Threadler

Dr. Öğr. Üyesi Ertan Bütün

Bu dersin içeriği hazırlanırken Operating System Concepts (Silberschatz, Galvin and Gagne) kitabı ve
Prof. Dr. M. Ali Akcayol'un (Gazi Üniversitesi Bilgisayar Mühendisliği Bölümü) ders sunumlarından faydalanılmıştır.

https://codex.cs.yale.edu/avi/os-book/OS10/index.html

http://w3.gazi.edu.tr/~akcayol/BMOS.htm

Konular

- Threadler
- Multithread programlamanın avantajları
- Multicore programlama
  - Multicore programlamanın zorlukları
  - Paralel çalışma türleri
- Multithreading modelleri
  - Many-to-one
  - One-to-one
  - Many-to-many
- Thread kütüphaneleri
- Dolaylı thread oluşturma
- Threadlerin yürütülmesi ile ilgili bazı hususlar
- Windows ve Linux thread'leri

Amaçlar

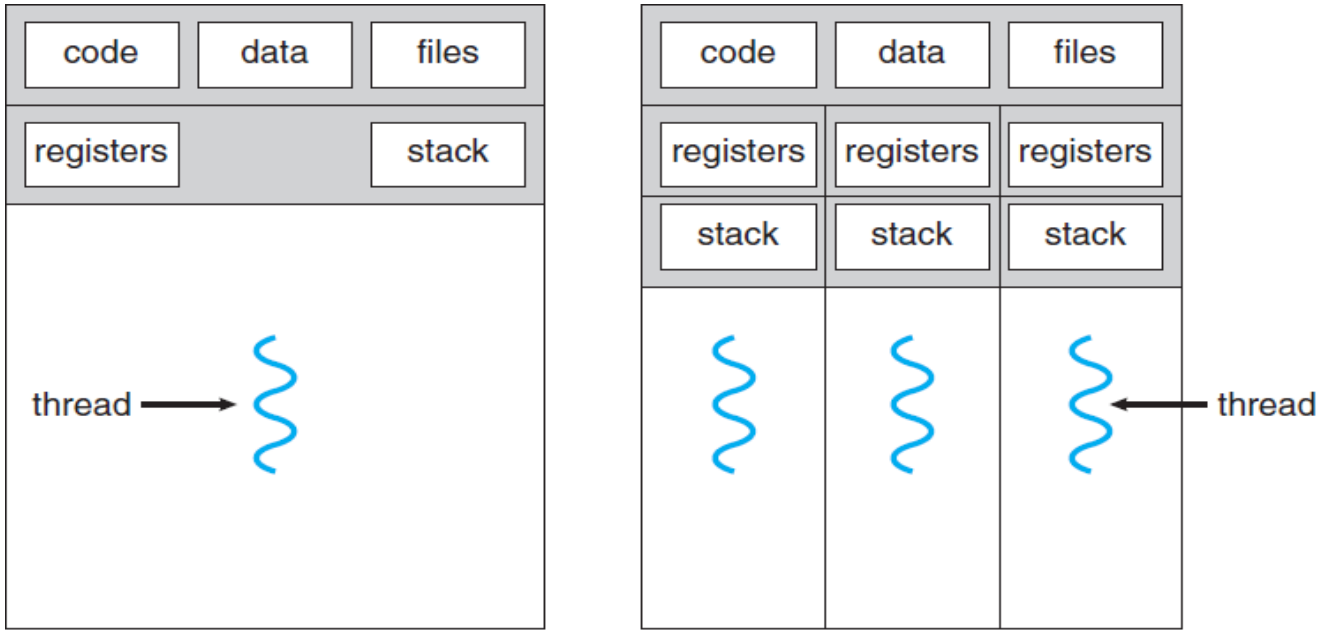
- Multithread bilgisayar sistemlerinin temelini oluşturan ve CPU kullanımının temel birimi olan thread kavramına giriş yapmak
- Pthreads, Windows ve Java thread kütüphaneleri için API'leri incelemek
- Dolaylı thread stratejilerini keşfetmek
- Multithread programlama ile ilgili sorunları incelemek
- Windows ve Linux'ta threadler için işletim sistemi desteğini incelemek

Thread'ler

- Günümüzdeki modern bilgisayarlarda ve mobil cihazlarda çalışan yazılım uygulamalarının çoğu multithread çalışırlar.
- Uygulamalar, çok sayıda thread'e sahip tek process şeklinde geliştirilirler.
- Bir process, birden fazla thread'e sahipse birden fazla görevi eşzamanlı yapabilir.
- Multithread uygulama örnekleri:
  - Bir web tarayıcısında bir thread resim veya metin görüntülerken, başka bir thread ağdan veri alır.
  - Bir word uygulamasında, bir thread klavyeden giriş alabilir, bir thread yazım denetleme yapabilir ve başka bir thread ekran görüntüsünü düzenleyebilir.

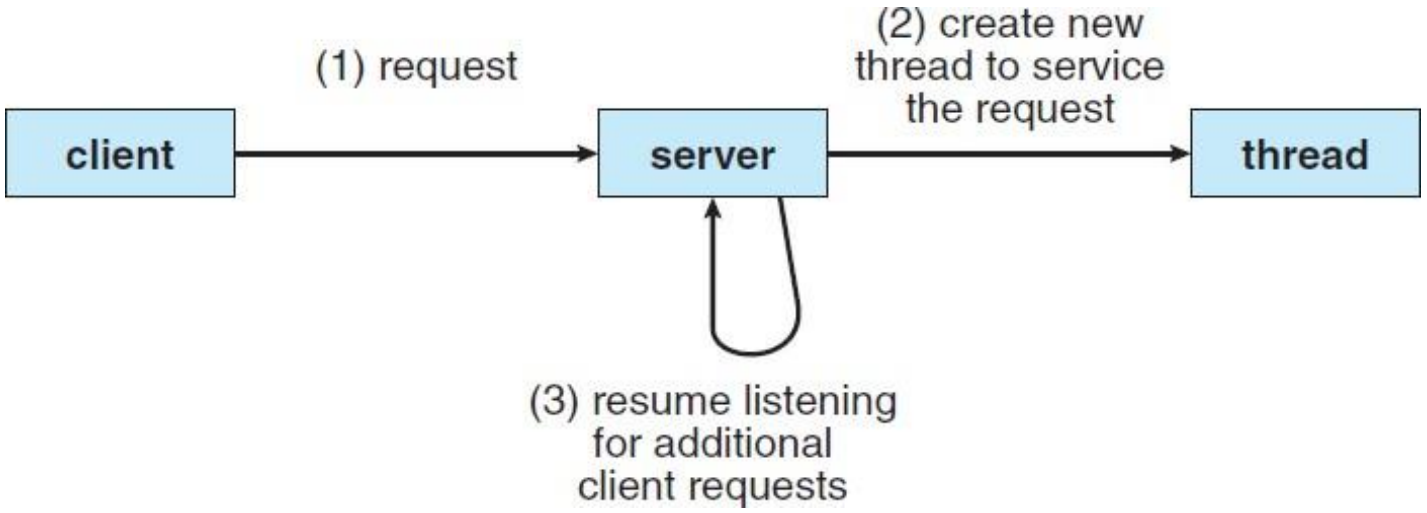
Thread'ler

- **Bir thread; thread ID, program counter, bir grup register ve bir stack yapısından oluşur.**
- **Aynı processe ait threadler, processin program kodunu, data kısmını ve processin kullandığı dosyalar ve başka diğer işletim sistemi kaynaklarını ortak kullanırlar.**



single-threaded process multithreaded process

Multithreaded Server Architecture



Konular

- Thread'ler
- Multithread programlamanın avantajları
- Multicore programlama
  - Multicore programlamanın zorlukları
  - Paralel çalışma türleri
- Multithreading modelleri
  - Many-to-one
  - One-to-one
  - Many-to-many
- Thread kütüphaneleri
- Dolaylı thread oluşturma
- Threadlerin yürütülmesi ile ilgili bazı hususlar
- Windows ve Linux thread'leri

Multithread programlamanın avantajları

- Multithread programlamanın avantajları 4 kategori halinde ifade edilebilir:

1. **Responsiveness (cevap verebilirlik)**: Kullanıcı etkileşimli multithread bir uygulamada, uygulamanın bir kısmı bloke olmuşsa veya uzun süren bir işlem yürütülüyorsa bile uygulama çalışmasını sürdürebilir, böylece kullanıcıya cevap verebilirlik özelliği artmış olur.
2. **Resource sharing**: Process'ler kaynaklarını ancak shared memory veya message passing aracılığıyla paylaşabilirler. Bu yöntemler programcı tarafından açıkça düzenlenmelidir.
    - Ancak thread'ler ait oldukları process'in sahip olduğu hafıza alanını ve diğer kaynakları default olarak paylaşırlar.
    - Kod ve data kısmının paylaşılmasıyla bir uygulama, aynı adres uzayında farklı aktiviteleri gerçekleştiren farklı threadlere sahip olur.

Multithread programlamanın avantajları

3. **Economy:** Bir process oluştururken hafıza ve kaynak tahsis edilmesi maliyetli bir iştir.
   - Threadler ait oldukları process'in kaynaklarını paylaştıklarından dolayı thread oluşturma ve threadler arası context switch işlemi düşük maliyetli ve daha hızlıdır.
4. **Scalability (Ölçeklenebilirlik):** Multithread çalışmanın avantajları, threadlerin **farklı işlemci** çekirdeklerinde paralel olarak çalıştığı multiprocessor bir mimaride daha da fazladır.
   - Single-threaded bir process, kaç tane kullanılabilir işlemci olduğuna bakılmaksızın yalnızca bir işlemcide çalıştırılabilir, birden çok işlemci olsa dahi onlardan faydalanılamaz.
   - Multiprocessor bir mimaride multithread kullanım paralelliği artırır.

Konular

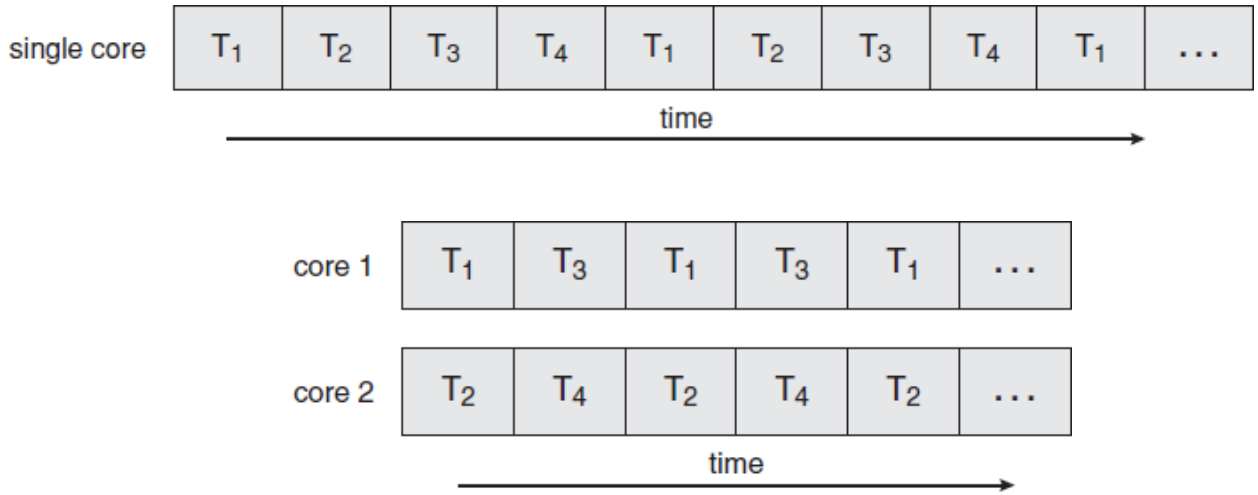
- Thread'ler
- Multithread programlamanın avantajları
- Multicore programlama
  - Multicore programlamanın zorlukları
  - Paralel çalışma türleri
- Multithreading modelleri
  - Many-to-one
  - One-to-one
  - Many-to-many
- Thread kütüphaneleri
- Dolaylı thread oluşturma
- Threadlerin yürütülmesi ile ilgili bazı hususlar
- Windows ve Linux thread'leri

Multicore programlama

- Bilgisayar tasarımındaki en önemli gelişmelerden birisi, çok işlemcili sistemlerin geliştirilmesidir.
- Son zamanlarda, tek chip içerisine birden fazla (çekirdek) core yerleştirilmektedir. Bu tür sistemler multicore veya multiprocessor olarak adlandırılır.
- Her bir core işletim sistemi tarafından ayrı bir işlemci olarak görünür.
- Parallelism (Paralellik), birden fazla görevin **aynı anda (simultaneously)** yapılmasını ifade eder.
- Concurrency (Eşzamanlılık), birden fazla görev arasında kısa aralıklarla geçiş yaparak görevlerin **birlikte ilerletilmesini** ifade eder. Parallelism olmadan concurrency, gerçekleştirilebilir.
  - Multiprocessor ve multicore mimarilerden önce çoğu bilgisayar single process idi, CPU scheduling yöntemiyle processler arasında hızlı geçişler yapılarak parallelism ilizyonu sağlanıyor, processler paralel olarak değil eşzamanlı olarak çalışıyordu.

Multicore programlama

- 4 thread'li olarak tasarlanmış bi uygulama düşünün:
  - Bu uygulamanın single core üzerinde **eşzamanlı (concurrency) çalışması**, threadlerin **belirli aralıklarla** çalıştırılmasını ifade eder.
  - Multicore sistemlerde **eşzamanlı (concurrency)** çalışma, her çekirdeğe bir thread atanarak thread'lerin **paralel** çalışmasını ifade eder.
  - Sistemdeki çekirdek sayısı arttıkça eşzamanlı gerçekleştirilen görev sayısı da artacaktır.



Amdahl kuralı

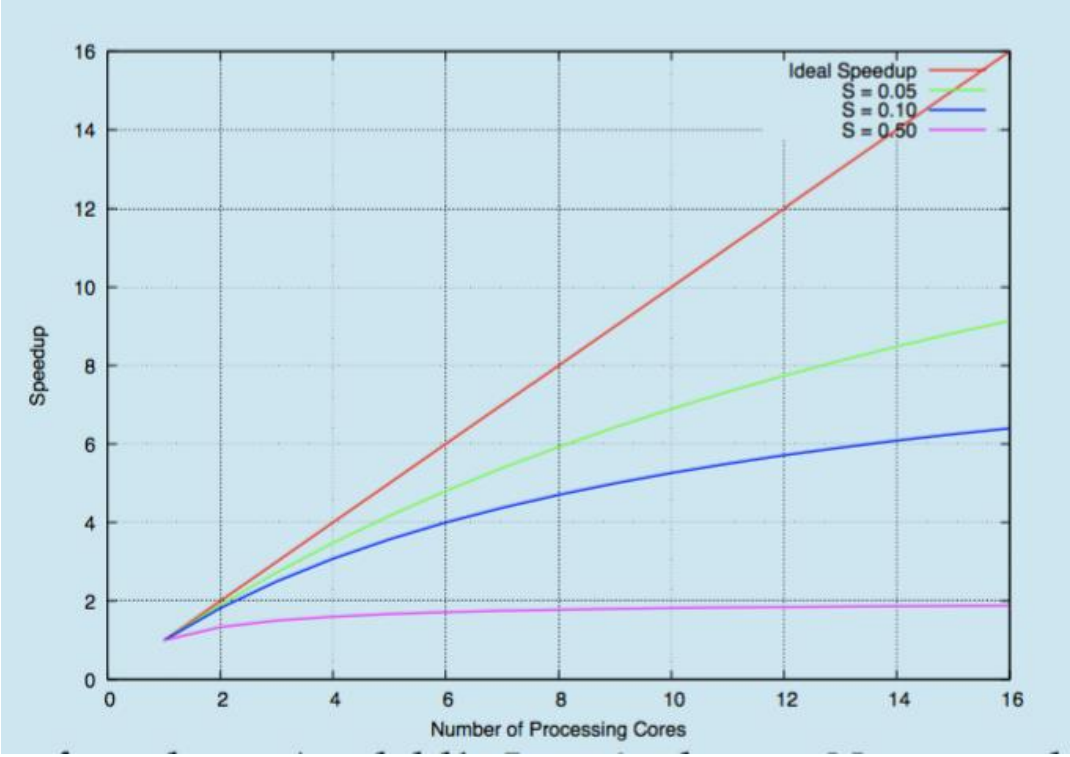
- **Amdahl kuralı** çekirdek (core) sayısına göre bir sistemdeki performans artışını aşağıdaki gibi ifade etmektedir:

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

- Burada, S uygulamada seri çalışması zorunlu olan kısmın oranını, N ise çekirdek sayısını ifade eder.
- Örneğin bir uygulama, %75 paralel ve %25 seri çalışıyorsa (S=0,25), 2 çekirdeğe (N=2) sahip sistemde bu uygulama 1,6 kat daha hızlı çalışır.
    - Çekirdek sayısı 4 olduğunda, 2,28 kat hız artışı sağlanır.
- Amdahl Yasası ile ilgili ilginç bir gerçek, N sonsuza yaklaştıkça hızlanma 1⁄S'ye yakınsar.
- Intel işlemciler her çekirdek için 2 thread, Oracle Sparc M7 işlemci, her processor için 8 çekirdek, her çekirdek için 8 thread destekler.

Amdahl kuralı

- Şekilde Amdahl Yasasını birkaç farklı senaryoda (çekirdek satısı arttıkça farklı S değerlerine göre hız) gösteren bir grafik bulunmaktadır.
- Örneğin, bir uygulamanın yüzde 50'si seri olarak yapılırsa, eklediğimiz işlemci çekirdeği sayısına bakılmaksızın maksimum hızlanma 2,0 kat olur.



Multicore programlamanın zorlukları

- İşletim sistemi tasarımcılarının paralel çalıştırmayı sağlamak için birden çok çekirdeği kullanan scheduling algoritmalarını yazmaları gerekir.
- Uygulama geliştiricilerin mevcut programları multithread olarak değiştirmelerinin ve yeni programları multihread şekilde tasarlamalarının bazı zorlukları vardır.

Multicore programlamanın zorlukları

- Multicore programlamada 5 önemli zorluk vardır:

1. **Identifying tasks:** Uygulamalar birbirine bağımlı olmayan eşzamanlı çalışabilecek görevler şeklinde tasarlanmalıdır. Birbirinden bağımsız görevler farklı çekirdekler üzerinde paralel çalışabilir.
2. **Balance**: Programcılar paralel çalışabilecek görevleri eşit değerde ve eşit iş yükünde olacak şekilde belirlemelidirler.
    - Bazı durumlarda, belli bir görev, tüm işe diğer görevler kadar değer katmayabilir. Bu görev için ayrı bir çekirdek kullanmak, maliyete değmeyebilir.
3. **Data splitting**: Verilerin farklı çekirdekler üzerinde çalışan görevler tarafından erişilecek ve işlem yapılacak şekilde ayrıştırılması gereklidir.

Multicore programlamanın zorlukları

4. **Data dependency**: Bir görevin erişeceği verinin diğer görevlerle bağımlılığının incelenmesi gereklidir.
   - Bir görev diğerinden gelen verilere bağlı olduğunda, görevlerin yürütülmesi programcılar tarafından senkronize edilmelidir.
5. **Testing and debugging**: Bir program birden çok çekirdekte paralel olarak çalıştığında, birçok farklı yürütme yolu mümkündür, eşzamanlı çalışan programların test ve debug işlemi daha zordur**.**

Paralel çalışma türleri

- Genel olarak paralel çalışmanın iki türü vardır: data parallelism ve task parallelism
- Data parallelism**,** aynı verinin alt kümelerinin çekirdeklere dağıtılmasına ve aynı işlemin her çekirdek üzerinde eşzamanlı yürütülmesine odaklanır.
  - Örneğin N elemanlı bir dizinin toplamı için iki çekirdek kullanılacaksa, dizinin yarısı 1. çekirdekte, diğer yarısı 2. çekirdekte **paralel** toplanır.
- Task parallelism**,** çekirdeklere veri dağıtılması yerine görevlerin (thread'ler) dağıtılmasına odaklanır.
  - Her thread **ayrı bir görevi gerçekleştirir**. Farklı thread'ler aynı veride veya farklı veride çalışabilir.
  - Örneğin aynı dizi elemanları üzerinde **farklı** istatistiksel hesaplamalar yapan threadler farklı çekirdeklerde aynı veriyi kullanarak **paralel** çalışır.

Paralel çalışma türleri

- Bir uygulama hybrid bir şekilde bu iki yöntemi birlikte kullanabilir.

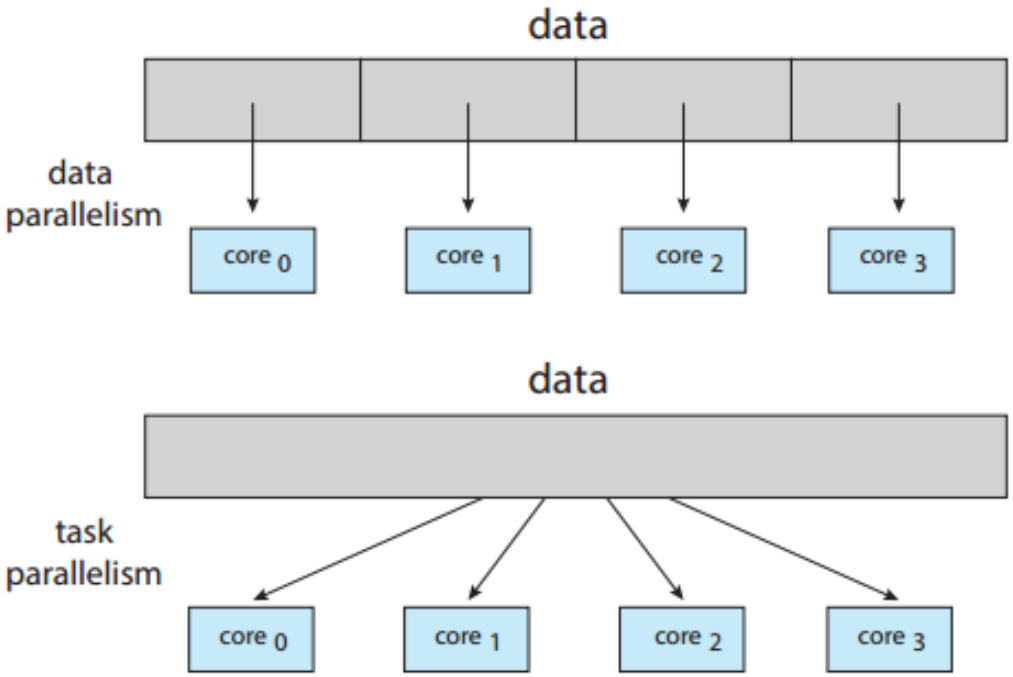


Figure 4.5 Data and task parallelism.

Konular

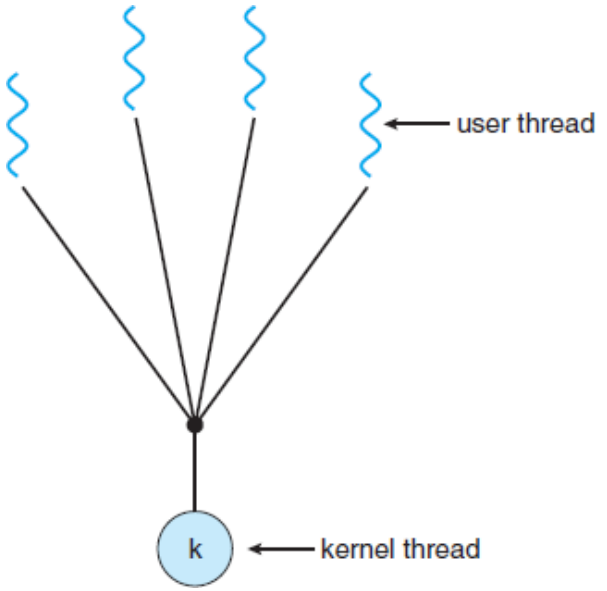
- Thread'ler
- Multithread programlamanın avantajları
- Multicore programlama
  - Multicore programlamanın zorlukları
  - Paralel çalışma türleri
- Multithreading modelleri
  - Many-to-one
  - One-to-one
  - Many-to-many
- Thread kütüphaneleri
- Dolaylı thread oluşturma
- Threadlerin yürütülmesi ile ilgili bazı hususlar
- Windows ve Linux thread'leri

Multithreading 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 ve Solaris gibi tüm modern işletim sistemleri kernel thread'leri destekler.
- Kernel thread'leri ile user thread'leri arasında bir ilişkilendirmenin yapılması gerekir. İlişkilendirmeyi kurmanın üç yaygın yolu vardır:
  - Many-to-one model
  - One-to-one model
  - Many-to-many model

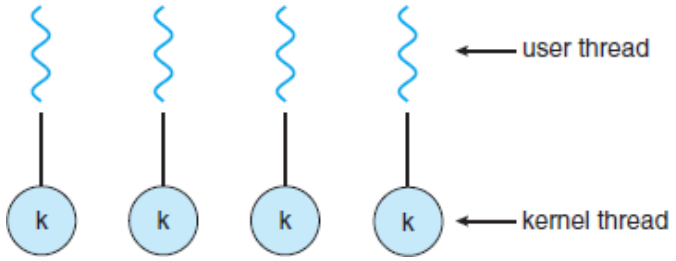
Many-to-one

- Many-to-one modelinde, çok sayıda kullanıcı thread'i bir tane kernel thread'i ile eşleştirilir (Solaris sistemleri).
- Eğer bir thread bloke olmasına neden olacak bir sistem çağrısı yaparsa tüm process bloke olur.
- Aynı anda sadece bir tane kullanıcı thead'i kernel thread'e erişebildiğinden multicore sistemlerde birden fazla thread eşzamanlı çalışamaz. Bu yüzden pek tercih edilmez.



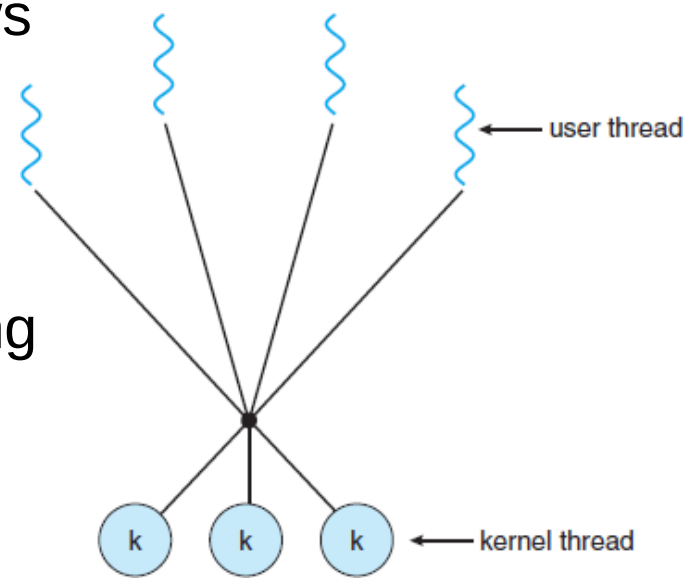
One-to-one

- One-to-one modelinde, bir kullanıcı thread'i bir kernel thread'i ile eşleştirilir (Linux ve Windows sistemlerde).
- Eğer bir thread bloke olmasına neden olacak bir sistem çağırısı yaparsa diğer thread'ler çalışmasına devam eder.
- Bu model ile birden fazla threadin multicore sistemlerde eşzamanlı çalışmasına izin verilir.
- Bu modelin tek dezavantajı, bir user thread oluşturmak için buna karşılık gelecek bir kernel threadi oluşturmayı gerektirmesidir. Çok sayıda kernel thread sistemin performansını etkiler.



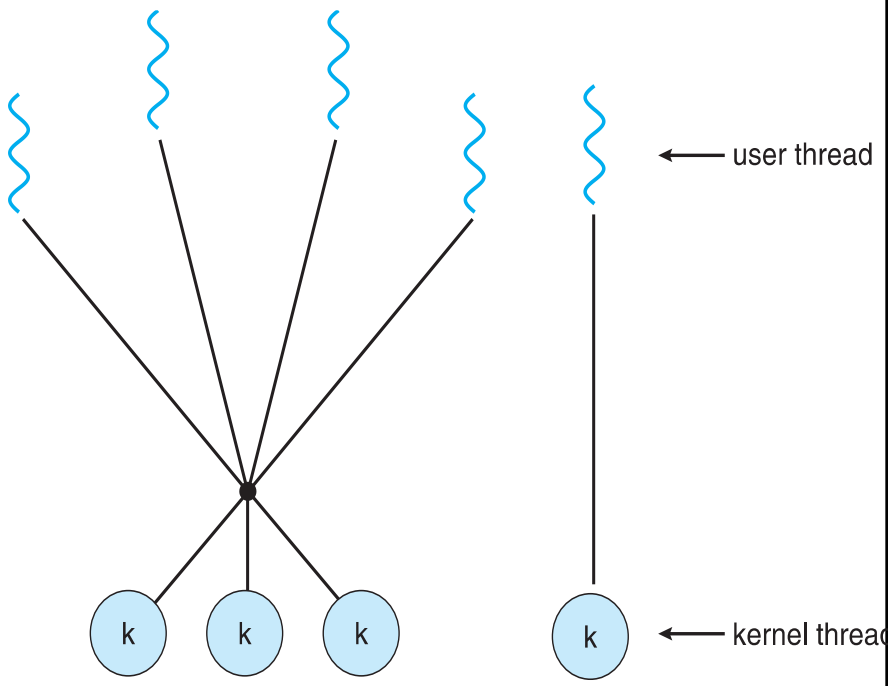
Many-to-many

- Many-to-many modelinde, çok sayıda kullanıcı thread'i ile aynı sayıdaki veya daha az sayıdaki kernel thread'i eşleştirilir (Solaris 9dan önceki versiyonlar, Windows ThreadFiber paketi).
- Eğer bir thread bloke olmasına neden olacak bir sistem çağırısı yaparsa kernel başka threadi çalıştırmak üzere scheduling yapar.
- Diğer iki modele göre daha esnektir, ancak gerçekleştirimi zordur.



Two-level model

- Many-to-many modeline benzer, ancak bir user thread'in bir kernel thread'ine bağlanmasına da izin verilir.
- Örnekler:
  - IRIX
  - HP-UX
  - Tru64 UNIX
  - Solaris 8 and earlier



Multithreading modelleri

- Sonuç olarak günümüzde çoğu işletim sistemi one-to-one modelini kullanmaktadır.
- Bununla birlikte, bazı modern eşzamanlı kütüphaneler, geliştiricilerin görevler tanımlamasına ve daha sonra bu görevlerin many-to-many modelini kullanarak thread'lerle eşleştirilmesini sağlarlar.

Konular

- Thread'ler
- Multithread programlamanın avantajları
- Multicore programlama
  - Multicore programlamanın zorlukları
  - Paralel çalışma türleri
- Multithreading modelleri
  - Many-to-one
  - One-to-one
  - Many-to-many
- Thread kütüphaneleri
- Dolaylı thread oluşturma
- Threadlerin yürütülmesi ile ilgili bazı hususlar
- Windows ve Linux thread'leri

Thread kütüphaneleri

- Thread kütüphanesi, programcıya thread oluşturmak ve yönetmek için API sağlar. Thread kütüphanesi oluşturulurken iki farklı yaklaşım kullanılır:
  1. Tüm thread kütüphanesi kullanıcı alanında oluşturulur ve kernel desteği yoktur.
  2. İşletim sisteminin doğrudan desteklediği kernel seviyesinde kütüphane oluşturulur.
- Multithread oluşturmak için iki farklı strateji kullanılmaktadır:
  - Asynchronous threading: Parent, yeni bir child thread oluşturduğunda eşzamanlı olarak çalışmasını sürdürür. Multithread serverlerde ve aynı zamanda responsive user interface tasarlamak için yaygın olarak kullanılır.
  - Synchronous threading: Parent, child thread oluşturduğunda çalışmasını durdurur ve tüm child thread'ler sonlandığında çalışmasına devam eder (fork-join strategy).
- Asynchronous threading, thread'ler arasında veri paylaşımı az olduğunda, Synchronous threading ise threadler arasında veri paylaşımı çok olduğunda kullanılır.

Thread kütüphaneleri

- Günümüzde 3 temel thread kütüphanesi kullanılmaktadır:
  - POSIX Pthreads
  - Windows
  - Java
- **Pthreads,** user-level veya kernel-level thread kütüphanesi sağlar.
- **Windows threads,** kernel-level thread kütüphanesi sağlar.
- **Java threads,** user-level thread kütüphanesi sağlar. Threadler doğrudan Java programlarında oluşturulur ve yönetilir.

Pthreads

- Pthreads standart API (IEEE1003.1c) thread oluşturma ve yönetmek için oluşturulmuştur.
- Çoğu Unix sistemlerde, Linux ve Mac OS'ta Pthreads kullanılır.
- Windows Pthreads standardını desteklemez. (third-party uygulamalarla mümkün)
- Bir Pthreads programda, ayrı threadler belirlenmiş bir fonksiyon içinde yürütülür, bu fonksiyon ismi thread oluşturulurken parametre olarak verilir.
- Tüm thread'ler global scope'ta tanımlanan verileri paylaşırlar.

Thread kütüphaneleri

- Sonraki slaytlarda, bu üç thread kütüphanesini kullanarak temel thread oluşturma gösterilecektir.
- Açıklayıcı bir örnek olarak, 1'den N'e kadar pozitif tam sayıların toplamını ayrı bir thread ile gerçekleştiren multithread bir program gösterilecektir.

$$sum = \sum_{i=1}^{N} i$$

Pthreads - Örnek

```
#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;
   }
```

```
   /* 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 programları için kullanılan header file

Yeni thread için ID tanımlar.

Yeni thread için özellikleri (stack size, …) belirler.

Varsayılan özellikler (senkron thread, sistem stack addr, …)

Yeni thread başlatıldı.

Yeni thread için başlama noktası.

Komut satırında girilen parametre

Komut satırında girilen parametre

fork-join stratejisi

Dönen değer

Pthreads - Örnek

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

```
#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);
```

Oluşturulacak thread sayısı

10 thread tanımlandı.

10 thread için fork-join yapıldı.

Windows threads

- Windows thread kütüphanesi ile thread oluşturma Pthreads ile birçok açıdan benzerlik gösterir.
- Tüm thread'ler global scope'ta tanımlanan verileri paylaşırlar.

Windows threads

Thread oluşturmak için kullanılan header file

Yeni thread için başlama noktası.

```
#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;
   }
```

DWORD-32 bit unsigned int
LPVOID- Void için pointer

```
   /* 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);
   }
}
```

fork-join stratejisi **WaitForMultipleObjects()** ile birden fazla thread senkron çalıştırılabilir.

Java threads

- Java thread'leri, JVM (Java Virtual Machine) kullanılabilen tüm sistemlerde çalışır.
- Java thread API; Windows, Linux, Unix, Mac OS X ve Android için kullanılabilir.
- Java thread'leri arasında veri paylaşımı parametre aktarımı ile yapılır.
- Java ile iki farklı teknik kullanılarak thread oluşturulabilir:
  - Thread sınıfından kalıtım yoluyla yeni bir sınıf türetilir ve run() metodu override yapılır.
  - Runnable arayüzünü kullanan bir sınıf oluşturulur. (yaygın kullanılır.)

Java threads

Implementing Runnable interface:

```
class Task implements Runnable
{
   public void run() {
      System.out.println("I am a thread.");
   }
}
```

Creating a thread:

```
Thread worker = new Thread(new Task());
worker.start();
```

Waiting on a thread:

```
try {
   worker.join();
}
catch (InterruptedException ie) { }
```

Thread kütüphaneleri

Java threads – ..

Ayrı thread için gereklidir.

Yeni thread oluşturuldu.

Yeni thread başladı.
run() çağırılır.

```
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);
   }
}
```

```
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>"); }
}
```

fork-join stratejisi
Thread'ler senkron çalışır.

Java Executor Framework

- Java 1.5 sürümünden başlayarak geliştiricilere thread oluşturma ve thread'lerle iletişim üzerinde çok daha fazla kontrol sağlayan birkaç yeni eşzamanlılık (concurrency) özelliği sunulmuştur.
  - Bu araçlar java.util.concurrent paketinde mevcuttur.
- Java, explicit şekilde thread oluşturmak yerine Executor interface framework'unde thread oluşturmaya da izin verir:

```
public interface Executor
{
   void execute(Runnable command);
}
```

- Executor aşağıdaki gibi kullanılır:

```
Executor service = new Executor;
service.execute(new Task());
```

  - Ayrı bir Thread nesnesi oluşturmak ve onun start() yöntemini çağırmak yerine **execute** kullanılır.

Java Executor Framework

- Aynı process'e ait thread'ler arasında veri paylaşımı, paylaşılan veriler basitçe global olarak bildirildiğinden Windows ve Pthreads'de kolayca gerçekleşir.
- Java'da Runnable uygulayan bir sınıfa parametreleri iletebiliriz, ancak Java thread'leri sonuçları döndüremez.
- Bu ihtiyacı karşılamak için, Java.util.concurrent package tanımlanmıştır. Bu paket ayrıca Runnable'a benzer şekilde davranan Callable interface'i de tanımlar.
- Callable görevlerden döndürülen sonuçlar Future objects olarak bilinir. Future interface'de tanımlanan get() yöntemi ile thread'in return ile döndürdüğü değer alınabilir.

Java Executor Framework

- call metoduna thread'in yapacağı iş yazılır.

```
import java.util.concurrent.*;

class Summation implements Callable<Integer>
{
   private int upper;
   public Summation(int upper) {
      this.upper = upper;
   }

   /* The thread will execute in this method */
   public Integer call() {
      int sum = 0;
      for (int i = 1; i <= upper; i++)
         sum += i;

      return new Integer(sum);
   }
}
```

Java Executor Framework

```
public class Driver
{
 public static void main(String[] args) {
    int upper = Integer.parseInt(args[0]);

    ExecutorService pool = Executors.newSingleThreadExecutor();
    Future<Integer> result = pool.submit(new Summation(upper));

    try {
        System.out.println("sum = " + result.get());
    } catch (InterruptedException | ExecutionException ie) { }
  }
}
```

Konular

- Thread'ler
- Multithread programlamanın avantajları
- Multicore programlama
  - Multicore programlamanın zorlukları
  - Paralel çalışma türleri
- Multithreading modelleri
  - Many-to-one
  - One-to-one
  - Many-to-many
- Thread kütüphaneleri
- Dolaylı thread oluşturma
- Threadlerin yürütülmesi ile ilgili bazı hususlar
- Windows ve Linux thread'leri

Implicit threading

- Multicore 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 ve run-time kütüphaneler tarafından yapılması** günümüzde giderek popüler hale gelmektedir.*
- Bu stratejiye implicit threading denilmektedir. Multicore işlemcilerden yararlanabilecek uygulamalar tasarlamak için alternatif implicit threading yöntemleri:
  - Thread Pools
  - Fork-Join
  - OpenMP
  - Grand Central Dispatch
  - Intel Threading Building Blocks

Thread pools

- Multithreaded bir web sunucu, gelen isteklerin her birisi için yeni thread oluşturur.
- Multihreaded bir web sunucu, eşzamanlı çok sayıda istemciye servis sağlayabilir. Ayrı bir thread oluşturmak, ayrı bir process oluşturmaktan kesinlikle daha avantajlı iken, multihreaded bir sunucunun yine de potansiyel sorunları vardır.
  - Sorunlardan biri gelen istek sayısının çok artmasıyla sistem kaynaklarının (CPU time, memory, …) tükenmesidir.
- Bu sorunun çözümü için thread pool bir yöntemdir, multithreaded sistemlerde sınırlı sayıda thread oluşturulmasını sağlar.

Thread pools

- thread pool şöyle çalışır:
  - Başlangıçta belli sayıda thread oluşturulur ve çalıştırılmak üzere havuzda beklerler.
  - Sunucuya bir istek geldiğinde yeni bir thread oluşturmak yerine gelen istek thread pool'a gönderilir ve başka istekler dinlemeye devam edilir.
  - Thread pool içerisinde kullanılabilir thread varsa isteğe hemen cevap verilir, yoksa bir thread'in serbest hale gelmesi beklenir.

Thread pools

- thread pool avantajları:
  - Bir isteğe var olan thread ile servis yapmak yeni thread oluşturmaktan daha hızlıdır.
  - Çok sayıda eşzamanlı threadleri destekleyecek kapasitesi olmayan sistemler için thred pool ile threadleri sınırlı sayıda tutmak önemlidir.
  - Yapılacak görevi, görevi oluşturma mekaniğinden ayırmak, görevi yürütmek için farklı stratejiler kullanmaya imkan tanır. Örneğin, görevler periyodik olarak zamanlanabilir.

Thread pools - Windows

- Windows API thread pools destekler:
- `PoolFunction()` fonksiyonu thread olarak çalıştırılmaktadır.

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

- Thread pool API içerisindeki `QueueUserWorkItem()` fonksiyonu, pool içerisindeki bir thread ile PoolFunction() fonksiyonunu çalıştırır.

```
QueueUserWorkItem(&PoolFunction, NULL, 0);
```

- QueueUserWorkItem() fonksiyonun sırasıyla parametreleri:
  - LPTHREAD_START_ROUTINE, thread olarak çalışacak fonksiyonun pointer'ı
  - PVOID, Gönderilecek parametre
  - ULONG, Bayrak bitleri (bekleme süresi, I/O gerekliliği, …)

Java Thread Pools

- Java.util.concurrent paketi, çeşitli thread pool mimarileri için bir API içerir:
  - static ExecutorService newSingleThreadExecutor(), 1 boyutunda bir pool oluşturur.
  - static ExecutorService newFixedThreadPool(int size), size boyutunda bir pool oluşturur.
  - static ExecutorService newCachedThreadPool(), birçok durumda thread'leri yeniden kullanarak sınırsız bir thread pool oluşturur.

Java Thread Pools

- Örnekte, CachedThreadPool ile bir thread pool oluşturulmuş ve execute() yöntemi ile pool'daki her bir thread'in yürüteceği görevleri gönderilmiştir.
- **shutdown**() yöntemi ile thread pool bundan sonra gelen görevleri reddeder ve mevcut tüm görevler yürütmeyi tamamladıktan sonra kapanır.

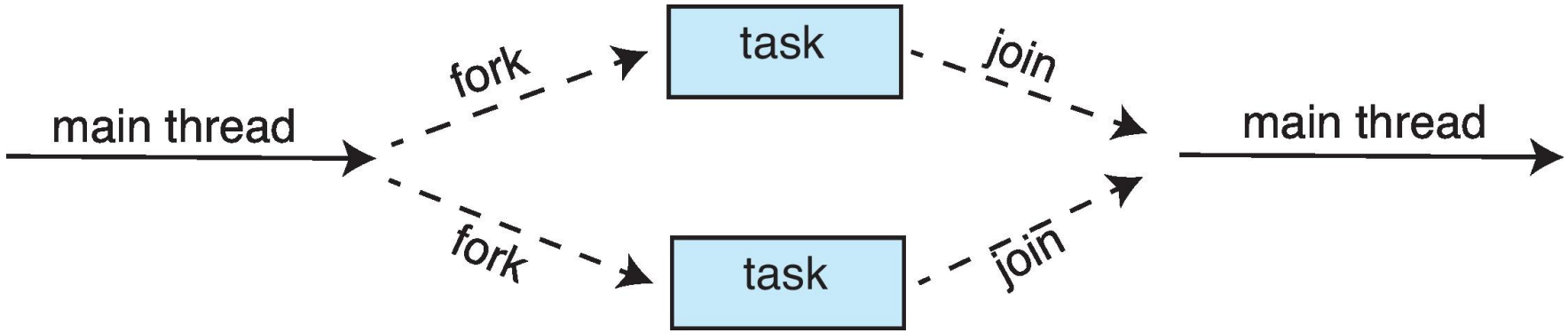
```
import java.util.concurrent.*;

public class ThreadPoolExample
{
public static void main(String[] args) {
   int numTasks = Integer.parseInt(args[0].trim());

   /* Create the thread pool */
   ExecutorService pool = Executors.newCachedThreadPool();

   /* Run each task using a thread in the pool */
   for (int i = 0; i < numTasks; i++)
      pool.execute(new Task());

   /* Shut down the pool once all threads have completed */
   pool.shutdown();
}
```

Implicit threading yöntemleri

- Thread Pools
- **Fork-Join**
- OpenMP
- Grand Central Dispatch
- Intel Threading Building Blocks

Fork-Join Parallelism

- Main thread bir veya daha fazla child thread oluşturur (fork) ve daha sonra child'lerin sonlanmasını bekler daha sonra sonuçları alıp birleştirebilir.
- Bir kütüphane, oluşturulan thread'lerin sayısını yönetmek ve thread'lere görev atamaktan sorumludur.

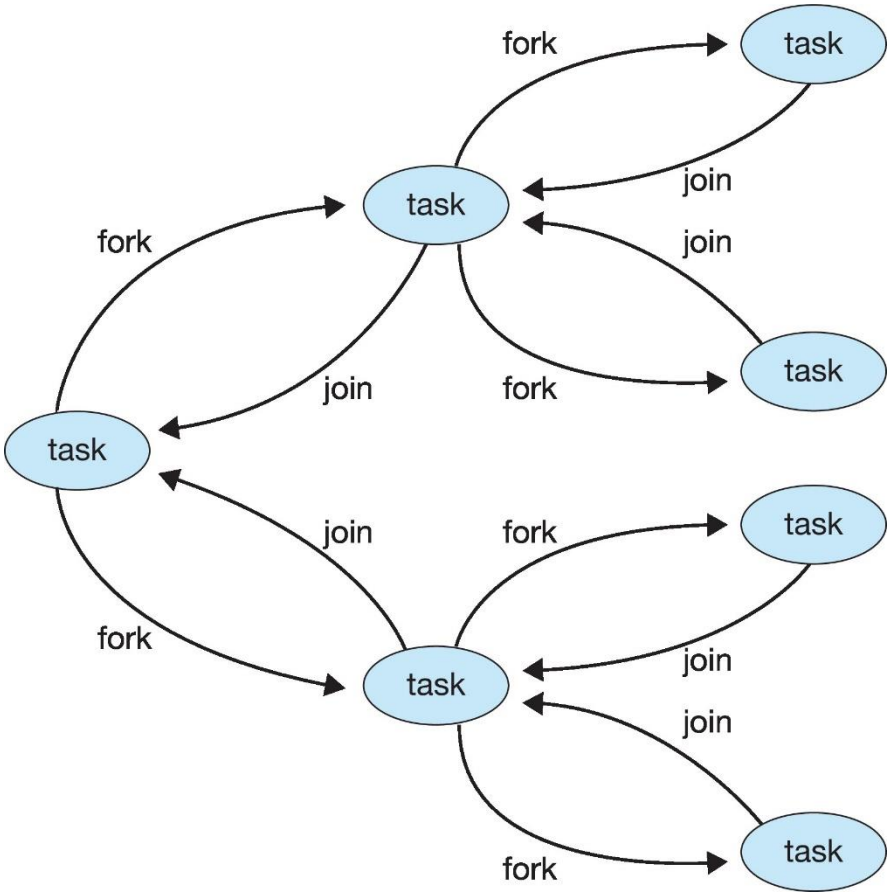


Fork-Join in Java

- Java API'nin 1.7 sürümünde , Quicksort ve Mergesort gibi özyinelemeli böl ve yönet (divide-and-conquer) algoritmalarıyla kullanılmak üzere tasarlanmış bir fork-join kütüphanesi sunulmuştur.
- Java'nın genel özyinelemeli algoritma arkasındaki fork-join modeli aşağıda gösterilmiştir:

```
Task(problem)
   if problem is small enough
      solve the problem directly
   else
      subtask1 = fork(new Task(subset of problem)
      subtask2 = fork(new Task(subset of problem)

      result1 = join(subtask1)
      result2 = join(subtask2)

      return combined results
```

Fork-Join in Java



Fork-Join in Java

- Örneğin bir tamsayı dizisindeki tüm öğeleri toplayan bir böl ve yönet algoritmasını Java fork-join stratejisi ile yapacak olursak:
- ForkJoinPool() ile bir thread pool oluşturulur, ve bu thread pool'a yapılacak görevler gönderilir:

```
ForkJoinPool pool = new ForkJoinPool();
// array contains the integers to be summed
int[] array = new int[SIZE];

SumTask task = new SumTask(0, SIZE - 1, array);
int sum = pool.invoke(task);
```

Fork-Join in Java

- Örnekte THRESHOLD 1000 olarak belirlenmiştir.
- Toplanacak sayılar 1000'den az ise yeni child thread'ler oluşturulmadan iş doğrudan yapılır,
- 1000'den fazla ise böl ve yönet özyinelemeli çağrı ile child thread'ler oluşturulur.

```
import java.util.concurrent.*;

public class SumTask extends RecursiveTask<Integer>
{
   static final int THRESHOLD = 1000;

   private int begin;
   private int end;
   private int[] array;

   public SumTask(int begin, int end, int[] array) {
      this.begin = begin;
      this.end = end;
      this.array = array;
   }

   protected Integer compute() {
      if (end - begin < THRESHOLD) {
         int sum = 0;
         for (int i = begin; i <= end; i++)
            sum += array[i];

         return sum;
      }
      else {
         int mid = (begin + end) / 2;

         SumTask leftTask = new SumTask(begin, mid, array);
         SumTask rightTask = new SumTask(mid + 1, end, array);

         leftTask.fork();
         rightTask.fork();

         return rightTask.join() + leftTask.join();
      }
   }
}
```

Implicit threading yöntemleri

- Thread Pools
- Fork-Join
- **OpenMP**
- Grand Central Dispatch
- Intel Threading Building Blocks

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];
}
```

- OpenMP, Linux, Windows ve Mac OS X sistemlerdeki çok sayıda açık kaynak ve ticari compiler'larda kullanılabilir.

OpenMP - Örnek

```
#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;
}
```

printf() deyimi paralel çalışacaktır.

Implicit threading yöntemleri

- Thread Pools
- Fork-Join
- OpenMP
- **Grand Central Dispatch**
- Intel Threading Building Blocks

Grand central dispatch

- Grand central dispatch (GCD), Apple Mac OS X ve iOS işletim sistemlerinde paralel çalışacak kısımları belirlemek için kullanılır.
- Paralel çalışacak bloğun belirtilmesi için ^ sembolü kullanılır.

```
^{ printf("I am a block"); }
```

- GCD, blokları dispatch queue içerisine yerleştirir.
- Bir blok kuyruktan atılırsa, tekrar thread havuzundaki bir thread'e atanabilir.
- Dispatch queue, serial veya concurrent şeklinde oluşturulabilir.
- Serial queue, FIFO çalışır ve sadece bir blok kuyruktan alınabilir.
- Concurent queue, FIFO çalışır ve kuyruktan birden fazla blok aynı anda alınabilir.

Grand central dispatch

- Concurrent queue, önceliklendirilmiş 3 tane dispatch kuyruğa sahiptir: **low, default** ve **high**.
- Önceliklendirme ile blokların önem derecesi belirlenmektedir.
- Aşağıdaki örnekte, default önceliğe sahip concurrent kuyruğa bir blok eklenmektedir.

```
dispatch_queue_t queue = dispatch_get_global_queue
   (DISPATCH_QUEUE_PRIORITY_DEFAULT, 0);

dispatch_async(queue, ^{ printf("I am a block."); });
```

Kuyruğa blok eklendi.

default öncelikli concurrent kuyruk

Implicit threading yöntemleri

- Thread Pools
- Fork-Join
- OpenMP
- Grand Central Dispatch
- **Intel Threading Building Blocks**

Intel Threading Building Blocks (TBB)

- Intel thread oluşturma blokları (TBB), C++'da paralel uygulamalar tasarlamayı destekleyen bir template kütüphanedir.
- Örneğin parametre değeri üzerinde bir işlem gerçekleştiren apply(float value) adlı bir fonksiyon olduğunu varsayalım.
- float değerler içeren n boyutunda bir v dizimiz olsaydı, v içindeki her değeri apply() fonksiyonuna iletmek için aşağıdaki seri for döngüsünü kullanabilirdik:

```
for (int i = 0; i < n; i++) {
   apply(v[i]);
}
```

- Yukarıdaki seri for döngüsünü aşağıdaki gibi TBB parallel_for şablonunu kullanarak yeniden yazabiliriz:

```
parallel_for (size_t(0), n, [=](size_t i) {apply(v[i]);});
```

Konular

- Thread'ler
- Multithread programlamanın avantajları
- Multicore programlama
  - Multicore programlamanın zorlukları
  - Paralel çalışma türleri
- Multithreading modelleri
  - Many-to-one
  - One-to-one
  - Many-to-many
- Thread kütüphaneleri
- Dolaylı thread oluşturma
- Threadlerin yürütülmesi ile ilgili bazı hususlar
- Windows ve Linux thread'leri

Threadlerin yürütülmesi ile ilgili bazı hususlar

- Semantics of fork() and exec() system calls
- Signal handling
  - Synchronous and asynchronous
- Thread cancellation of target thread
  - Asynchronous or deferred
- Thread-local storage
- Scheduler Activations

fork() ve exec() sistem çağrıları

- Processler konusunda fork() sistem çağrısıyla ayrı veya kopya processler oluşturulduğu açıklanmıştı.
- Bazı Unix sistemlerde threadler için fork() çağrısı iki türlü uygulanır:
  - Tüm threadler kopyalanır.
  - Sadece fork() çağrısını yapan thread kopyalanır.
- Bir thread exec() sistem çağrısı yaparsa kopya process ve kopya threadler oluşturmak yerine, exec() metoduna parametre olarak verilen belirlenmiş bir programı yürütmek üzere processin tamamı ve tüm threadler değiştirilir.

fork() ve exec() sistem çağrıları

- Tüm threadlerin kopyalanıp kopyalanmayacağı fork() çağrısının nasıl kullanıldığına bağlıdır:
  - Eğer fork() çağrısından hemen sonra exec() çağrısı yapılırsa tüm threadlerin kopyalanmasına gerek yoktur, process exec() metotunda belirlenmiş programı yürütmek üzere değiştirilir. Sadece exec() çağrısını yapan thread kopyalanır.
  - Eğer process, fork() çağrısından hemen sonra exec() çağrısını yapmamışsa tüm threadlerin kopyası oluşturulur.

Signal handling

- Unix sistemlerde bir signal belirli bir olayın gerçekleştiğini gösterir. Sinyaller için aşağıdaki durumlar ortaktır.
  1. Özel bir olay gerçekleştiğinde sinyal oluşturulur.
  2. Sinyal ilgili processe iletilir.
  3. Sinyal iletildikten sonra ya default şeklinde ya da user-defined şeklinde ele alınmalıdır.

Signal handling

- Oluşan sinyal, senkron veya asenkron alınabilir.
  - senkron sinyal, sinyalin oluşmasına neden olan olayı gerçekleştiren process'e iletilir. Örneğin illegal hafıza erişimi veya 0'a bölme gibi.
  - Çalışan bir processin dışında harici bir sinyal oluşturulduğunda process bu sinyali asenkron şekilde alır. Örneğin belirli klavye tuşlarına basarak (<ctrl><C> gibi) bir processi sonlandırmak gibi.

Signal handling

- Multithread sistemler için sinyaller farklı hedeflere gönderilebilir:
  - Sinyale neden olan threade gönderilebilir.
  - Process içerisindeki tüm threadlere gönderilebilir.
  - Process içerisindeki belirli threadlere gönderilebilir.
  - Bir process için tüm sinyalleri almak üzere bir thread belirlenebilir.

Thread Cancellation

- **Thread'in iptal edilmesi (cancellation) -** bir threadin tamamlanmadan çalışmasının sonlandırılmasıdır.
- İstenen bir sonucun bir thread tarafından bulunması halinde diğerleri iptal edilebilir.
  - Örneğin bir veritabanında threadlerle eşzamanlı bir arama yapılırken bir thread sonucu döndürdüğünde geri kalan threadler iptal edilebilir.
  - Bir başka örnek, bir Web sayfası yüklenirken stop butonuna basıldığında process içerisindeki web sayfasının yüklenmesiyle ilgi çeşitli işleri yürüten tüm threadler iptal edilir.

Thread Cancellation

- Cancel edilecek thread **target thread** olarak ifade edilir.
- Genel olarak iki yaklaşım vardır:
  - **Asynchronous cancellation** - Bir thread başka bir thread'i aniden sonlandırabilir.
  - **Deferred cancellation** - Target thread (cancel edilecek thread) periyodik olarak sonlandırılıp sonlandırılmayacağını kontrol eder, bu şekilde kendisini sonlandırma fırsatı verilir.
- Pthread için thread create ve cancel komutları

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

Thread Cancellation

- Pthreads tabloda verilen üç farklı iptal etme modunu destekler.

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

- Eğer thread cancellation disabled ise, cancellation işlemi thread enable olana kadar bekletilir.
- Deferred varsayılan moddur.
  - cancellation sadece thread **cancellation point**'e ulaşınca olur, ulaştıktan sonra **cleanup handler** çağrılır.
- Linux sistemlerde cancellation sinyaller üzerinden ele alınır.

Thread-Local Storage

- Thread-local storage (TLS) her bir threadin kendisine ait kopya verisinin olmasını sağlar.
- Thread oluşturma işlemi üzerinde kontrole sahip değilseniz kullanışlıdır. (örneğin thread pool kullanırken)
- TLS'yi local değişkenlerle karıştırmak kolaydır.
  - Local değişkenler yalnızca tek bir fonksiyon çağrısı sırasında görülebilir.
  - TLS verileri fonksiyon çağrıları boyunca görülebilir.
- TLS static verilere benzerdir, farklı olarak TLS verisi her thread için unique olur.

Scheduler Activations

- Hem M: M hem de two level modellerde kernel ile thread kütüphanesi arasında bir iletişime ihtiyaç duyulur.
- Bu iletişim, en iyi performansı yakalamak için uygulamaya tahsis edilecek kernel thread sayısının dinamik olarak en uygun şekilde ayarlanması için gereklidir.
- Genellikle user ve kernel thread arasında aracı bir veri yapısı kullanılır - **lightweight process (LWP)**
  - User thread kütüphanesinde; LWP, uygulamanın çalıştırılacak bir user threadini schedule edebilen sanal bir işlemci gibi görünür.
  - Bir uygulamanın verimli bir şekilde çalışması için herhangi bir sayıda LWP gerekebilir.

Scheduler Activations

- **Scheduler activation**, user-thread kütüphanesi ile kernel arasında iletişimi sağlan bir mekanizmadır.
- Kernel, bir dizi sanal işlemciye (LWP) sahip bir uygulama sağlar, bu uygulama user threadlerini mevcut bir sanal işlemciye schedule edebilir.
- Ayrıca, kernel uygulamaya belirli olaylar hakkında bilgi vermelidir. Bu prosedür **upcall** olarak bilinir. **upcall**, thread kütüphanesi tarafından bir **upcall handler** ile işlenir, **upcall handler** bir sanal işlemci üzerinde çalıştırılmalıdır.
- **Scheduler activation** sayesinde bir uygulama için kernel thread sayısının en uygun şekilde olması sağlanır.

Konular

- Thread'ler
- Multithread programlamanın avantajları
- Multicore programlama
  - Multicore programlamanın zorlukları
  - Paralel çalışma türleri
- Multithreading modelleri
  - Many-to-one
  - One-to-one
  - Many-to-many
- Thread kütüphaneleri
- Dolaylı thread oluşturma
- Thread çalıştırma kuralları
- Windows ve Linux thread'leri

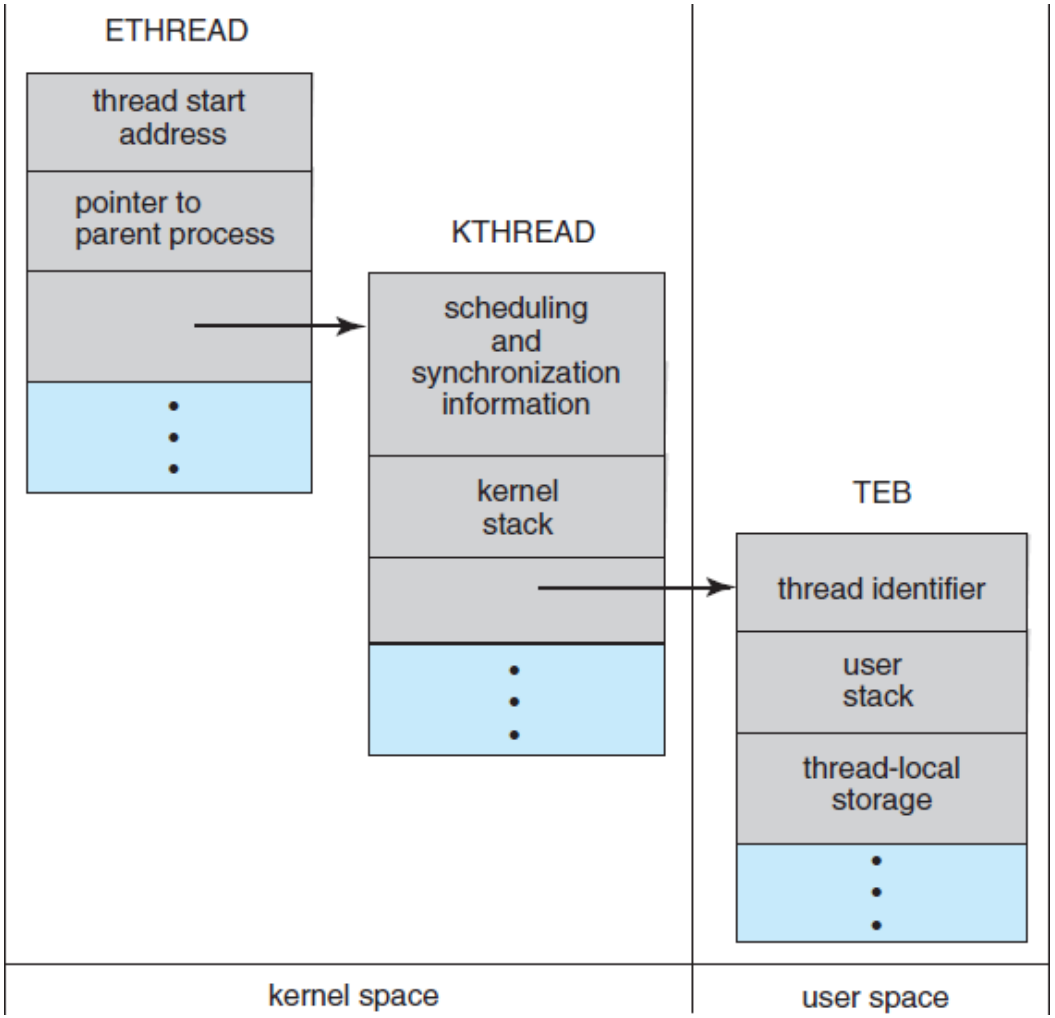
Windows thread'leri

- Microsoft işletim sistemleri için temel API olarak **Windows API** kullanılır.
- Bir Windows uygulaması ayrı bir process olarak çalışır ve her process bir veya daha fazla thread içerebilir.
- Windows, kullanıcı thread'leri ile kernel thread'leri arasında **one-to-one** eşleştirme yapar.
- Her thread şunları içerir:
  - **thread id**
  - **Register set**
  - **Ayrı user ve kernel stacks**
  - **Özel depolama alanı** - run-time libraries ve dynamic link libraries (DLLs) tarafından kullanılan
- Bir thread için ayrılan **register kümesi**, **stack**, **özel depolama alanı**, context olarak adlandırılır.

Windows thread'leri

- Windows bir thread için aşağıdaki veri yapılarını kullanır:
  - ETHREAD: Yürütücü thread blok
  - KTHREAD: Kernel thread blok
  - TEB: Thread environment (ortam) blok

Windows thread'leri



Linux thread'leri

- Linux, **fork()** sistem çağrısının yanı sıra **clone()** sistem çağrısı ile de thread oluşturabilir.
- Linux'te process ya da thread yerine **görev** (**task**) terimi kullanılır.
- **Görevler**, Linux kernel içerisinde `task_struct` isminde bir veri yapısına sahiptir, burada **görev** ile ilgili açık dosyalar, virtual memory, sinyal vb. bilgiler tutulur.
- Linux, fork() ile yeni bir görev başlattığında, **parent task** veri yapısı kopyalanır.

Linux thread'leri

- clone () çağrıldığında, **parent** ve **child task** arasında neyin paylaşılacağını belirten bir dizi flag kullanılır.

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.