Bölüm 2: Süreçler

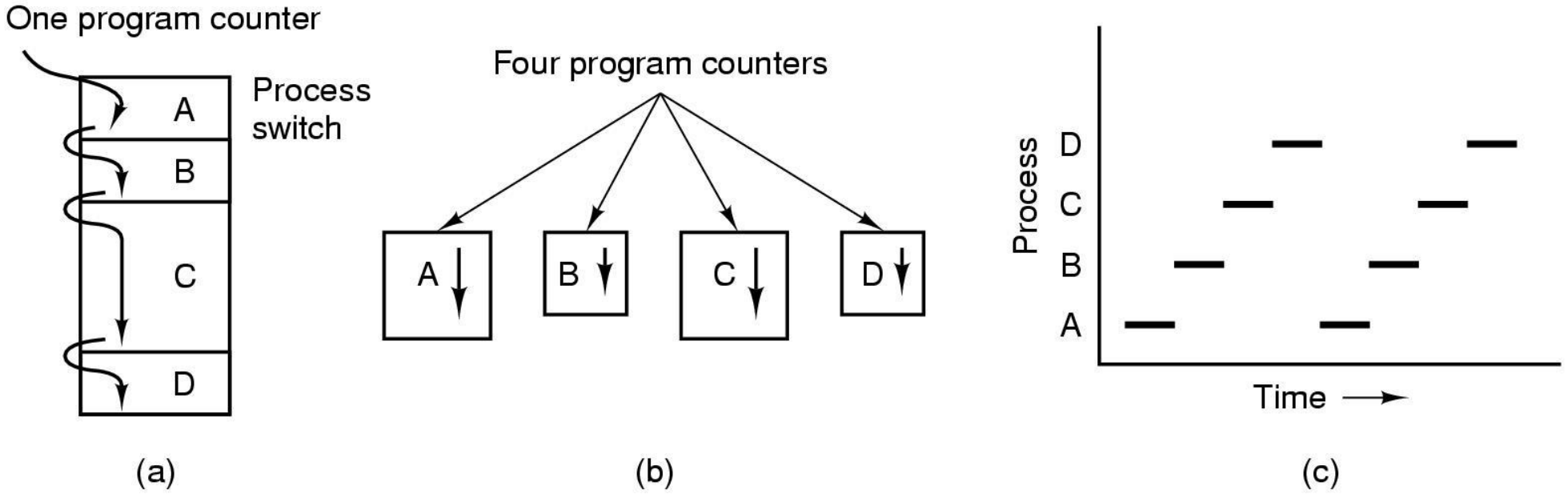
İşletim Sistemleri

Sözde Paralellik

- Tüm modern bilgisayarlar aynı anda birçok iş yapar.
- Tek işlemcili bir sistemde, herhangi bir anda, işlemci sadece bir işlem yürütebilir.
- Ancak çoklu programlama sisteminde işlemci, her biri onlarca veya yüzlerce ms boyunca çalışan işlemler arasında hızlıca geçiş yapar.
- Sözde paralellik kullanıcılar için çok faydalıdır. Ancak; yönetimi bir o kadar zordur.

Süreç Modeli

(a) Dört programın çoklu programlanması. (b) Birbirinden bağımsız dört ardışık sürecin kavramsal modeli. (c) Aynı anda bir program etkindir.



Tekrarlanamaz Yürütme

- Non-reproducible

Program 1: repeat n = n + 1;

Program 2: repeat print(n); n = 0;

Yürütme sırası farklı olabilir.

- n = n + 1; print(n); n = 0;
- print(n); n = 0; n = n + 1;
- print(n); n = n + 1; n = 0;

Süreç ve Program Arasındaki Farklar

- Program, bilgisayar kodlarının bir koleksiyonudur ve çalıştırılabilir bir dosya halindedir.
- Bir program, bir süreç oluşturulduğunda çalıştırılır.
- Süreçler bellekte yer kaplar.
- Bir program birden fazla süreç oluşturabilir ve her süreç ayrı sistem kaynakları kullanır.
- Süreçler arasında haberleşme, veri paylaşımı ve iş bölümü gerçekleşebilir.

Süreç Başlatma

Süreç oluşturmaya neden olan olaylar:

- Sistem başlatma.
- Çalışan bir süreç tarafından bir süreç oluşturma sistem çağrısının yürütülmesi.
- Yeni bir süreç oluşturmak için bir kullanıcı isteği.
- Toplu işin başlatılması. (batch)

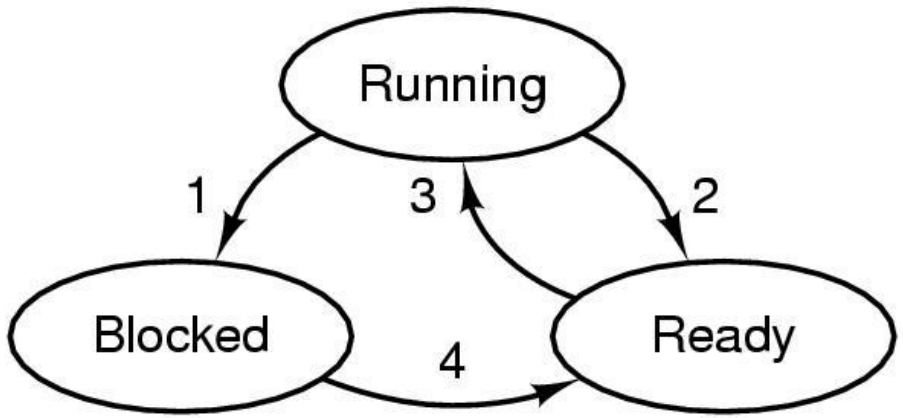
Süreç Sonlandırma

İşlemin sonlandırılmasına neden olan olaylar:

- Normal çıkış (gönüllü).
- Hata sonrası çıkış (gönüllü).
- Ölümcül hata sonrası çıkış (istem dışı).
- Başka bir süreç tarafından sonlandırılma (kill) (istemsiz).

Süreç Durumları

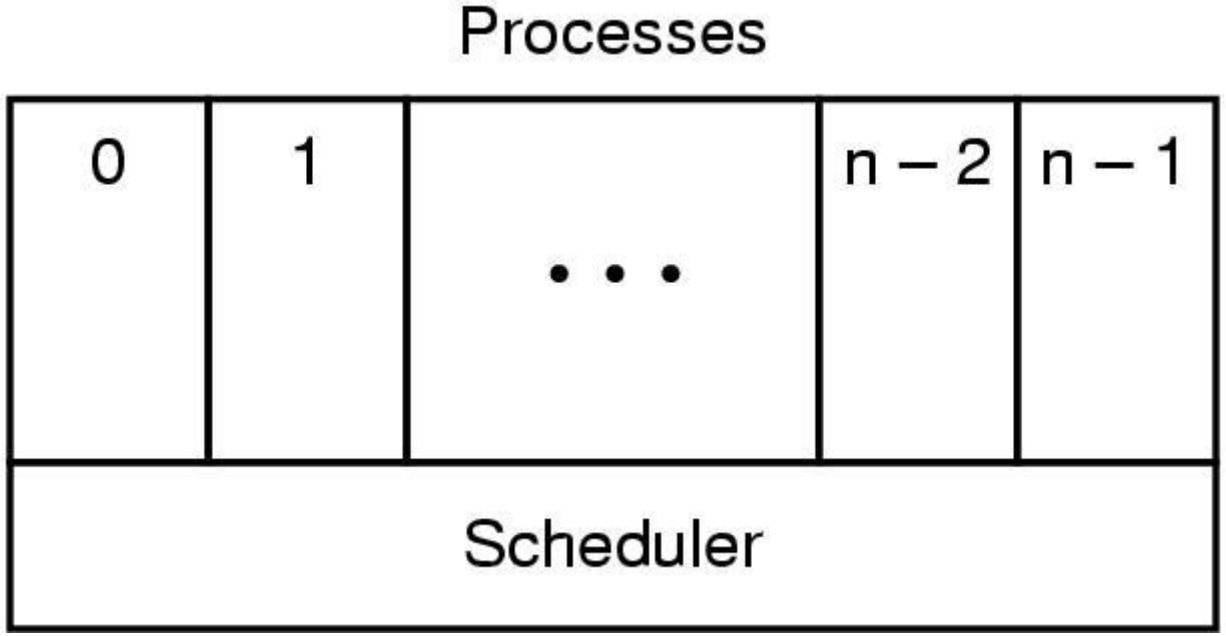
Bir süreç çalışıyor, engellenmiş veya hazır durumda olabilir.



1. Process blocks for input
2. Scheduler picks another process
3. Scheduler picks this process
4. Input becomes available

Süreçleri Gerçekleştirme

Süreç yapılı bir işletim sisteminin en alt katmanı kesilmeleri ve çizelgelemeyi yönetir. Bu katmanın üzerinde sıralı süreçler bulunur.



Süreçleri Gerçekleştirme

Süreç tablosunda bulunan bazı alanlar.

Process management	**Memory management**	**File management**
Registers	Pointer to text segment info	Root directory
Program counter	Pointer to data segment info	Working directory
Program status word	Pointer to stack segment info	File descriptors
Stack pointer		User ID
Process state		Group ID
Priority		
Scheduling parameters		
Process ID		
Parent process		
Process group		
Signals		
Time when process started		
CPU time used		
Children's CPU time		
Time of next alarm		

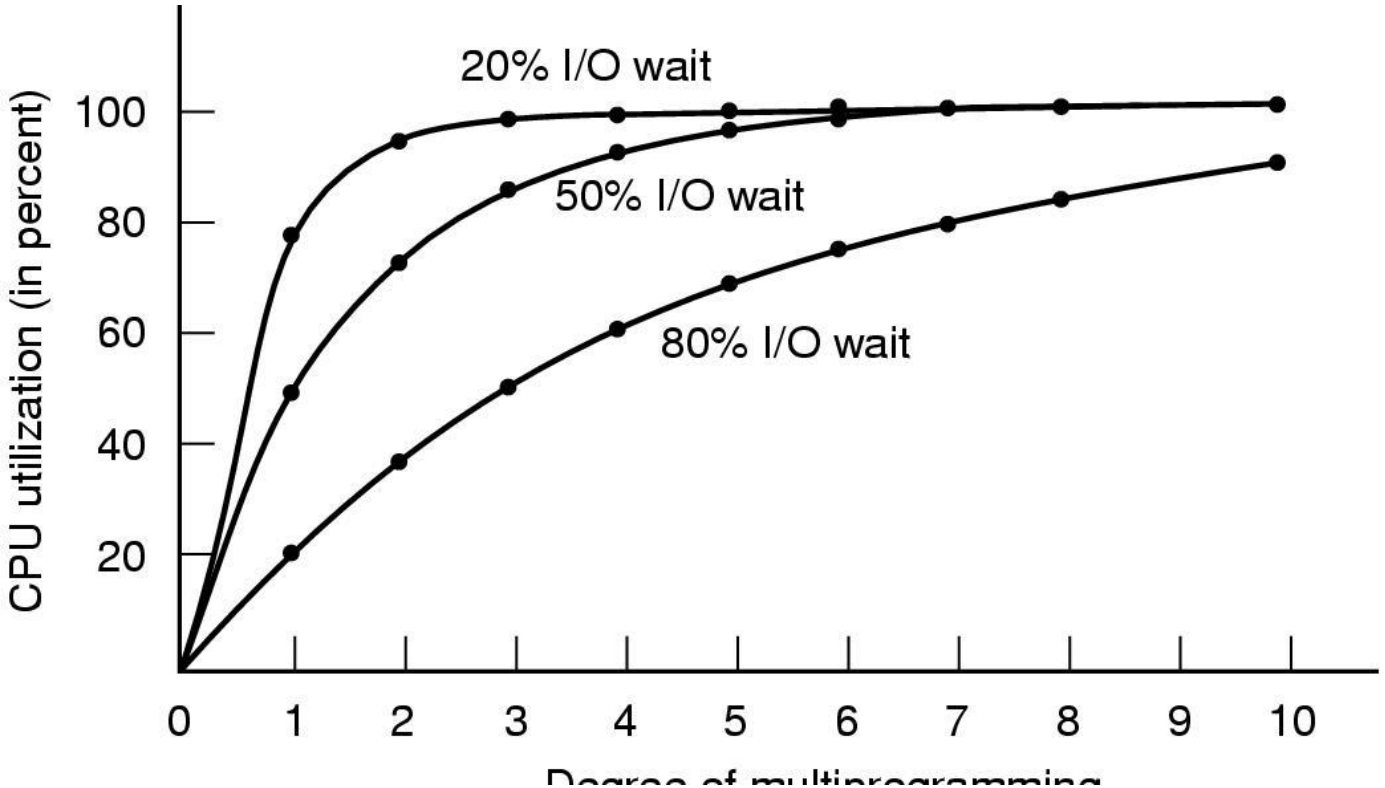
Süreçleri Gerçekleştirme

Bir kesilme oluştuğunda işletim sisteminin en düşük seviyesi ne yapar.

1. Hardware stacks program counter, etc.
2. Hardware loads new program counter from interrupt vector.
3. Assembly language procedure saves registers.
4. Assembly language procedure sets up new stack.
5. C interrupt service runs (typically reads and buffers input).
6. Scheduler decides which process is to run next.
7. C procedure returns to the assembly code.
8. Assembly language procedure starts up new current process.

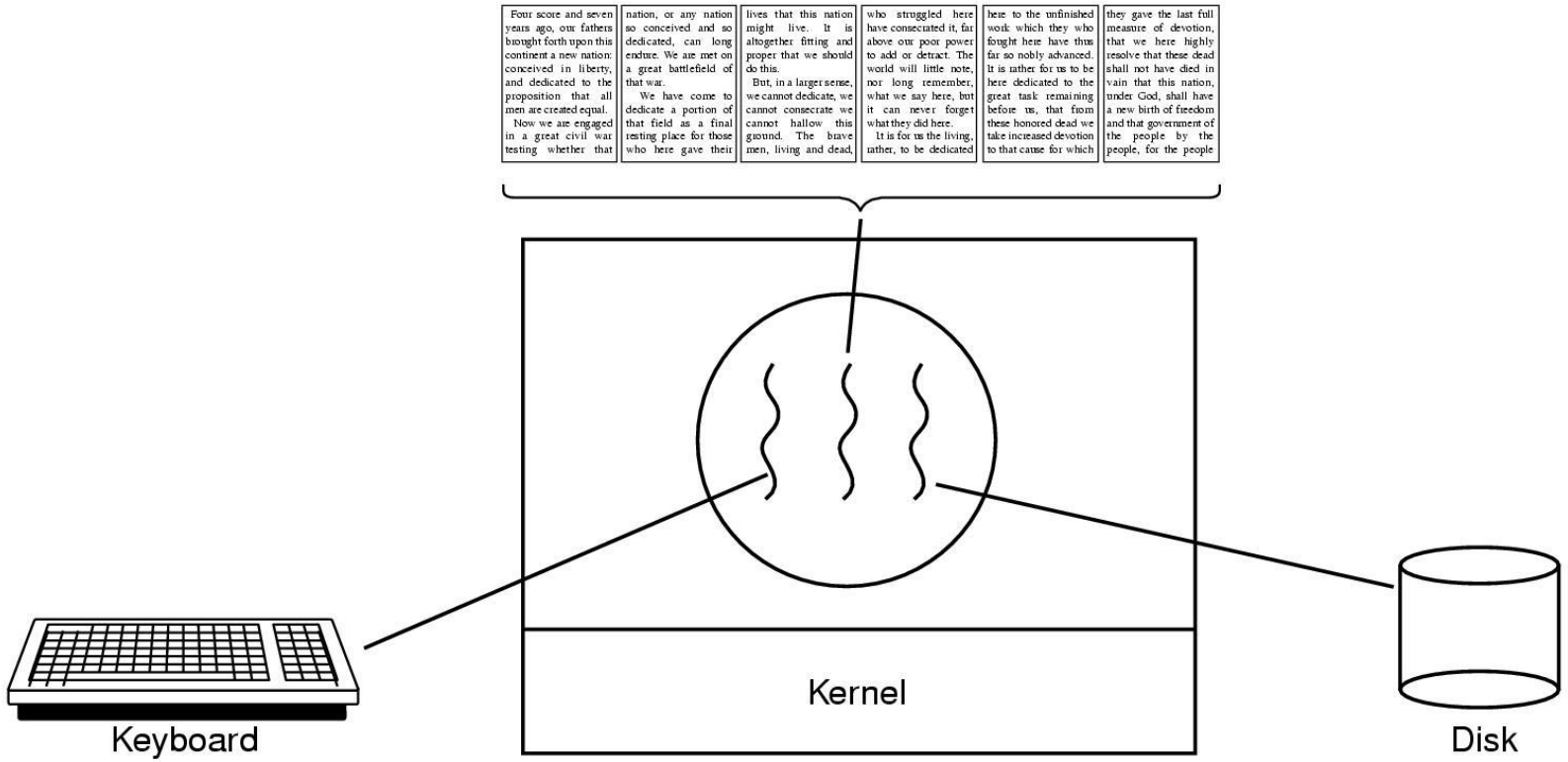
Çoklu Programlama Modellemesi

Bellekte bulunan süreç sayısının bir fonksiyonu olarak CPU kullanımı grafiği.



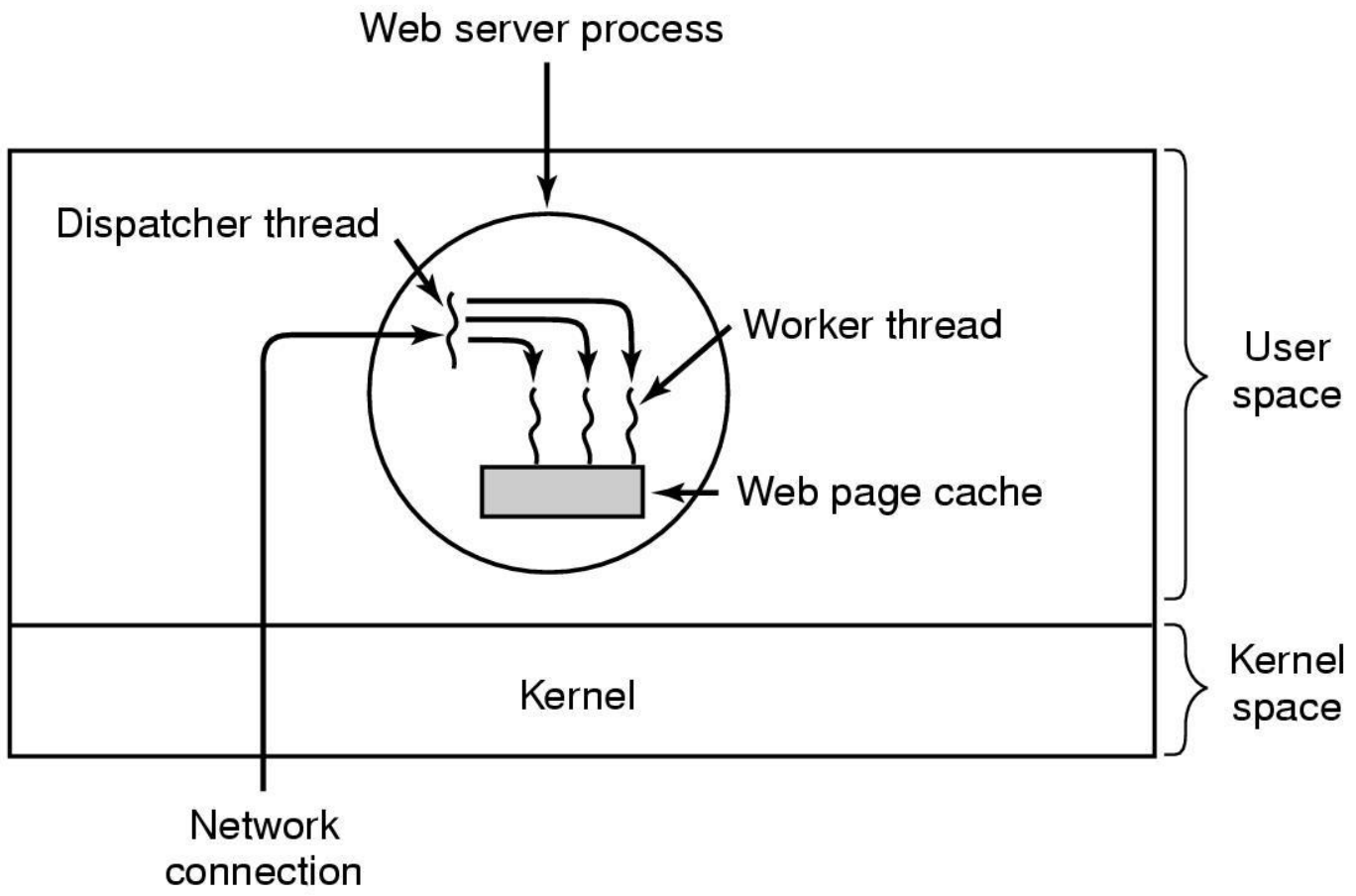
İş Parçacığı

3 iş parçacığına sahip bir uygulama



İş Parçacığı Kullanımı

Çoklu iş parçacığına sahip bir web sunucusu



İş Parçacığı Kullanımı

(a) İşlemci zamanlayıcı (dispatcher) iş parçacığı

(b) İşçi (worker) iş parçacığı

```
while (TRUE) {
  get_next_request(&buf);
  handoff_work(&buf);
}
```

(a)

```
while (TRUE) {
  wait_for_work(&buf)
  look_for_page_in_cache(&buf, &page);
  if (page_not_in_cache(&page)
      read_page_from_disk(&buf, &page);
  return_page(&page);
}
```

(b)

İş Parçacığı Modeli

- Süreç içerisindeki tüm iş parçacıkları ile paylaşılanlar
- Her bir iş parçacığına özel veriler

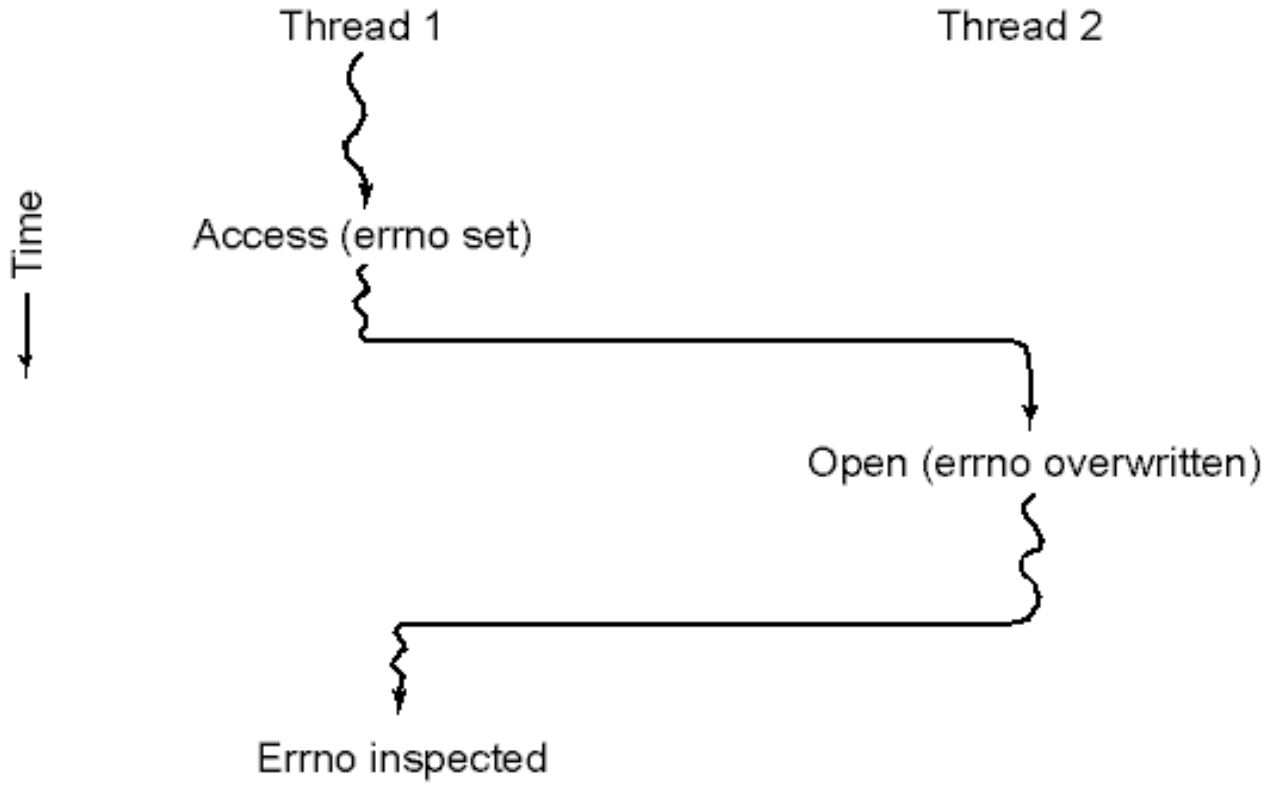
Per process items	**Per thread items**
Address space	Program counter
Global variables	Registers
Open files	Stack
Child processes	State
Pending alarms	
Signals and signal handlers	
Accounting information	

İş Parçacığı

- Kendi program sayacı, yazmaç kümesi ve yığını vardır
- Kod (text), global veri ve açık dosyaları paylaşır
  - Aynı süreci sonlandırmak için paralel çalıştığı iş parçacıkları ile
- Kendi süreç kontrol bloğuna (PCB) sahip olabilir
  - İşletim sistemine bağlıdır
  - Bağlam, iş parçacığı kimliğini, program sayacını, kayıt kümesini, yığın işaretçisini içerir
  - Aynı süreçteki diğer iş parçacıklarıyla bellek adres uzayı paylaşılır
    - bellek yönetimi bilgileri paylaşılır

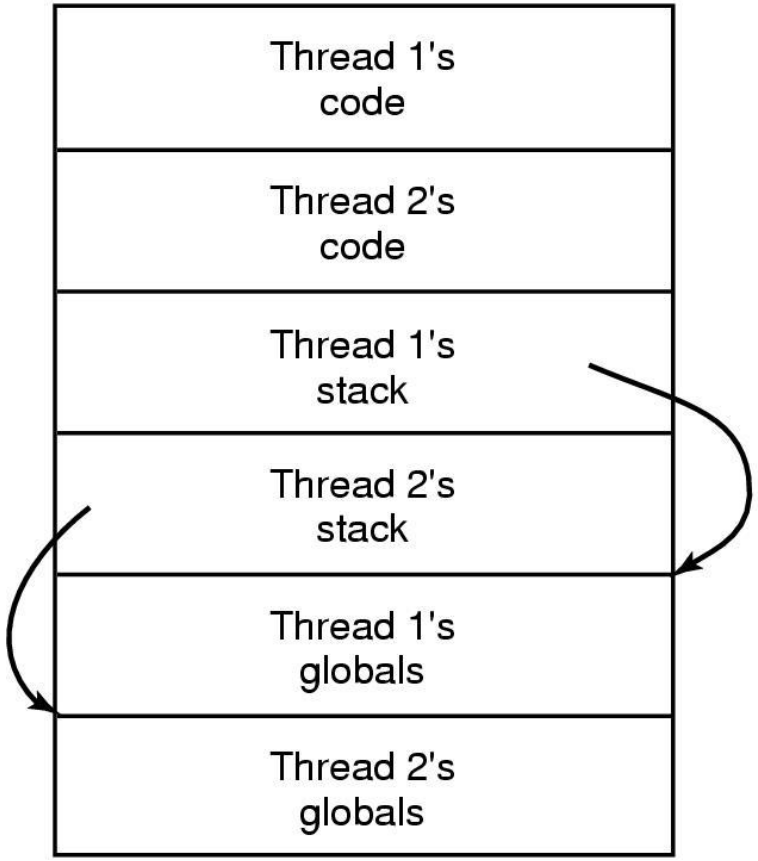
İş Parçacıkları Arasında Çakışma

Global bir değişkenin kullanımıyla ilgili iş parçacıkları arasında yaşanabilecek çakışma



Çoklu İş Parçacıklı Programlama

İş parçacıkları kendilerine ait global değişkenlere sahip olabilir



İş Parçacıklarının Avantajları

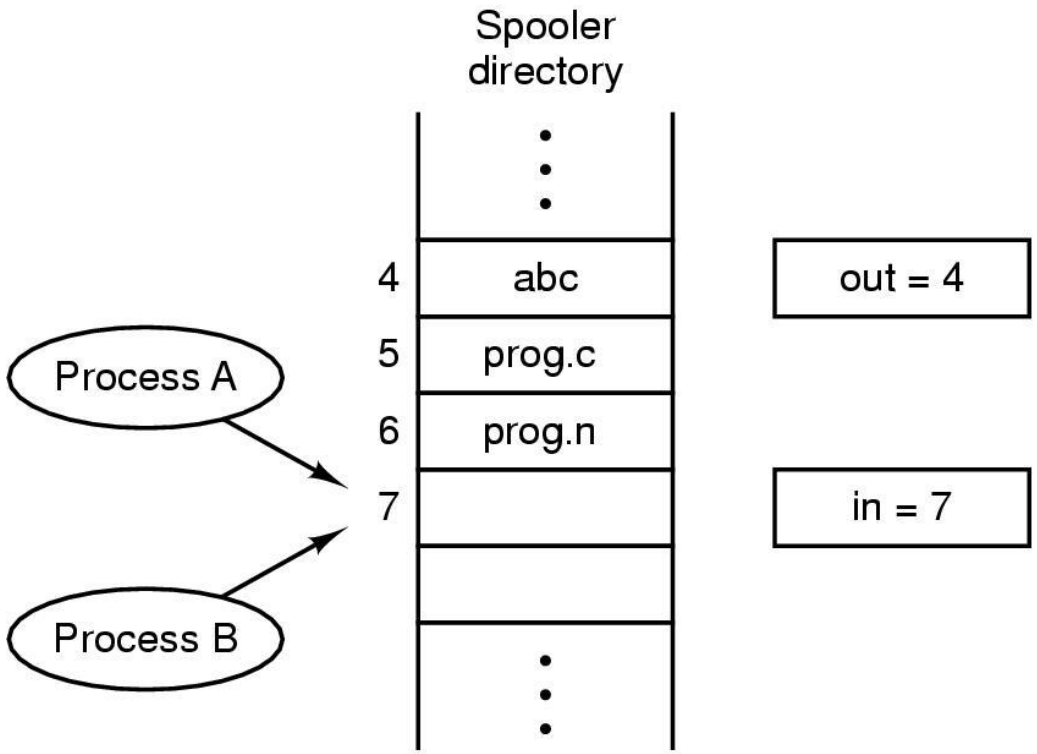
- Kullanıcı duyarlılığı
  - Bir iş parçacığı bloke olduğunda, diğeri kullanıcı G/Ç'sini işleyebilir. Ancak: iş parçacığı uygulamasına bağlı
- Kaynak paylaşımı: ekonomi
  - Bellek paylaşılır (yani adres alanı paylaşılır), Açık dosyalar, soketler
- Hız
  - İş parçacığı oluşturma süreç oluşturmaya göre yaklaşık 30 kat daha hızlı, bağlam geçişi 5 kat daha hızlı
- Donanım paralelliğinden yararlanma
  - Ağır süreçler, çoklu işlemcili mimarilerden de faydalanabilir

İş Parçacıklarının Dezavantajları

- Senkronizasyon
  - Paylaşımlı bellek ve değişkenlere erişim kontrol edilmelidir.
  - Program koduna karmaşıklık, hatalar ekleyebilir. Yarış koşullarından, kilitlenmelerden ve diğer sorunlardan kaçınmak gerekir
- Bağımsızlık eksikliği
  - Ağır Ağırlık İşlemde (HWP) iş parçacıkları bağımsız değildir
  - RAM adres uzayı paylaşıldığından bellek koruması yoktur
  - Her iş parçacığının yığınları bellekte ayrı yerde olması amaçlanır, ancak bir iş parçacığının hatası nedeniyle başka bir iş parçacığının yığınının üzerine yazma yapılabilir.

Süreçler Arası İletişim

- Yarış durumu: iki süreç aynı bellek alanına aynı anda erişmek istediğinde



Yarış Durumu

- İki veya daha fazla süreç, bazı paylaşılan verileri okuyor veya yazıyor ve nihai sonuç hangisinin ne zaman çalıştığına bağlı.

- Karşılıklı dışlama
  - Birden fazla işlemin paylaşılan verileri aynı anda okumasını ve yazmasını engelleme
- Kritik bölge
  - Programın paylaşılan alana erişim yaptığı kod bölümü

Karşılıklı Dışlama

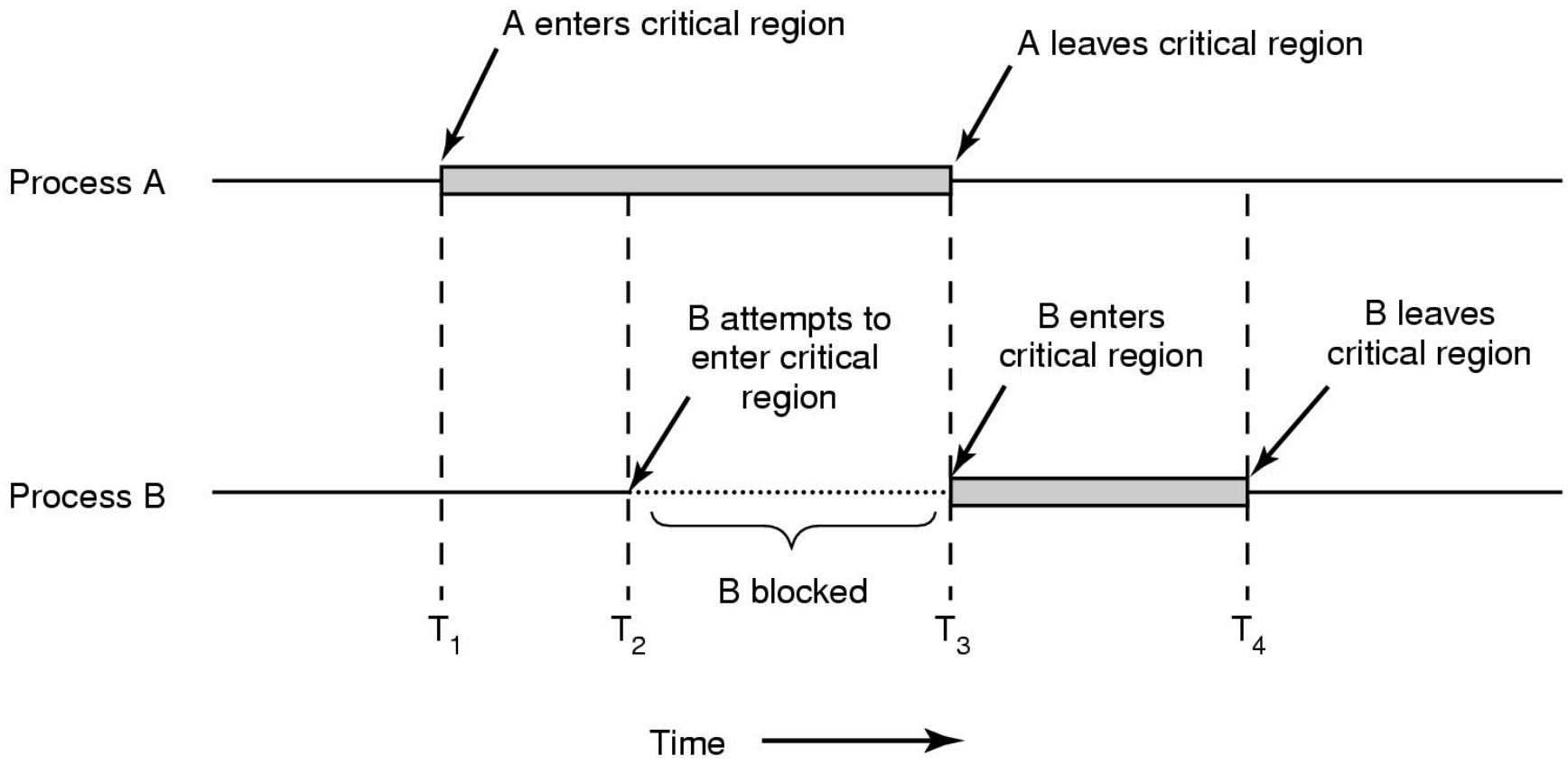
Karşılıklı dışlama sağlamak için dört koşul

- İki süreç aynı anda kritik bölgede olmamalı
- İşlemci hızı ve sayısı hakkında varsayım yapılmamalı
- Kritik bölgesinin dışında çalışan hiçbir süreç başka bir süreci engellememeli
- Hiçbir süreç kritik bölgesine girmek için sonsuza kadar beklememeli

Kritik Bölge

```
do {
    entry section
        critical section
    exit section
        remainder section
} while (TRUE);
```

Kritik Bölge Kullanarak Karşılıklı Dışlama



Meşgul Bekleme ile Karşılıklı Dışlama

- Kesmeleri devre dışı bırakma
  - Kritik bölgeye girdikten sonra tüm kesmeleri devre dışı bırak
  - Clock yalnızca bir kesme olduğundan, hiçbir CPU önalımı (preemption) gerçekleşemez.
  - Kesmeleri devre dışı bırakmak, işletim sisteminin kendisi için yararlıdır, ancak kullanıcılar için değildir

Meşgul Bekleme ile Karşılıklı Dışlama

- Lock değişkeni
  - Yazılımsal bir çözüm
  - Tek ve paylaşımlı bir değişen (lock) tanımlanır
  - Kritik bölgeye girmeden önce değeri kontrol edilir
  - Değer 0 ise kritik bölgeye girilmez, beklenir
  - Değer 1 ise kritik bölgeye girilir.

- Sorun ne?

Önerilen Çözüm

```
while (TRUE) {
    while (turn != 0)        /* loop */ ;
    critical_region( );
    turn = 1;
    noncritical_region( );
}
```

(a)

```
while (TRUE) {
    while (turn != 1)        /* loop */ ;
    critical_region( );
    turn = 0;
    noncritical_region( );
}
```

(b)

Kavramlar

- Busy waiting
  - Bir değere ulaşana kadar bir değişkeni sürekli olarak test etme
- Spin lock
  - Meşgul beklemeyi kullanan bir kilit, döndürme kilidi olarak adlandırılır

Peterson'un Çözümü

```
#define FALSE  0
#define TRUE   1
#define N      2                        /* number of processes */

int turn;                               /* whose turn is it? */
int interested[N];                      /* all values initially 0 (FALSE) */

void enter_region(int process);         /* process is 0 or 1 */
{
    int other;                          /* number of the other process */

    other = 1 – process;                /* the opposite of process */
    interested[process] = TRUE;         /* show that you are interested */
    turn = process;                     /* set flag */
    while (turn == process && interested[other] == TRUE) /* null statement */ ;
}

void leave_region(int process)          /* process: who is leaving */
{
    interested[process] = FALSE;        /* indicate departure from critical region */
}
```

TSL Komutu

```
enter_region:
        TSL REGISTER,LOCK                | copy lock to register and set lock to 1
        CMP REGISTER,#0                  | was lock zero?
        JNE enter_region                 | if it was non zero, lock was set, so loop
        RET | return to caller; critical region entered

leave_region:
        MOVE LOCK,#0                     | store a 0 in lock
        RET | return to caller
```

Uyuma ve Uyanma

- Meşgul beklemenin dezavantajı
  - Düşük öncelikli bir süreç kritik bölgede iken,
  - Yüksek öncelikli bir süreç geldiğinde daha düşük öncelikli süreci engeller,
  - Lock’tan dolayı meşgul beklemede CPU'yu boşa harcar,
  - Daha düşük öncelikli süreç kritik bölge dışına çıkamaz
  - Öncelikleri değiştirmek/ölümcül kilitlenme

- Meşgul beklemek yerine block
  - Önce Uyandır, sonra uyut (wake up, sleep)

Üretici Tüketici Problemi

- İki işlem ortak, sabit boyutlu bir arabelleği paylaşmakta
- Üretici arabelleğe veri yazar
- Tüketici arabellekten veri okur

- Basit bir çözüm

Ölümcül Yarış Durumu

```
#define N 100                                   /* number of slots in the buffer */
int count = 0;                                  /* number of items in the buffer */

void producer(void)
{
     int item;

     while (TRUE) {                             /* repeat forever */
          item = produce_item( );               /* generate next item */
          if (count == N) sleep( );             /* if buffer is full, go to sleep */
          insert_item(item);                    /* put item in buffer */
          count = count + 1;                    /* increment count of items in buffer */
          if (count == 1) wakeup(consumer);     /* was buffer empty? */
     }
}


void consumer(void)
{
     int item;

     while (TRUE) {                             /* repeat forever */
          if (count == 0) sleep( );             /* if buffer is empty, got to sleep */
          item = remove_item( );                /* take item out of buffer */
          count = count – 1;                    /* decrement count of items in buffer */
          if (count == N – 1) wakeup(producer); /* was buffer full? */
          consume_item(item);                   /* print item */
     }
}
```

Veri Kaybı Sorunu

- Paylaşılan değişken: sayaç
- Eşzamanlılıktan kaynaklanan sorun
- Tüketici 0 ile sayaç değişkenini okuduğunda; ancak zamanında uykuya geçmediğinde, sinyal kaybolacaktır.

Semafor

- Dijkstra tarafından önerilen yeni bir değişken türü
- Atomik Eylem, tek ve bölünmez
- Down (P)
  - semafor kontrol edilir, değeri 0 ise uyku, değilse değeri azalt ve devam et
- Up (v)
  - semafor kontrol edilir,
  - Süreçler semaforda bekliyorsa, işletim sistemi devam etmeyi seçecek ve düşüşünü tamamlayacaktır.
  - Kaynak sayısının bir işareti olarak farz edilir

Tekrarlanamaz Yürütme

SON