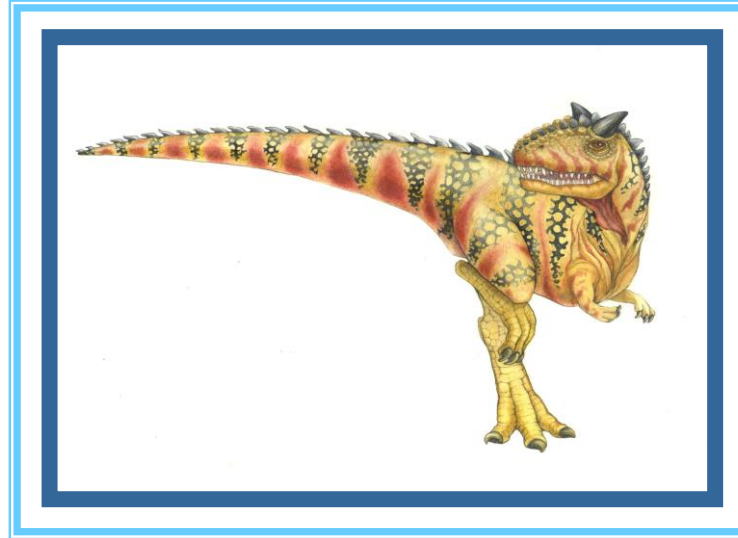


Bölüm 4: İş Parçacıkları (Threads)





Hafta 4: İş Parçacıkları

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





Hedefler

- İş parçacığı kavramını tanıtmak —CPU kullanımının en temel birimi (A thread is a basic unit of CPU utilization)
- Pthreads API'leri, Win32 ve Java iş parçacığı kütüphanelerini tanıtmak
- Çoklu iş parçacığı programlaması ile ilgili sorunları incelemek
- Örtülü iş parçacığını sağlayan stratejileri öğrenmek
- Windows ve Linux işletim sistemlerinde iş parçacığı desteğini kavramak





Motivasyon

- Çoğu modern uygulamalar çoklu iş parçacıklıdır
- İş parçacıkları uygulamanın içinde çalışırlar.
- Uygulama içinde birden fazla görev, ayrı iş parçacıkları tarafından gerçekleştirilebilir.
 - Güncellemeleri göstermek
 - Veri çekmek
 - Yazım denetimi, tuşa basılması, karakterin ekrana basılması
 - 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





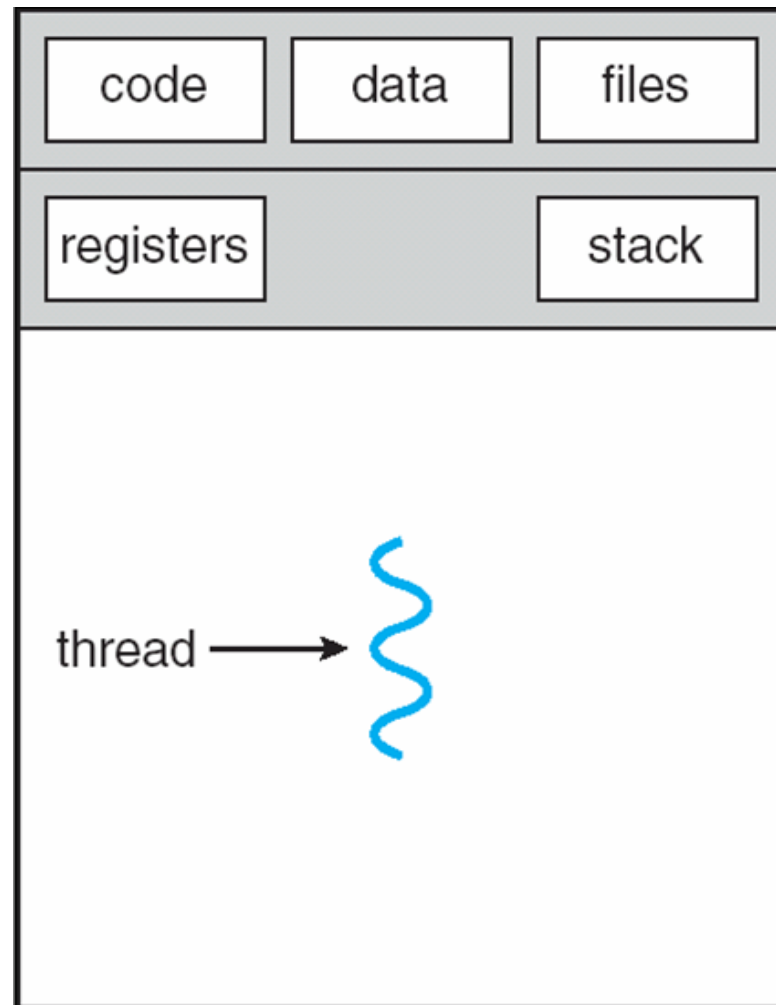
İş parçacığı ne demektir? (wikipedia)

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

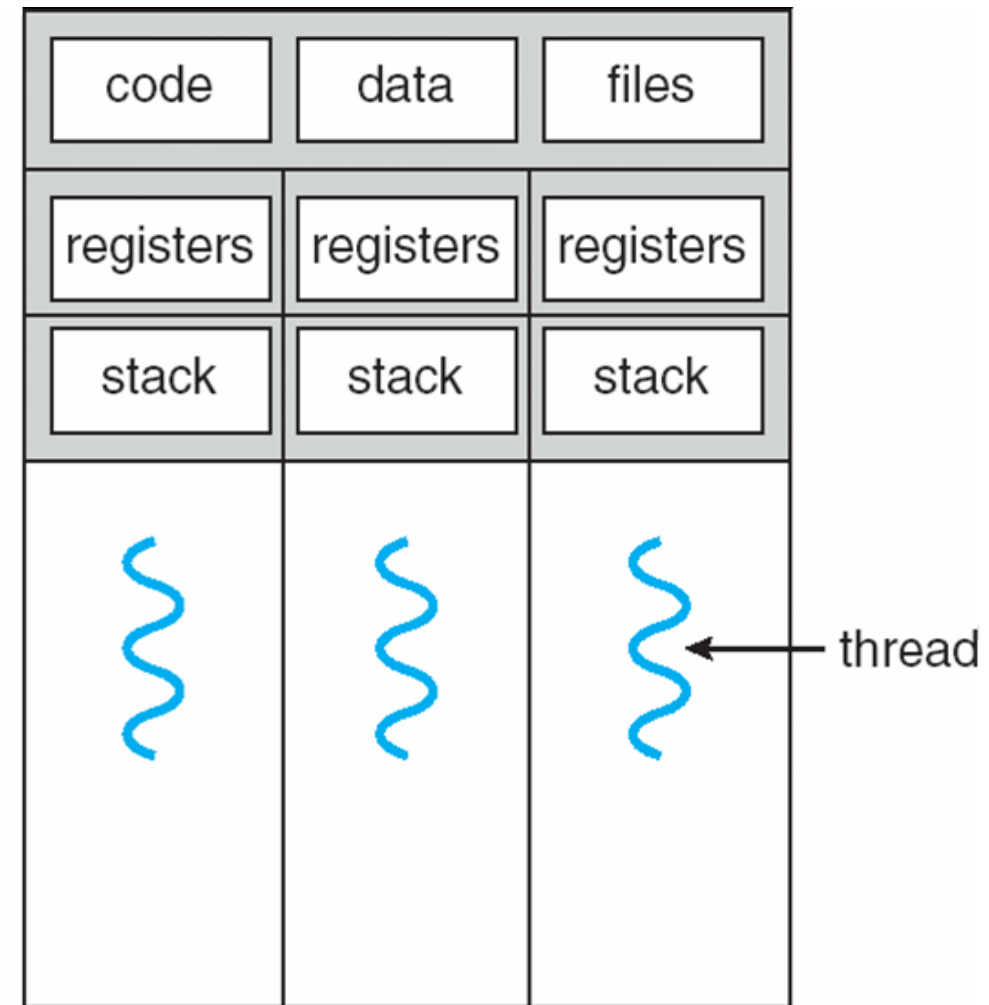




Tekli ve Çoklu iş parçacıklı Prosesler



single-threaded process



multithreaded process

- Bir iş parçacığının kendisine ait bir ID'si, program sayacı, a register kümesi, ve yığını(stack) vardır





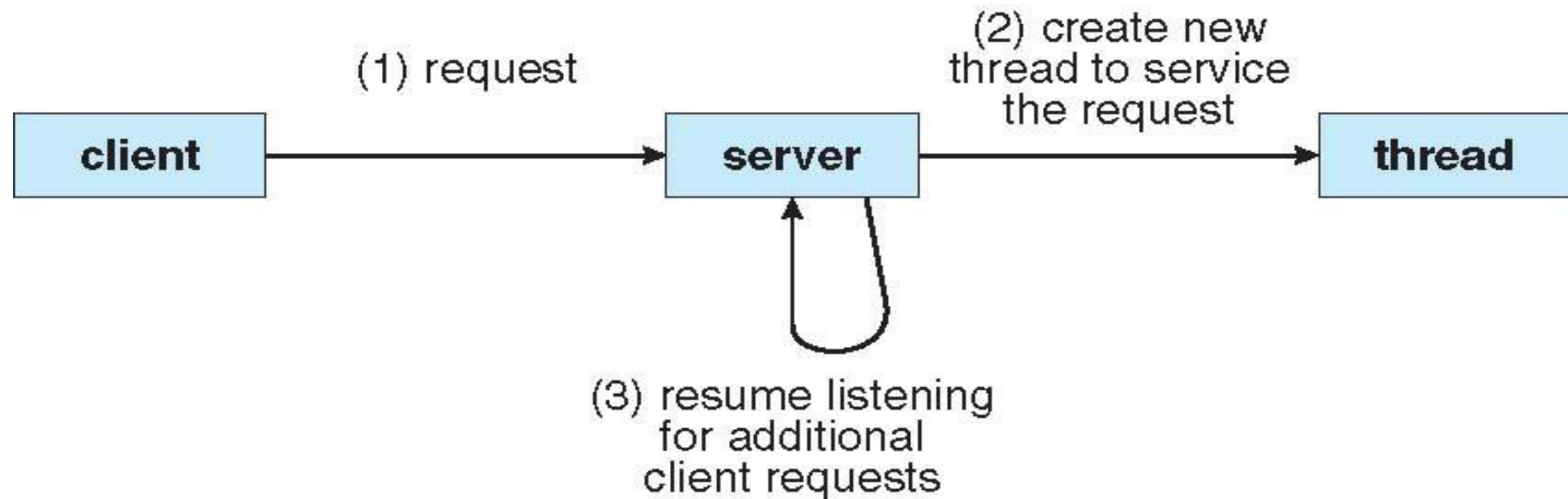
Faydaları

- **Duyarlılık/Hassasiyet:** Prosesin bir kısmı bloke olmuş olsa ise bile yürütmenin/programın devam etmesine izin verir. Özellikle arayüzlerde çok kullanışlı
- **Kaynak Paylaşımı:** İş-parçacıkları prosesin kaynaklarını kullanır. Paylaşılmış bellek veya mesajlaşmadan gibi haberleşmeye gerek yok zaten aynı belleği kullanırlar
- **Tasarruf:** prosesin oluşturulmasına göre daha kolay daha az masraflı. İş-parçacıklarının değiştirilmesi içerik değiştirmekten daha kolay (Solaris de proses oluşturma 30 kat daha yavaş)
- **Ölçeklenebilirlik:** Çoklu-işlemcili sistemlerin mimarilerinin avantajlarından faydalanabilir. Çoklu iş parçacıkları paralel olarak değişik çekirdeklerde çalışabilir





Çoklu iş parçacığı Sunucu Mimarisi





Çok Çekirdekli (Multicore) Programlama

- Çok çekirdekli sistemler, programcılarını verimlilik amacıyla aşağıdaki konulara zorlamaktadır:
 - Aktiviteleri bölme
 - Denge
 - Veri Bölümleme
 - Veri Bağımlılığı
 - Test ve Hata Ayıklama
- **Paralellik**, birden fazla görevi aynı anda yapmamıza olanak sağlar
- **Eş zamanlılık(Concurrency)**, birden fazla görevin yürütülmesini destekler
 - Tek çekirdekli/işlemcili sistemlerde çizelgeleyiciler eş zamanlılığı sağlar





Multicore Programming (Cont.)

■ Paralellik çeşitleri

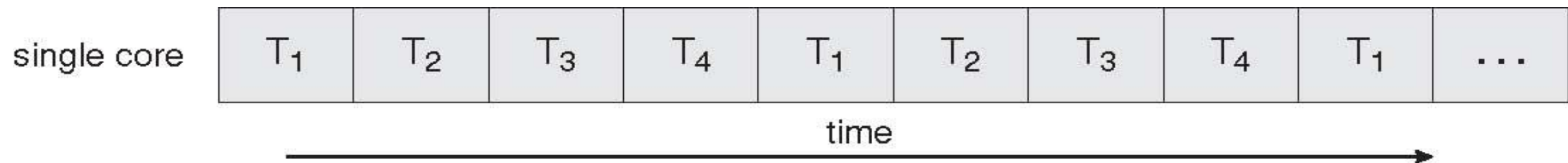
- **Veri Paralelliği**– Aynı verinin alt kümelerini birden çok çekirdeğe dağıtır, (her birinde aynı işlem)
- **Görev Paralelliği**– İş parçacıklarını çekirdeklere dağıtır, her iş parçacığı ayrı bir işlem gerçekleştirir.
- İş parçacıklarının sayısı arttıkça, iş parçacığı için mimari destek de artar
 - ▶ Donanım İş parçacığı sayısı kadar CPU'da çekirdek vardır
 - ▶ 8 çekirdekli Oracle SPARC T4 ve çekirdek başına 8 donanım iş parçacığı düşünün.



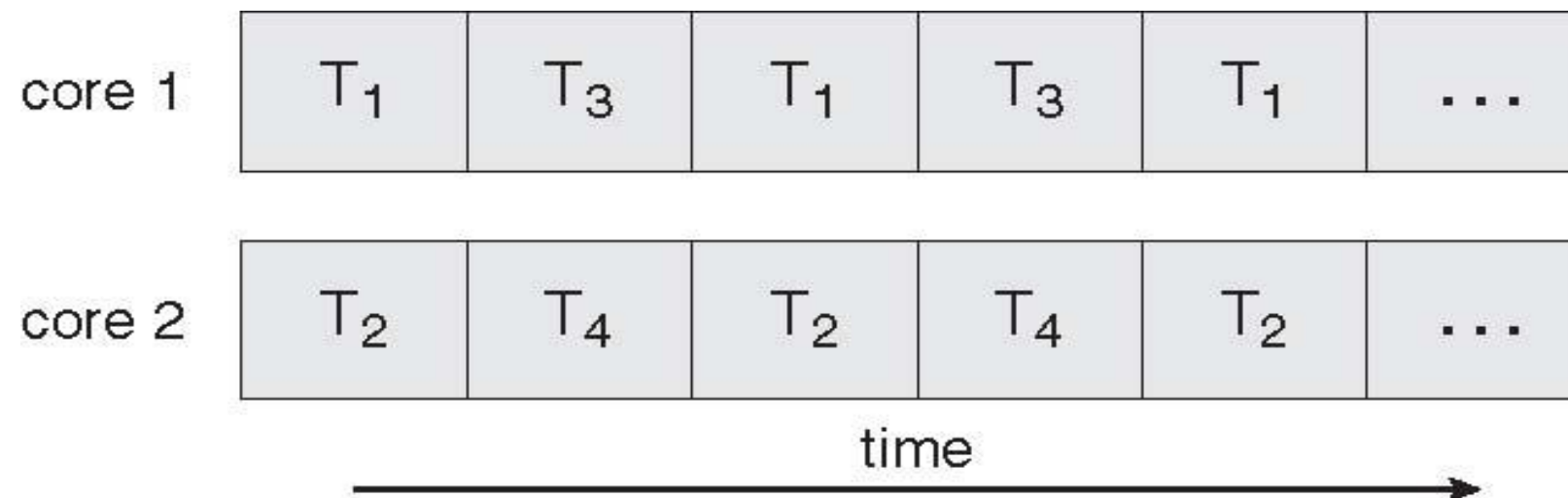


Tek Çekirdekli Sistemlerde Eşzamanlılık

- Tek-çekirdekli sistemlerde eş-zamanlı yürütme



- Çok-çekirdekli sistemlerde paralellik





Amdahl Kanunu

- Hem seri hem de paralel bileşenlere sahip bir uygulamaya ek çekirdek eklenmesinden elde edilen performans artışlarını belirler.
- S seri bölümdür
- N işlemci sayısı

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

- Yani, eğer uygulama% 75 paralel /% 25 seri ise, 1'den 2 çekirdeğe çıkma, 1,6 kat hızlanma sağlar
- N , sonsuzluğa yaklaşırken, hızlanma $1 / S$ 'ye yaklaşır





Kullanıcı İş Parçacıkları

- Kullanıcı İş parçacığı, kullanıcı seviyeli iş parçacığı kütüphaneleri ile yönetilir.
- 3 adet temel iş parçacığı kütüphanesi vardır:
 - POSIX **iş parçacığı (Pthreads)**
 - Win32 iş parçacıkları
 - Java iş parçacıkları





Çekirdek İş Parçacıkları

- Çekirdek tarafından çalıştırılır.
- Örnekler (Hemen hemen tüm genel amaçlı işletim sistemleri):
 - Windows XP/2000
 - Solaris
 - Linux
 - Tru64 UNIX
 - Mac OS X





Çoklu İş Parçacığı Modelleri

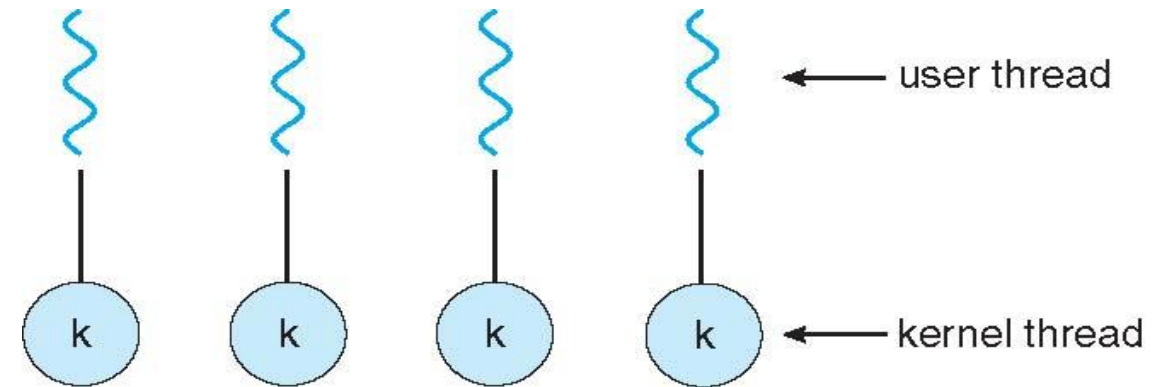
- Çok-a-bir
- Bir-e-bir
- Çok-a-çok





Bir-e-Bir

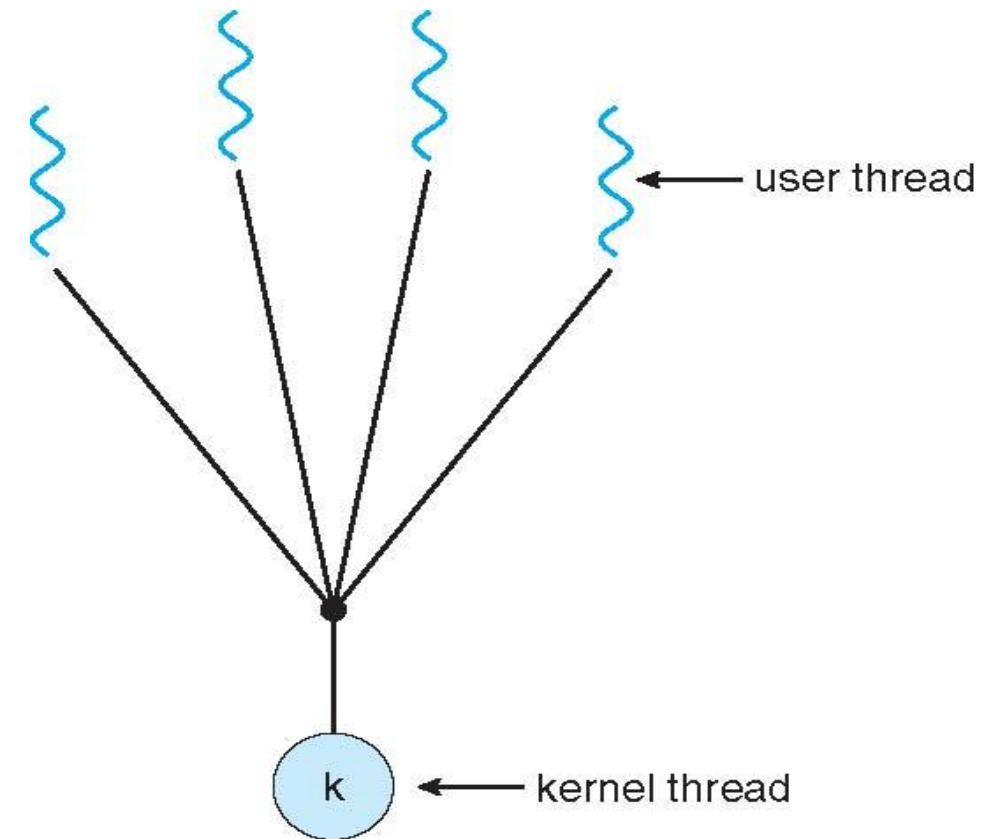
- Her kullanıcı-seviyeli iş parçacığı, bir çekirdek iş parçacığı ile ilişkilendirilir.
- Kullanıcı düzeyinde bir iş parçacığı oluşturmak, bir çekirdek iş parçacığı oluşturur
- Çok-a-tek'e göre daha fazla eşzamanlılık sağlar
- İşlem başına düşen iş parçacığı sayısı, bazen ek yük(overhead) nedeniyle kısıtlanmıştır
- Örnekler:
 - Windows NT/XP/2000
 - Linux
 - Solaris 9 ve üstü





Çok-a-bir

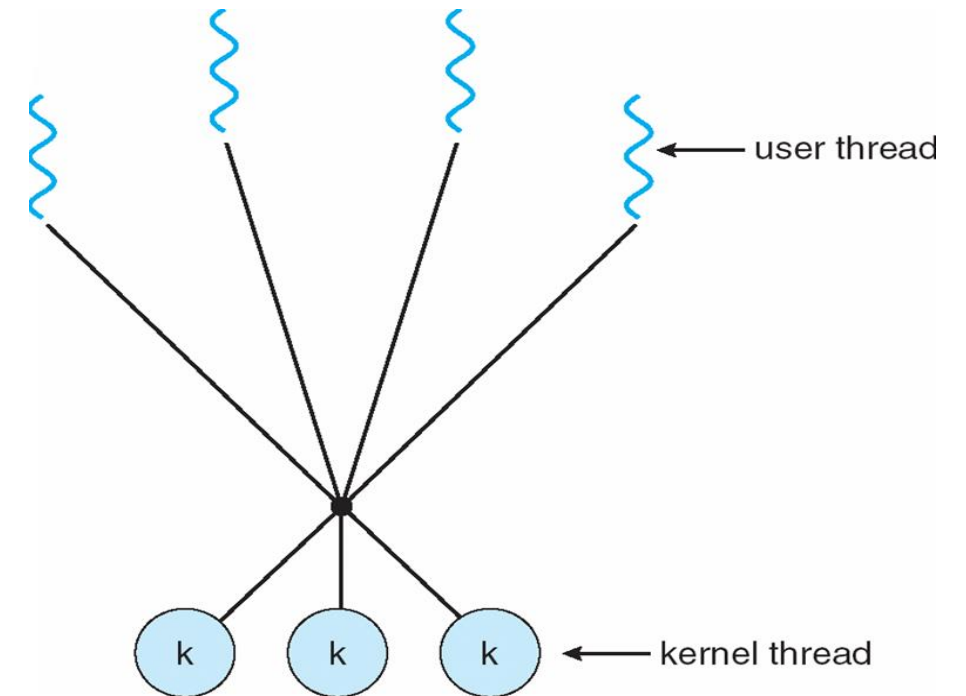
- Birden fazla kullanıcı-seviyeli iş parçacığı, tek bir çekirdek iş parçacığı ile ilişkilendirilir.
- Bir iş parçacığı engelleme herkesin engellenmesine neden olur
- Birden çok iş parçacığı, çok çekirdekli sistemde paralel çalışmayabilir, çünkü bir seferde sadece bir tane çekirdek işlemde olabilir.
- Az miktardaki sistemler bu modeli kullanır
- Örnekler :
 - **Solaris Green iş parçacığı**
 - **GNU Portable iş parçacığı**





Çok-a-Çok Modeli

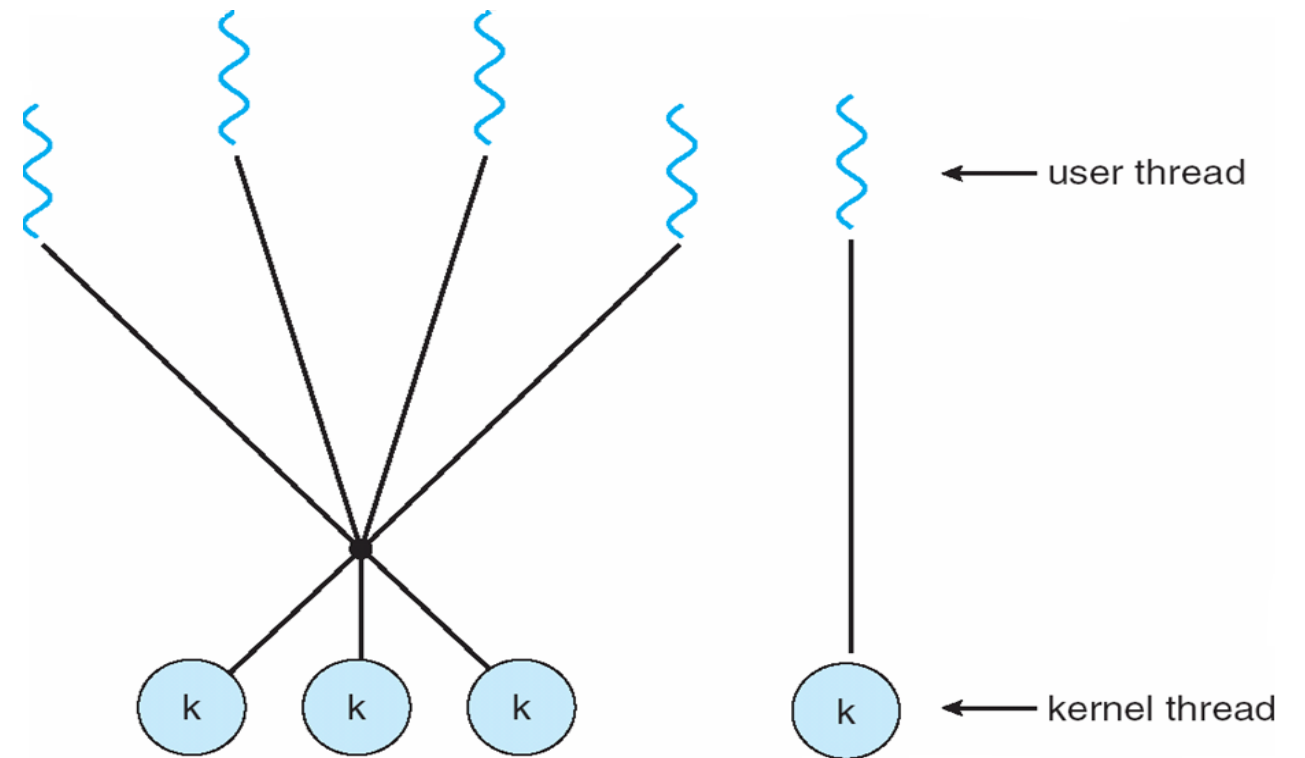
- Çok sayıda kullanıcı-seviyeli iş parçacığının, yine çok sayıda çekirdek iş parçacığı ile ilişkilendirilmesine izin verir.
- İşletim sisteminin yeterli sayıda çekirdek iş parçacığı oluşturmaya izin verir.
- Solaris 9 ve önceki sürümlerinde
- *ThreadFiber* paketi ile Windows NT/2000





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





İş Parçacığı Kütüphaneleri

- **İş parçacığı kütüphanesi** programcılara API vasıtasıyla iş parçacıkları oluşturma ve bunları yönetme imkanı sağlar.
- İki temel uygulama yöntemi
 - Kütüphane tamamen kullanıcı tarafındadır(Çekirdek desteği olmadan). Kütüphane için **kodlar ve veri yapıları kullanıcı tarafındadır.** Kütüphaneden bir fonksiyonun çağırılması yerel fonk. çağırma gibi olur sistem çağırısı değil.
 - Çekirdek-seviyesinde kütüphane, işletim sistemi tarafından sağlanır. Kütüphane için **kodlar ve veri yapıları çekirdek tarafındadır.** Kütüphaneden bir fonksiyonun çağırılması sistem çağırısı çağırma gibi olur.
- İki türlü iş parç. oluşturma tekniği var; Asenkron şekilde, ebeveyn çocuk iş parçacığını oluşturduktan sonra çalışmaya devam eder (Eş zamanlı çalışırlar). Senkron şekilde çocuk iş parç. İşleminin bitirilmesi beklenir





Java İş parçacığı

- Java iş parçacıkları JVM tarafından yönetilir.
- Genelde, işletim sistemi tarafından sağlanan iş parçacığı modelleri kullanılarak gerçekleştirilir.
- Java iş parçacıkları şunlar tarafından oluşturulabilir :
 - iş parçacığı sınıfından türetilerek, ve run() metodu override edilerek kullanılır
 - Runnable arayüzünün (Interface) uygulanması ile

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





Java Çoklu İş parçacığı Programı

```
class Sum
{
    private int sum;

    public int getSum() {
        return sum;
    }

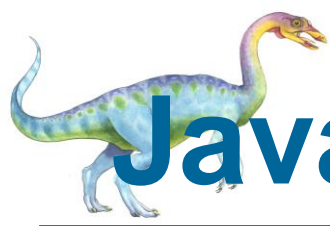
    public void setSum(int sum) {
        this.sum = sum;
    }
}

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 İş 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 {
                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>"); }
}
```





Pthread İş parçacığı

- Kullanıcı-seviyesinde ya da çekirdek-seviyesinde olabilir.
- İş parçacığı oluşturma ve senkronizasyon için bir POSIX standardı (IEEE 1003.1c) vardır.
- Tanımlama, uygulama değil
- API, iş parçacığı kütüphanesinin davranışını belirtir. Uygulama kütüphanenin oluşumuna/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); /* 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;
    }
}
```





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);
}
```





Pthreads Code for Joining 10 Threads

```
#define NUM_THREADS 10

/* an array of threads to be joined upon */
pthread_t workers[NUM_THREADS];

for (int i = 0; i < NUM_THREADS; i++)
    pthread_join(workers[i], NULL);
```





Win32 API Çoklu iş parçacığı

- iş parçacığı kütüphanesi Çekirdek tarafındadır
- Birçok açıdan Pthreads tekniğine benzer yapıda
- windows.h header dosyasını içermesi zorunlu
- Herhangi bir veri global olarak deklare edilir. Prosesdeki bütün fonk. ulaşabilir





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;

    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);
}
}
```





Implicit Threading

- İş parçacıkları sayısı arttıkça, program doğruluğu daha da zorlaştı.
- İş parçacıklarının oluşturulması ve yönetimi Derleyiciler ve run-time kütüphaneleri tarafından yapılmaya başlandı
- Genel 3 metot var;
 - Thread havuzları
 - OpenMP
 - Grand Central Dispatch (Gönderim merkezleri)
- Diğerleri, Microsoft Threading Building Blocks (TBB) ve `java.util.concurrent` package





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
 - Görev oluşturma mekanizmasında gerçekleştirilecek görevin ayrı olması, görevin yürütülmesi için farklı stratejilere izin verir.
- Windows API İş parç. Havuzlarına izin verir

```
DWORD WINAPI PoolFunction(AVOID Param) {  
    /*  
    * this function runs as a separate thread.  
    */  
}
```





OpenMP

- C, C ++, FORTRAN için bir API ve Derleyici yönergesi kümesi
- Paylaşılan bellek ortamlarında paralel programlama için destek sağlar
- Paralel bölgeleri tanımlar - paralel çalışabilen kod blokları

```
#pragma omp parallel
```

- Çekirdek kadar çok İş parç. Oluşturun

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

- For döngüsünü paralel olarak çalıştır

```
#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;  
}
```





Grand Central Dispatch

- Mac OS X ve iOS İşletim Sist. İçin geliştirilmiş Apple teknolojisi
- C, C ++ dilleri uzantıları, API ve run-time kütüphaneleri
- Paralel bölümlerin tanımlanmasına izin verir
- İş parçacıklarının detaylarını yönetir
- Blok, basitçe kendi başına çalışan bir iş birimi.
- Blok “`^ { }`” içinde tanımlanır `^ { printf("I am a block"); }`
- Bloklar Gönderim kuyruğuna yerleştirilir
 - Kuyruktan kaldırıldığında, Bloku, yönettiği iş parçacığı havuzundan mevcut bir iş parçacığına atar.





Grand Central Dispatch

- İki tür gönderme kuyruğu:
 - seri - FIFO düzenine göre kaldırılan bloklar, her bir prosesin kendine ait kuyruğu vardır, **ana kuyruk** olarak adlandırılır
 - ▶ Programcılar, program içinde ek seri kuyruklar oluşturabilir
 - Eş zamanlı - FIFO sırasına göre kaldırılır, ancak aynı anda birkaç tane çıkartılabilir
 - ▶ düşük, varsayılan, yüksek olarak üç sistem Öncelikli kuyruk modeli var

```
dispatch_queue_t queue = dispatch_get_global_queue  
    (DISPATCH_QUEUE_PRIORITY_DEFAULT, 0);  
  
dispatch_async(queue, ^{ printf("I am a block."); });
```





İş parçacığı Sorunları

- **fork()** ve **exec()** sistem çağrılarının anlamı çoklu iş parçacıklı yapıda değişir.
- **Sinyal** işleme
 - Senkron ve Asenkron
- **Hedeflenen iş parçacıklarının sonlandırılması**
 - Asenkron ya da deferred (gecikmeli)
- **İş parçacığı- Yerel depolama**
- **Çizelgeleyici aktivasyonları**





fork() ve exec()'in Semantiği

- fork() yalnızca çağırılan iş parçacığını mı kopyalar yoksa tüm iş parçacıklarını mı?
- Proseslerde fork() sistem çağrısının yeni proses oluşturduğundan bahsetmiştik. Unix'de iki türlü fork() çağrısı vardır;
 - Fork() çağrısında bulunan iş parçacığı kopyalanır
 - Fork() çağrısında bulunan iş parçacığının prosesindeki tüm iş parçacıkları kopyalanır
- exec() sistem çağrısı Aynı şekilde çalışır ve tüm proses (Bütün iş parçacıkları dahil) yer değiştirir





Sinyal İşleme

- Sinyaller UNIX sistemlerde belirli bir olayın meydana geldiğini belirtmek için kullanılır.
- **Sinyal işleyiciler** sinyalleri işlemek için kullanılır
 1. Belirli bir olay tarafından sinyal oluşturulur.
 2. Sinyal prosese iletilir.
 3. 2 Sinyal işleyicisi tarafından sinyaller işlenir.
 1. Varsayılan
 2. Kullanıcı tanımlı
- Her sinyalin varsayılan bir işleyicisi vardır ve sinyal işleneceği zaman çekirdek tarafından yürütülür
 - Kullanıcı tanımlı işleyiciler varsayılan değerin üzerine yazılabilirler
 - Tek iş parçacıklıda sinyal prosese iletilir. !





Sinyal İşleme

- Çoklu iş parçacıklı da seçenekler:
 - Sinyali uygulayan iş parçacığına sinyali gönder
 - Sinyali proses içindeki herbir iş parçacığına gönder
 - Sinyali proses içindeki belirli iş parçacıklarına gönder
 - Proses içinde tüm sinyalleri alan özel bir iş parçacığı belirle
- Senkron gönderimde sinyal sadece olay olmasına sebep olan iş parçacığına gönderilir
- Asenkron gönderim ise tam kesin değil mesela bir olay olduğunda (Mesela Ctrl+c'ye basıldı) bu olay tüm iş parçacıklarına gönderilebilir.





Sinyal İşleme

- Sinyal ileten UNIX fonksiyonu:

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

- POSIX Pthread'lerinde istenilen iş parçacığına sinyal gönderilebilmesi aşağıdaki komutla sağlanmaktadır;

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

- Windows açıktan sinyalleri desteklemez. Asenkron prosedür çağırısı kullanarak sinyallerin benzeri işleri görmektedir
 - APC =~ UNIX'de asenkron sinyal

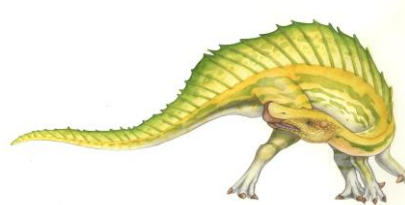




İş Parçacığı İptali

- İş parçacığının işlemi bitmeden önce sonlandırılması.
- İptal edilecek iş parçacığı **hedef iş parçacığı** olarak adlandırılır.
- İki genel yaklaşım vardır:
 - **Asenkron** iptalde, hedef iş parçacığı hemen sonlandırılır.
 - **Gecikmeli** iptal, periyodik olarak hedef iş parçacığının iptal edilmesinin gerekip gerekmediğinin kontrol edilmesi sağlanır. Aslında iş parçacığına kendisini sonlandırma fırsatı sunar.
- Pthreads de iş parçacığı oluşturma ve iptal etme;

```
pthread_t tid;  
  
/* create the thread */  
pthread_create(&tid, 0, worker, NULL);  
  
. . .  
  
/* cancel the thread */  
pthread_cancel(tid);
```





İş Parçacığı İptali

- İş parçacığının iptal edilmesi isteği geldiğinde sonlandırma işlemi iş parçacığının durumuna göre gerçekleşir;

Mode	State	Type
Off	Disabled	–
Deferred	Enabled	Deferred
Asynchronous	Enabled	Asynchronous

- İş parçacığı iptal edilmeyi geçersiz duruma getirmiş ise iptal işlemi için geçerli durumuna gelene kadar beklenir
- Varsayılan tip gecikmeli dir.
 - İptal edilme işlemi sadece iş parçacığının **iptal edilme noktasına** geldiğinde gerçekleşir.
 - ▶ `pthread_testcancel()`
 - ▶ Sonra `cleanup handler` çağrılır





İş Parçacığına Özel Veri

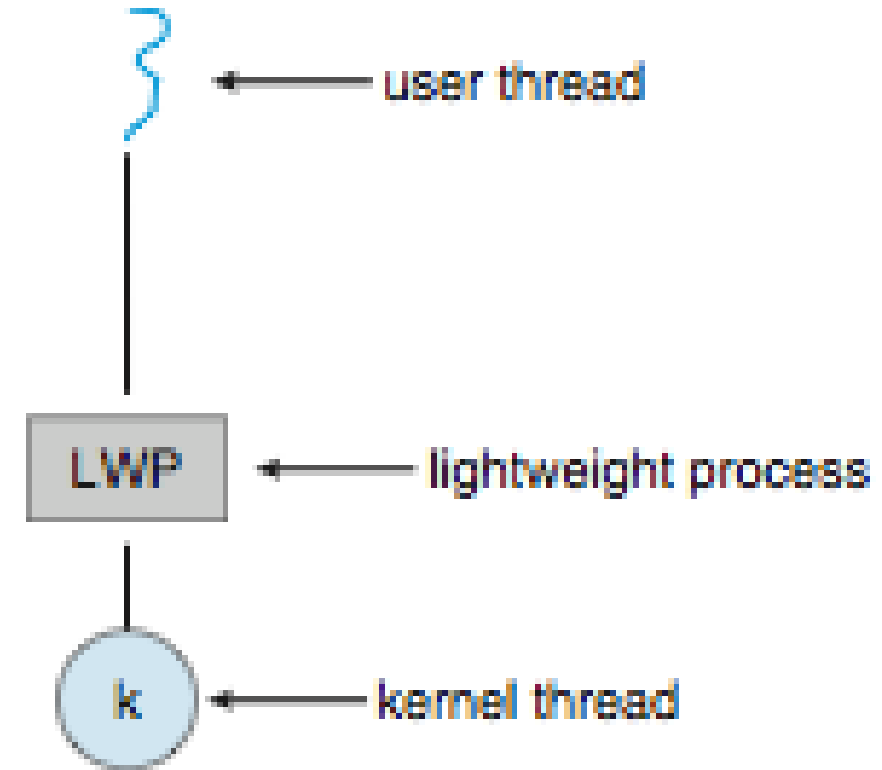
- **Thread-local storage (TLS)** olarak adlandırılır
- Her iş parçacığı kendi verisinin kopyalanmasına izin verir.
- İş parçacığı oluşturma sürecinin kontrol edilemediği durumlarda yararlıdır. (örneğin, iş parçacığı havuzu kullandığınızda)
- Yerel değişkenlerden farklıdır
 - Yerel değişkenler sadece fonksiyon çağırıldığı süre boyunca görünür olur
 - TLS ise fonksiyonlar çağırımları boyunca görünür olur
- Statik veriye benzerdirler
 - TLS her bir iş parçacığı için tekildir





İş Çizelgeleyici Aktivasyonları

- Çekirdek ile iş parçacığı kütüphanesi arasında iletişime ihtiyaç duyulur (Hem Çok-a-Çok hemde İki-seviyeli modellerde). Bu sayede uygulamaya tahsis edilen çekirdek iş parçacıklarını sayısını ayarlanır
- Kullanıcı iş parç. ile çekirdek iş parç. Ortasında bir veri yapısı kullanılır. Buna hafif prosesler denir.
 - Hangi prosesin kullanıcı iş parçacığını yürütmesine karar verir
 - Herbir LWP çekirdek iş parçacığı ile ilişkilendirilmiştir
 - Ne kadar LWP üretilecek?
 - ▶ Tek işlemci de CPU-bağımlı uygulama varsa 1 LWP gerekli. Eğer I/O bağımlı uyg için birden fazla LWP olabilir.





İş Çizelgeleyici Aktivasyonları

- İş çizelgeleyici aktivasyonları, **upcalls**'ı destekler. – çekirdekten iş parçacığı kütüphanesine yönelik bir iletişim mekanizmasıdır.
- Bu iletişim, bir uygulamanın yeter sayıda çekirdek iş parçacığını sürdürebilmesini sağlar





İşletim Sistemleri Örnekleri

- Windows XP iş parçacıkları
- Linux iş parçacıkları





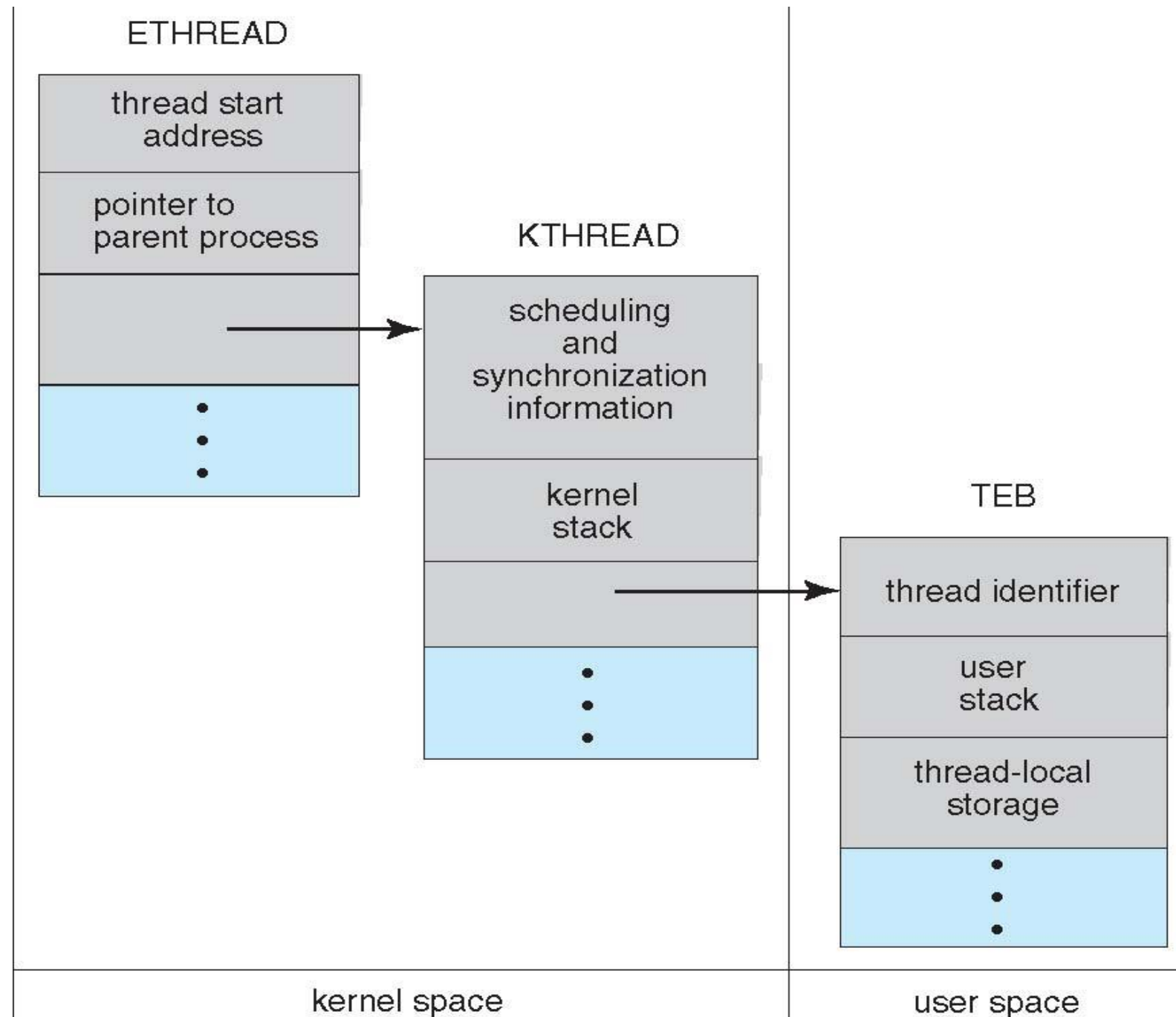
Windows XP iş parçacıkları

- 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 **İçeriği** olarak bilinir.
- Bir iş parçacığı, temel veri yapılarını içerir:
 - ETHREAD (yürütücü iş parçacığı bloğu)
 - KTHREAD (çekirdek iş parçacığı bloğu)
 - TEB (iş parçacığı ortamı bloğu)





Windows XP iş parçacığı Veri Yapıları





Linux İş 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.





Linux İş Parçacığı

- `fork()` ve `clone()` sistem çağrıları
- Proses ve iş parçacığı arasında ayırım yapmaz.
- `clone()` proses oluşturma üzerine paylaşımı belirlemek için seçeneklere sahiptir
- `struct task_struct` proses veri yapılarını gösterir (paylaşımlı veya tek)
 - Kontrol bayrakları

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



Bölüm 4 Sonu

