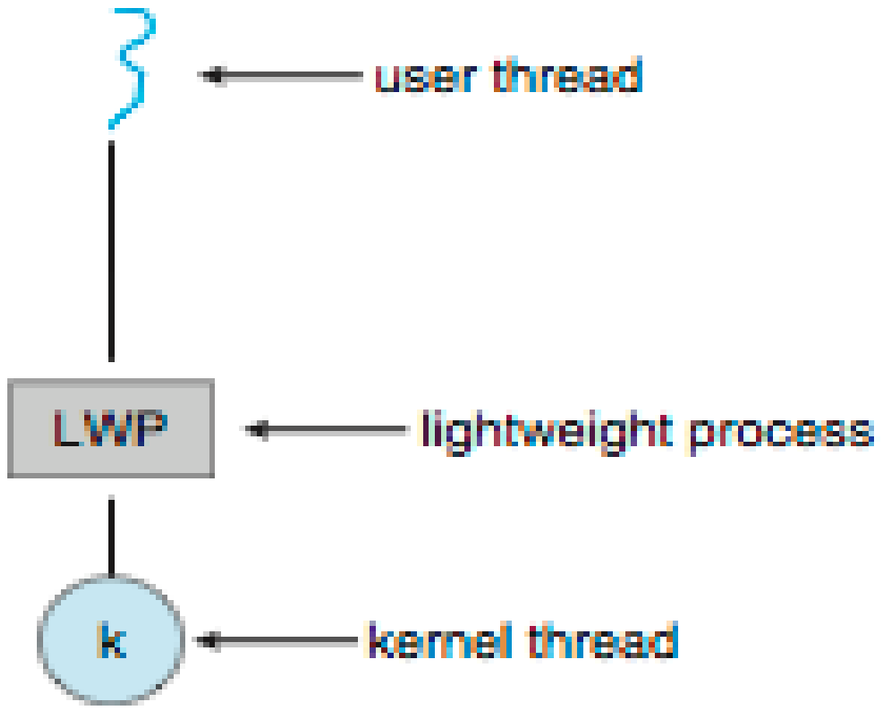
Iş Parçacığına Özel Veri

- Her iş parçacığı kendi verisinin kopyalanmasına izin verir.
- İş parçacığı oluşturma sürecinin kontrol edilmediği durumlarda yararlıdır. (örneğin, iş parçacığı havuzu kullandığınızda)

İş Sıralayıcı Aktivasyonları

- Hem Çok-a-Çok hemde İki-seviyeli modeller uygulamaya tahsis edilen çekirdek iş parçacıklarını yönetmek için iletişimi gerektirir
- İş sıralayıcı aktivasyonları, **upcalls'**ı destekler. – çekirdekten iş parçacığı kütüphanesine yönelik bir iletişim mekanizmasıdır.
- Bu iletişim bir uygulamanın yeter sayıda çekirdek iş parçacığını bulundurmasını sağlar

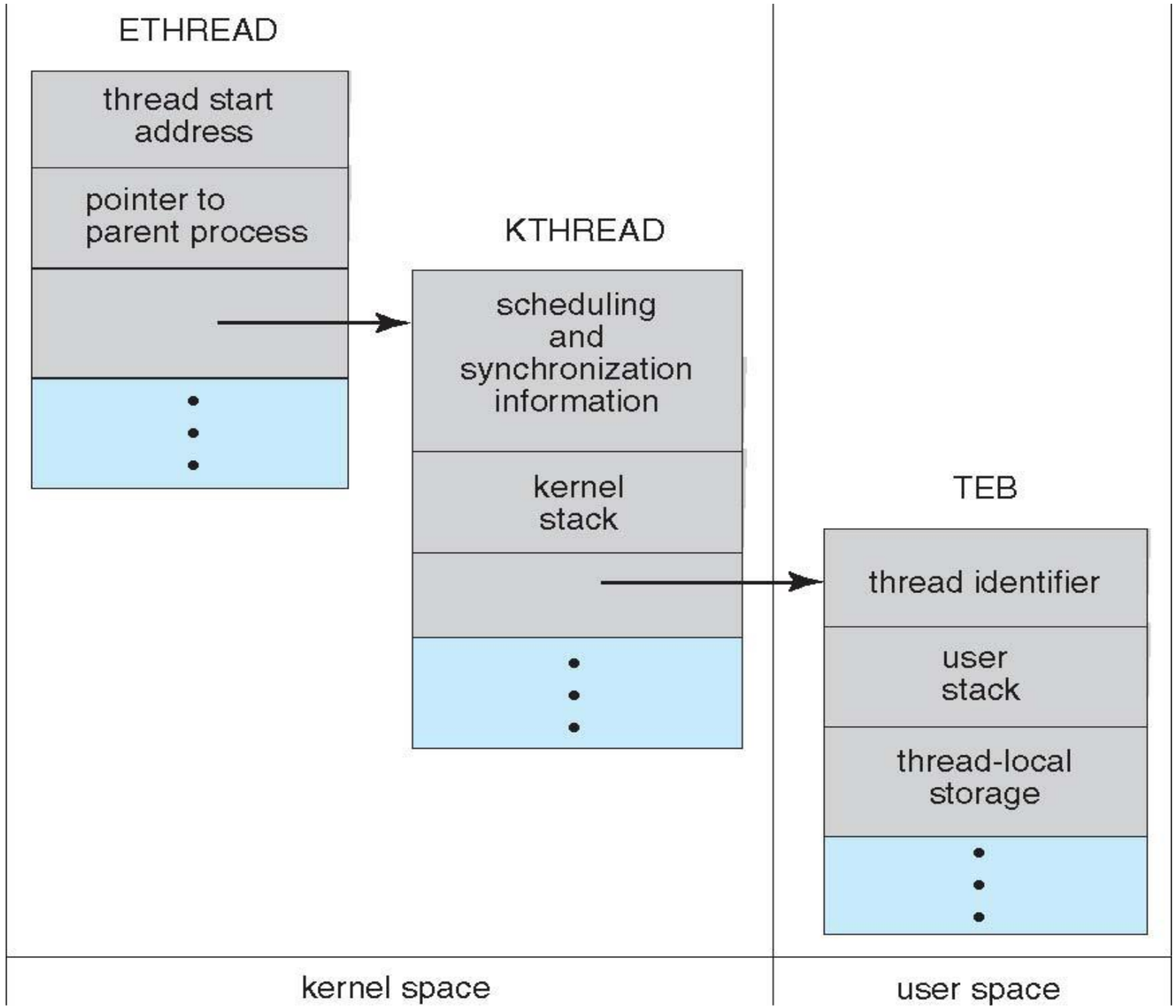
Hafif Prosesler



İşletim Sistemleri Örnekleri

- Windows XP iş parçacıkları
- Linux iş parçacıkları

Windows XP iş parçacığı Veri Yapıları



Windows XP iş parçacıkları

- Bir-e-bir model uygulanır.
- Her iş parçacığı şu özellikleri içerir :
  - Bir iş parçacığı id'si
  - Kaydedici seti
  - Ayrı kullanıcı ve çekirdek yığınları
  - Özel veri depolama alanları

- Kaydedici seti, yığınlar ve özel depolama alanı, iş parçacığının bağlamı olarak bilinir.

- Bir iş parçacığı, temel veri yapılarını içerir:
  - ETHREAD (yürütücü iş parçacığı bloğu)
  - KTHREAD (çekirdek iş parçacığı bloğu)
  - TEB (iş parçacığı ortamı bloğu)

Linux Iş Parçacığı

- Linux iş parçacığı yerine görevler terimini kullanır
- İş parçacığı oluşturulma işlemi clone() sistem çağrısı ile yapılır.
- `clone()` çocuk görevin (task), ebeveyn görevin adres alanını paylaşmasını sağlar.
- `struct task_struct` proses veri yapılarını gösterir (paylaşımlı veya tek)

Linux Iş Parçacığı

- `fork()` ve `clone()` sistem çağrıları
- Proses ve iş parçacığı arasında ayrım yapmaz.
  - iş parçacığından ziyade görev kullanır.
- `clone()` proses oluşturma üzerine paylaşımı belirlemek için seçeneklere sahiptir
- `struct task_struct` proses veri yapılarını gösterir (paylaşımlı veya tek)

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.

Bölüm 4 Sonu