

İşletim Sistemleri

Bölüm4: Threads (iplikler)

M.Ali Akcayol Gazi Üniversitesi Bilgisayar Müh. Ders sunularındır

Bu dersin sunumları, "Abraham Silberschatz, Greg Gagne, Peter B. Galvin, Operating System Concepts 9/e, Wiley, 2013." kitabı kullanılarak hazırlanmıştır.

Konular

- Thread'ler
- Thread'lerin sağladığı faydalar
- 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

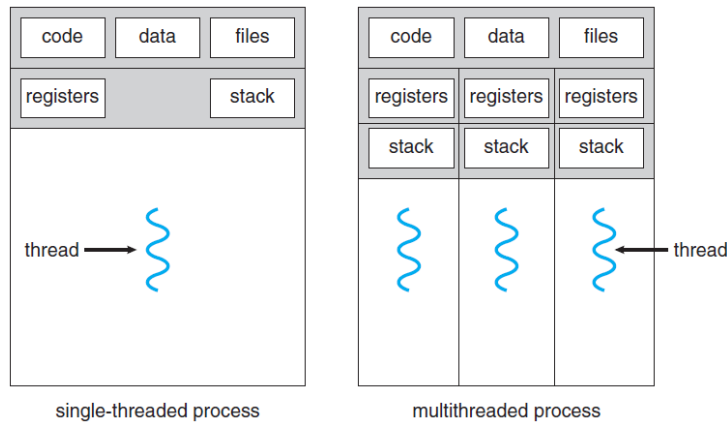
Thread'ler

- Bir **thread**, program counter, bir grup register ve bir stack yapısına sahiptir.
- Thread'ler, program kodunu, data kısmını, dosyalar gibi işletim sistemi kaynaklarını ortak kullanır.
- Klasik process'ler tek thread'e sahiptirler.
- Eğer bir process, birden fazla thread'e sahipse birden fazla görevi eşzamanlı yapabilir.
- Günümüzdeki modern bilgisayarlarda çalışan yazılım uygulamalarının çoğu **multithread** çalışır.
- Uygulamalar, çok sayıda thread'e sahip tek process şeklinde geliştirilirler.
- Bir Web browser, bir thread ile veri aktarımı yapabilir, başka thread ile verileri ekranda görüntüleyebilir.

3

Thread'ler

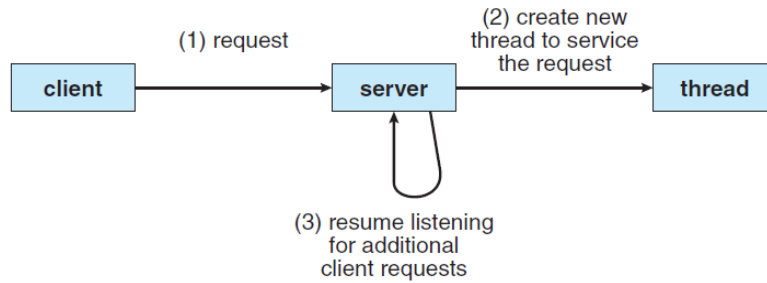
- Bir kelime işlemci uygulaması, **bir thread ile klavyeden giriş alabilir, bir thread ile spell check yapabilir ve başka bir thread ile ekran görüntüsünü düzenleyebilir.**
- Her thread, **paylaşmadan kullandığı** kendisine ait bileşenlere sahiptir.



4

Thread'ler

- **Uygulamalar**, multicore sistemlerin **kapasitesini maksimum kullanacak şekilde tasarlanabilir.**
- Bir Web sunucu process'i multithreaded çalışırsa, her gelen istek için ayrı bir thread oluşturulur ve process portu dinlemeye devam eder.
- **Çoğu işletim sisteminin kernel'ı multithreaded yapıdadır** ve cihazların yönetimi, hafıza yönetimi veya interrupt işlemi aynı anda yapılabilir.



Konular

- Thread'ler
- **Thread'lerin sağladığı faydalar**
- 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

Thread'lerin sağladığı faydalar

- Thread'lerin sağladığı faydalar 4 kategori halinde ifade edilebilir:

1. Responsiveness: Kullanıcı etkileşimli uygulamalarda, bir kısım **bloklanmış, kilitlenmiş** veya **uzun süren işlem yürütüyorsa**, kullanıcı ile etkileşim yapan **başka bir kısım çalışmasını sürdürür**.
Sistemin cevap verebilirlik özelliği artmış olur.

2. Resource sharing: Process'ler kaynaklarını shared memory veya message passing teknikleri aracılığıyla paylaşabilirler.
Thread'ler ait oldukları process'in sahip olduğu hafıza alanını ve diğer kaynakları paylaşabilirler.

7

Thread'lerin sağladığı faydalar

- Thread'lerin sağladığı faydalar 4 kategori halinde ifade edilebilir:

3. Economy: Bir process oluştururken **hafıza ve kaynak tahsis edilmesi** maliyeti yüksek bir iştir.
Thread'ler ait oldukları process'in kaynaklarını paylaştıklarından dolayı context switch daha düşük maliyetle yapılır.
(Solaris işletim sisteminde, **thread oluşturma 30 kat daha hızlıdır** ve **thread'lerde context switch 5 kat daha hızlıdır.**)

4. Scalability: Çok işlemcili mimarilerde thread'ler farklı core'lar üzerinde eşzamanlı çalışabilir.
Ancak, tek thread yapısına sahip process sadece bir işlemci üzerinde çalışabilir.

8

Konular

- Thread'ler
- Thread'lerin sağladığı faydalar
- **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

9

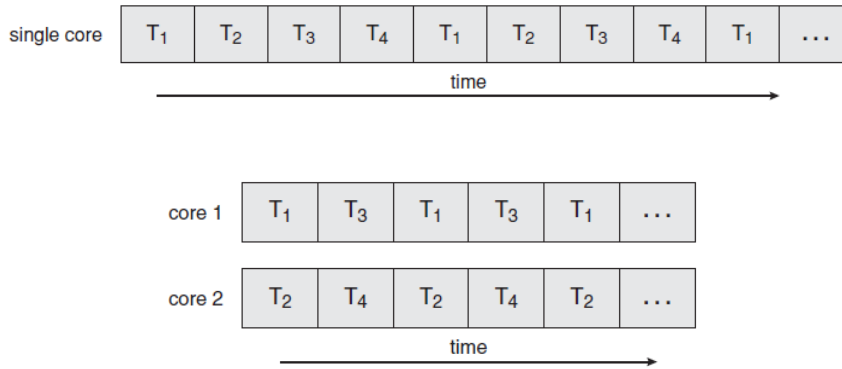
Multicore programlama

- Bilgisayar tasarımındaki en önemli gelişmelerden birisi, çok işlemcili sistemlerin geliştirilmesidir.
- Son zamanlarda, **tek chip içerisine birden fazla core yerleştirilmektedir.** Bu tür sistemler **multicore** veya **multiprocessor** olarak adlandırılır.
- Her bir **core işletim sistemi için ayrı bir işlemci olarak görünür.**
- **Bir core üzerinde çalışan 4 thread'e sahip bir uygulama için eşzamanlı çalışma, thread'lerin belirli aralıklarla çalıştırılmasını ifade eder.**
- **Çok core'a sahip sistemlerde eşzamanlı çalışma, her core'a bir thread atanarak thread'lerin paralel çalışmasını ifade eder.**
- **Parallelism**, birden fazla görevin **eşzamanlı** yapılmasını ifade eder.
- **Concurrency**, birden fazla görev arasında **kısa aralıklarla geçiş yaparak birlikte ilerletilmesini** ifade eder.

10

Multicore programlama

- Sistemdeki core sayısı arttıkça eşzamanlı gerçekleştirilen görev sayısı da artacaktır.



11

Multicore programlama

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

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

- Burada, S uygulamada seri çalışması zorunlu olan kısmın oranını, N ise core sayısını ifade eder.
- Bir uygulamada, %75 paralel ve %25 seri çalışıyorsa ($S=0,25$), **2 core'a** ($N=2$) sahip sistemde bu uygulamayı çalıştırınca **1,6 kat hız artar**.
- **Core sayısı 4 olduğunda, 2.28 kat hız artışı sağlanır.**
- **Core sayısı sonsuza giderken hız artışı ($1/S$) 'e doğru gider.**
- **Intel CPU'lar** her core için 2 thread, **Oracle T4 CPU** ise 4 thread destekler.

12

Konular

- Thread'ler
- Thread'lerin sağladığı faydalar
- 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

13

Multicore programlamanın zorlukları

- **İşletim sistemi tasarımcıları** multicore sistemlerin performansını artırmak için **scheduling algoritmaları yazmak zorundadır.**
- Uygulama geliştiricilerin mevcut programları değiştirmeleri ve **yeni programları multithreaded şekilde tasarlamaları gerekmektedir.**
- Multicore programlamada 5 önemli zorluk vardır:
 - **Identifying tasks:** Uygulamalarda **eşzamanlı çalışabilecek** ayrı alanların bulunması gereklidir. Bu alanlar farklı core'lar üzerinde paralel çalışacaktır.
 - **Balance:** Programcılar görevleri ayrıştırırken **iş yükünün eşit dağıtılması** gereklidir.
 - **Data splitting:** Verilerin farklı core'lar üzerinde çalışan görevler tarafından erişilecek ve işlem yapılacak şekilde ayrıştırılması gereklidir.
 - **Data dependency:** Bir görevin erişeceği verinin diğer görevlerle bağımlılığının incelenmesi gereklidir.
 - **Testing and debugging:** Multithreaded çalışan programların **test ve debug işlemi daha zordur.**

14

Konular

- Thread'ler
- Thread'lerin sağladığı faydalar
- 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

15

Paralel çalışma türleri

- Genel olarak **data parallelism** ve **task parallelism** olarak iki tür paralel çalışma türü vardır.
- **Data parallelism, aynı veri kümesine ait parçaların core'lara dağıtılması ve aynı tür işlemin eşzamanlı yürütülmesine odaklanır.**
- N elemanlı bir **dizinin toplamı** için **iki core** kullanılacaksa, **[0]..[(N/2)-1]** eleman 1.core'da, **[N/2]..[N-1]** eleman 2.core'da toplanır.
- **Task parallelism, core'lara görevlerin (thread'ler) dağıtılmasına odaklanır.**
- Her thread **ayrı bir işlemi gerçekleştirir**. Farklı thread'ler aynı veride veya farklı veride çalışabilir.
- **Aynı dizi** elemanları üzerinde **farklı istatistiksel hesaplamalar** yapan thread'ler aynı veriyi kullanır farklı core'larda çalışır.

16

Konular

- Thread'ler
- Thread'lerin sağladığı faydalar
- 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

17

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 X ve Solaris gibi işletim sistemleri **kernel thread'leri destekler.**
- **Kernel thread'leri ile user thread'leri arasında aşağıdaki ilişkilendirme modellerinden birisinin oluşturulması zorundadır.**
 - Many-to-one model
 - One-to-one model
 - Many-to-many model

18

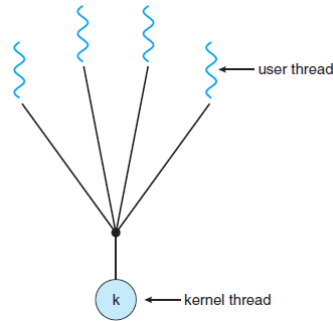
Konular

- Thread'ler
- Thread'lerin sağladığı faydalar
- 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

19

Many-to-one

- Many-to-one modelinde, **çok sayıda kullanıcı thread'i bir tane kernel thread'i ile eşleştirilir (Solaris işletim sistemi kullanır.)**.



- Eğer **bir thread sistem çağrısını bloklarsa** tüm process bloklanmış olur.
- Aynı anda sadece bir tane kullanıcı thread'i kernel thread'e erişebilir.
- Sadece bir kernel thread'i kullanıldığı için **multicore sistemlerde birden fazla thread için eşzamanlı çalışma yapılamaz.**

20

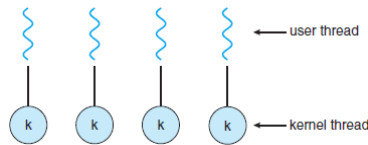
Konular

- Thread'ler
- Thread'lerin sağladığı faydalar
- 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

21

One-to-one

- One-to-one modelinde, **bir kullanıcı thread'i bir kernel thread'i ile eşleştirilir** (Linux, Windows işletim sistemleri kullanır.).



- Eğer **bir thread sistem çağırısını bloklarsa** diğer thread'ler çalışmasına devam eder.
- Birden fazla kernel thread'inin **multicore sistemlerde eşzamanlı çalışmasına izin verir.**
- Bir user thread için bir kernel thread oluşturulması gereklidir.

22

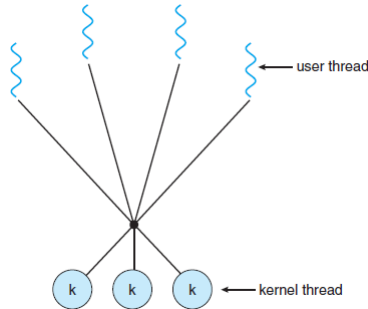
Konular

- Thread'ler
- Thread'lerin sağladığı faydalar
- 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

23

Many-to-many

- Many-to-many modelinde, **çok sayıda kullanıcı thread'i ile aynı sayıdaki veya daha az sayıdaki kernel thread'i eşleştirilir (Solaris 9, Unix işletim sistemleri kullanır.).**



- **Bir thread sistem çağrısını bloklarsa, kernel başka bir thread'i çalıştırır.**

24

Konular

- Thread'ler
- Thread'lerin sağladığı faydalar
- 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

25

Thread kütüphaneleri

- Thread kütüphanesi, **programcıya thread oluşturmak ve yönetmek için API sağlar.**
- Thread kütüphanesi oluşturulurken **iki farklı yaklaşım kullanılır:**
 - Tüm thread kütüphanesi **kullanıcı alanında oluşturulur** ve **kernel desteği yoktur.**
 - İşletim sisteminin doğrudan desteklediği **kernel seviyesinde kütüphane oluşturulur.**
- Çoklu thread oluşturmak için iki farklı strateji kullanılmaktadır:
 - **Asenkron threading: Parent**, yeni bir child thread oluşturduğunda **eşzamanlı olarak çalışmasını sürdürür.**
 - **Senkron threading: Parent**, child process oluşturduğunda **çalışmasını durdurur** ve tüm child process'ler sonlandığında çalışmasına devam eder (**fork-join strategy**).
- **Asenkron threading**, thread'ler arasında **veri paylaşımı az olduğunda**, **senkron threading** ise threadler arasında **veri paylaşımı çok olduğunda** kullanılır.

26

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.

27

Thread kütüphaneleri

Pthreads

- **Pthreads**, IEEE1003.1c standardıyla thread oluşturma ve yönetmek için tanımlanan API'dir.
- **Linux, Unix, Mac OS X ve Solaris** işletim sistemleri Pthreads standardını kullanır.
- **Windows** Pthreads standardını **desteklemez**.
- **Pthreads standardında thread'lerin hepsi ayrı fonksiyonlar halinde oluşturulur.**
- **Tüm thread'ler global scope'ta tanımlanan verileri paylaşırlar.**

28

Thread kütüphaneleri

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

Thread kütüphaneleri

Pthreads - Örnek

- Önceki örnekte bir thread oluşturulmuştur. 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.

Thread kütüphaneleri

Windows threads

- Windows thread kütüphanesi ile thread oluşturma Pthreads ile birçok açıdan benzerlik gösterir.
- **Thread'lerin hepsi ayrı fonksiyonlar halinde oluşturulur.**
- **Tüm thread'ler global scope'ta tanımlanan verileri paylaşırlar.**

31

Thread kütüphaneleri

Windows threads - Örnek

```
#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;
    }

    /* 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);
    }
}
```

Thread oluşturmak için kullanılan header file

Yeni thread için başlama noktası.

DWORD-32 bit unsigned int
LPVOID- Void için pointer

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

32

Thread kütüphaneleri

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 parameter passing ile yapılır.**
- Java ile iki farklı teknik kullanılarak thread oluşturulabilir:
 - Thread sınıfından yeni bir sınıf türetilir ve run() metodu override yapılır.
 - Runnable arayüzünü kullanan bir sınıf oluşturulur. (yaygın kullanılır.)

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

Aynı thread için override yapmak gereklidir.

33

Thread kütüphaneleri

Java threads – Örnek

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

Aynı thread için gereklidir.

Yeni thread oluşturuldu.

Yeni thread başladı.
run() çağırılır.

fork-join stratejisi
Thread'ler senkron çalışır.

34

Konular

- Thread'ler
- Thread'lerin sağladığı faydalar
- 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

35

Dolaylı thread oluşturma

- **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 tarafından yapılması** günümüzde giderek popüler hale gelmektedir.
- Bu stratejiye **implicit threading** denilmektedir.

36

Dolaylı thread oluşturma

Thread pools

- Multithreaded bir Web sunucu, gelen isteklerin her birisi için yeni thread oluşturur.
- Multithreaded bir Web sunucu, eşzamanlı çok sayıda istemciye servis sağlayabilir.
- **Gelen istek sayısı çok artarsa** sistem kaynakları (CPU time, memory, ...) tükenir.
- **Multithreaded sistemlerde belirli sayıda thread oluşturulmasına izin vermek için thread pool oluşturulur.**
- Yeni istek geldiğinde thread pool içerisinde kullanılabilir thread varsa **cevaplanır**, yoksa bir thread'in serbest hale gelmesi beklenir.
- **Varolan thread'in kullanımı yeni thread oluşturmaya göre daha hızlıdır.**

37

Dolaylı thread oluşturma

Thread pools - Windows

- Örnekte `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);
```

- **Parametreler:**

- `LPTHREAD_START_ROUTINE`, thread olarak çalışacak fonksiyonun pointer'ı
- `PVOID`, Gönderilecek parametre
- `ULONG`, Bayrak bitleri (bekleme süresi, I/O gerekliliği, ...)

38

Dolaylı thread oluşturma

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.

39

Dolaylı thread oluşturma

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() deyimini paralel çalışacaktır.

40

Dolaylı thread oluşturma

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.**
- **Concurrent queue**, FIFO çalışır ve kuyruktan **birden fazla blok aynı anda alınabilir.**

41

Dolaylı thread oluşturma

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

42

Konular

- Thread'ler
- Thread'lerin sağladığı faydalar
- 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

43

Thread çalıştırma kuralları

fork() ve exec() sistem çağrıları

- Bazı Unix sistemlerde, iki tür fork() çağrısı vardır.
- Birisi ile tüm thread'ler duplicate yapılır, diğeri ile sadece fork() ile başlatılan thread duplicate yapılır.
- exec() sistem çağrısı ile tüm thread'leri ile birlikte process duplicate yapılır.

44

Thread çalıştırma kuralları

Signal handling

- Unix sistemlerde bir **signal** belirli bir olayın gerçekleştiğini gösterir.
- **Oluşan olaya karşılık gelen sinyal bir process'e iletilir.**
- Oluşan sinyal, **senkron** veya **asenkron** alınabilir.
- **Senkron sinyal, sinyalin oluşmasına neden olan olayı gerçekleştiren process'e iletilir.**
- Senkron sinyale illegal hafıza erişimi veya 0'a bölme verilebilir.
- **Asenkron sinyal, sinyali oluşturan process'ten başka bir process'e iletilir.**
- Asenkron sinyale <ctrl><C> tuşlarına birlikte basmak verilebilir.

45

Thread çalıştırma kuralları

Signal handling

- İşletim sistemlerinde **sinyaller farklı hedeflere gönderilebilir:**
 - Process içerisindeki **sadece bir thread'e gönderilebilir** (Senkron).
 - Process içerisindeki **tüm thread'lere gönderilebilir** <ctrl><C>.
 - Process içerisindeki **bazı thread'lere gönderilebilir.**
 - Bir process için **tüm sinyalleri almak üzere bir thread atanabilir.**
- **Senkron sinyaller sadece oluşturan thread'e gönderilir.**
- Aşağıdaki UNIX fonksiyonu ile ID değeri verilen process'e iletilir.

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

- POSIX Pthreads ile aşağıdaki fonksiyon kullanılır.

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

46

Thread çalıştırma kuralları

Thread iptal etme

- Bir thread'in çalışması tamamlanmadan iptal edilebilir.
- İstenen bir **sonucun bir thread tarafından bulunması halinde diğerleri iptal edilebilir.**
- Bir **Web sayfası yüklenirken stop butonuna basıldığında** process içerisindeki tüm thread'ler iptal edilir.
- Bir thread başka bir thread'i aniden sonlandırabilir (**Asenkron cancellation**).
- Bir thread başka bir thread'in kendi kendisini sonlandırmasını sağlayabilir (**Deferred cancellation**).

47

Thread çalıştırma kuralları

Thread iptal etme

- Pthreads aşağıdaki tabloda verilen üç farklı iptal etme modunu destekler.

```
pthread_t tid;

/* create the thread */
pthread_create(&tid, 0, worker, NULL);

. . .

/* cancel the thread */
pthread_cancel(tid);
```

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

48

Konular

- Thread'ler
- Thread'lerin sağladığı faydalar
- 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**

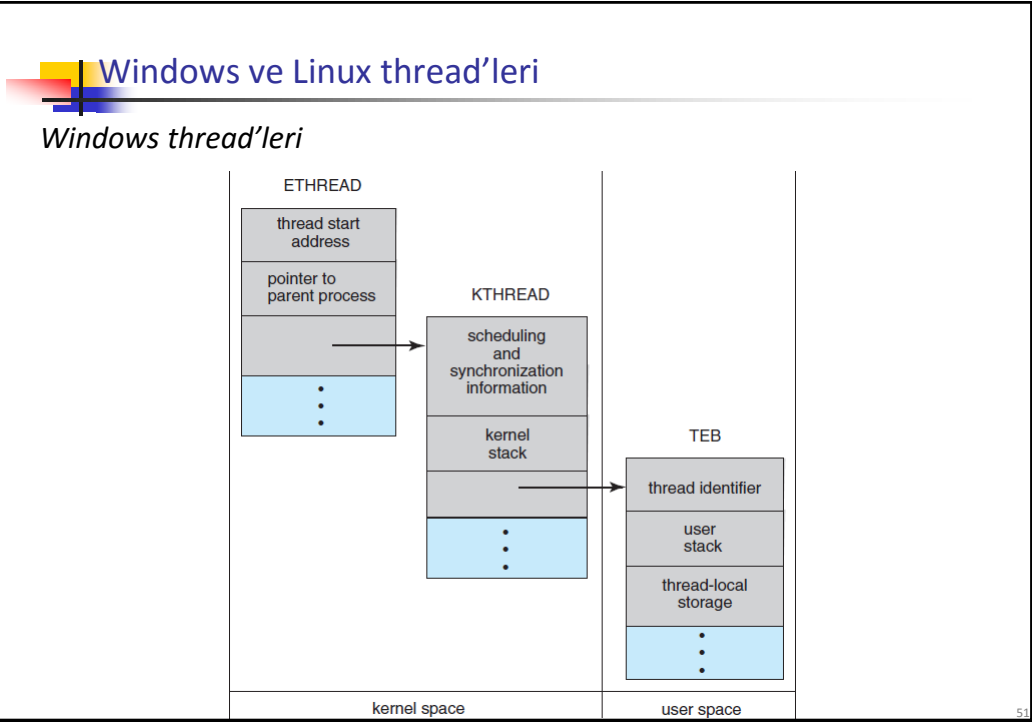
49

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.**
- Bir thread için ayrılan **register kümesi, stack, depolama alanı, context** olarak adlandırılır.
- Windows bir thread için aşağıdaki veri yapılarını kullanır:
 - **ETHREAD:** Yürütücü thread blok
 - **KTHREAD:** Kernel thread blok
 - **TEB:** Thread environment (ortam) blok

50



Windows ve Linux thread'leri

Linux thread'leri

- Linux, `fork()` sistem çağrısının yanı sıra `clone()` sistem çağrısı ile thread oluşturabilir.
- Linux, `clone()` ile yeni bir görev başlattığında, parent task ile child task arasında paylaşım miktarını da gönderir.
- **Dosya sistemi, hafıza aralığı, sinyaller veya açık olan dosyalar paylaştırılabilir.**
- Görevler, Linux kernel içerisinde bir veri yapısına sahiptir (`struct task_struct`) ve açık dosyalar, virtual memory ve sinyal bilgilerini gösterir.
- Linux, `fork()` ile **yeni bir görev başlattığında, parent task veri yapısı kopyalanır.**

52

Windows ve Linux thread'leri

Linux thread'leri

- Linux, clone() ile yeni bir görev başlattığında bayrak bitlerine göre veri yapısı oluşturulur.

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.