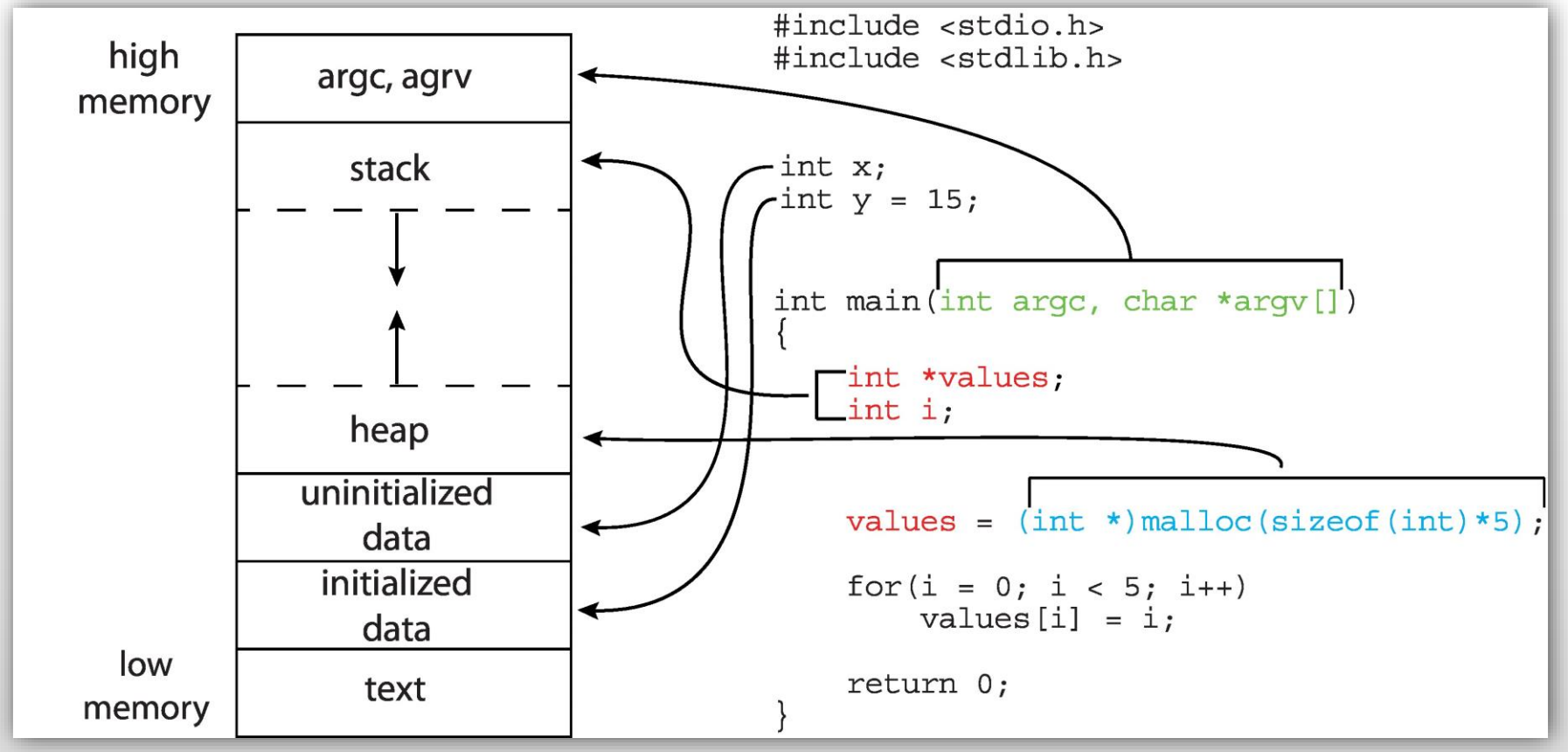
Bölüm 3: Süreçler

İşletim Sistemleri

Süreç

- Yürütülmekte olan programlardır
- Program belleğe yüklendiğinde süreç halini alır
- Programlar pasif, süreçler aktif
- Bir süreç sıralı bir şekilde yürütülür, paralel yürütülemez
- **Metin (text)** bölümü, program kodunu tutar
- **Program sayacı**, mevcut etkinliği tutan işlemci yazmacı
- **Yığıt (stack),** geçici verileri tutar
  - Fonksiyon parametreleri, dönüş adresleri, yerel değişkenler
- **Veri (data)** bölümü, genel değişkenleri içerir
- **Yığın (heap)**, çalışma süresi boyunca dinamik olarak ayrılan belleği içerir

Bellekte Süreç Yerleşimi

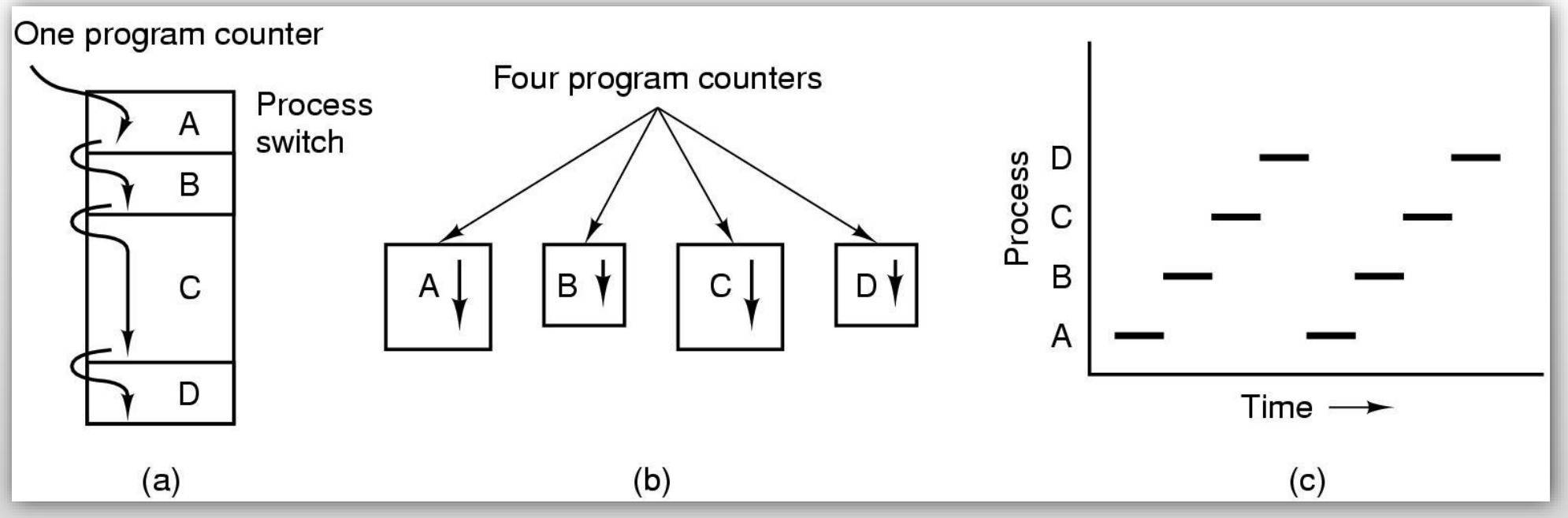


Sözde Paralellik

- Modern bilgisayarlardan aynı anda birçok işlem yürütmesi beklenir.
- Tek işlemcili bir sistemde, herhangi bir anda, sadece bir işlem yürütülebilir.
- 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.

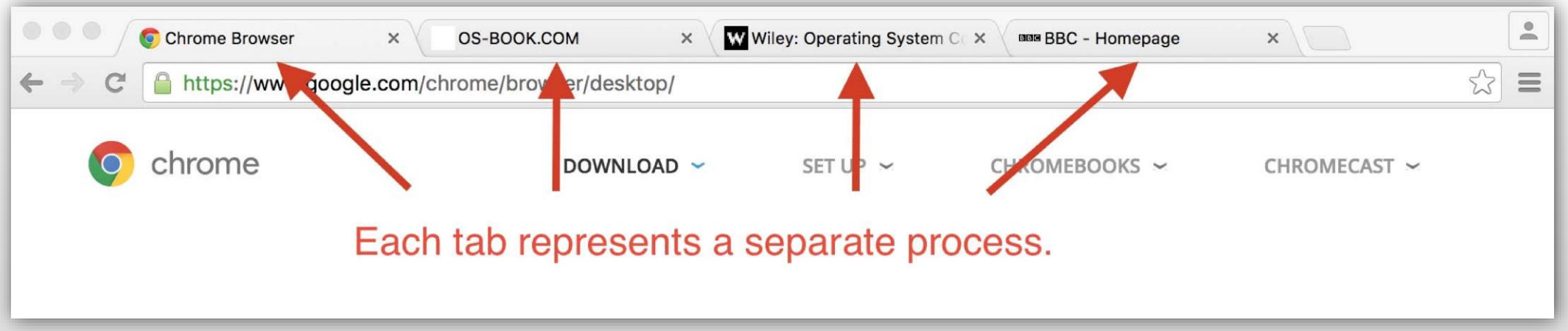
Çoklu Programlama Süreç Modeli

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



Chrome

- Browser, kullanıcı arayüzü, disk ve ağ arayüzlerini yönetir
- Renderer, web sayfası, HTML, javascript kodları ile ilgilenir.
- Plug-in, her bir plug-in için.



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 komutlarının bir koleksiyonudur
- Çalıştırılabilir dosya halindedir.
- Program çalıştırıldığında, bir süreç oluşturulur.
- Süreçler bellekte yer kaplar.
- Bir programdan birden fazla süreç oluşturulabilir
- Her süreç ayrı sistem kaynakları kullanır.
- Süreçler arasında haberleşme, veri paylaşımı ve iş bölümü gerçekleşebilir.

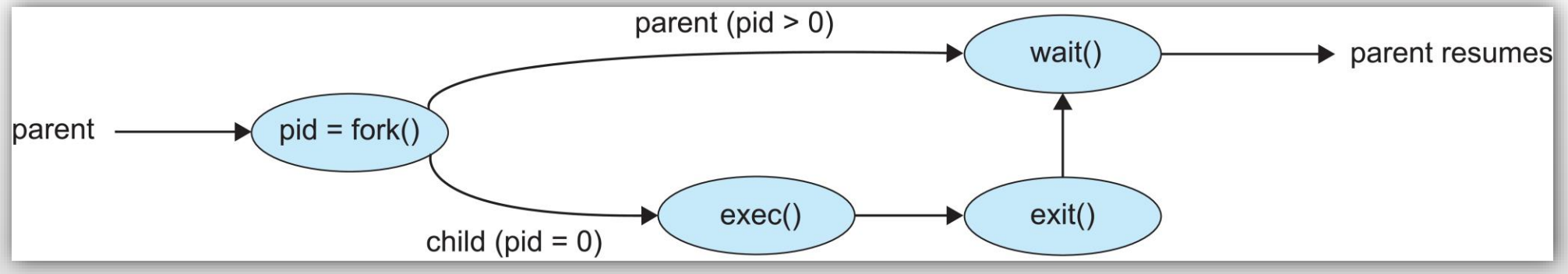
Süreç Başlatma

Süreç oluşturmaya neden olan olaylar:

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

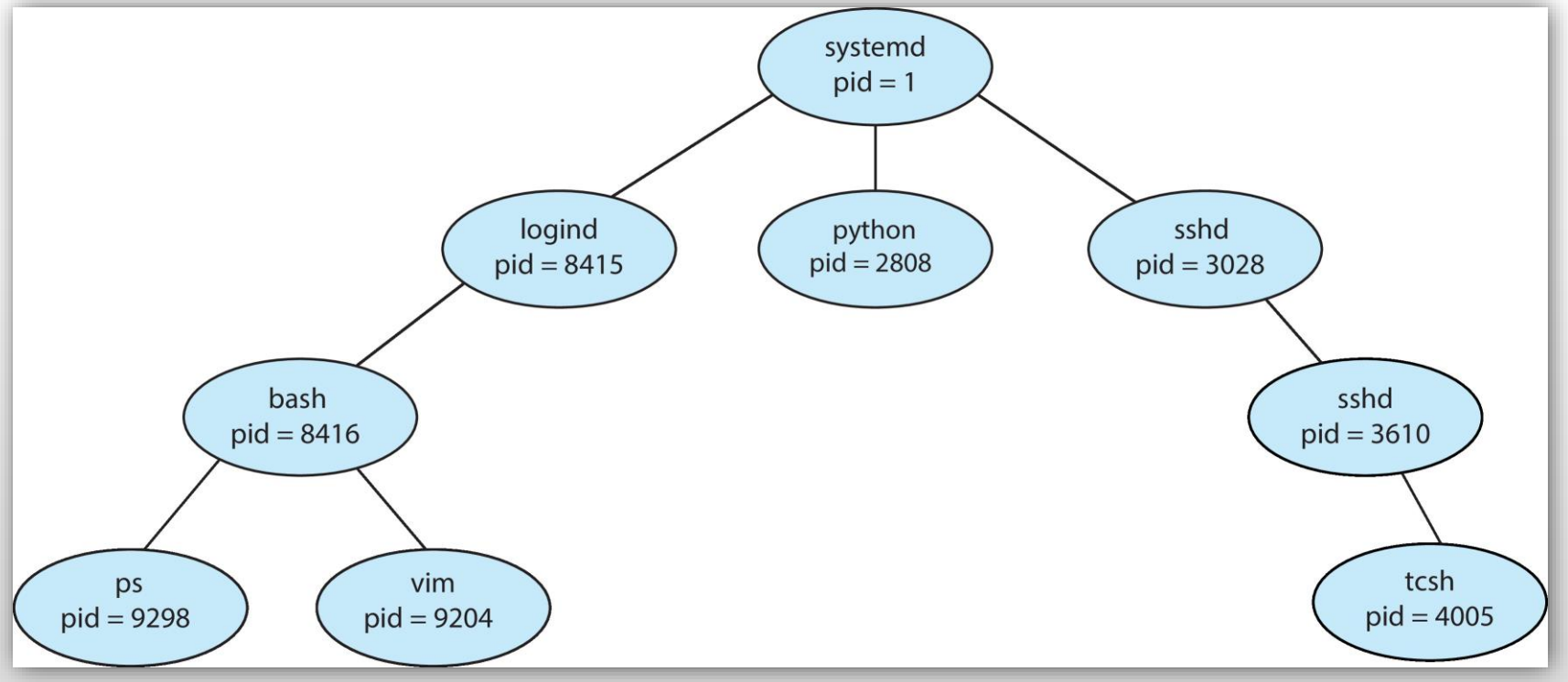
Süreç Başlatma

- Ebeveyn süreç, çocuk süreç başlatabilir. Tüm kaynakları paylaşabilirler. Paralel olarak çalışırlar. Ebeveyn çocuk sürecin sonlanmasını bekler. Çocuk süreç, ebeveyn sürecin adres uzayını kullanır.



Süreç Başlatma

- .



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).

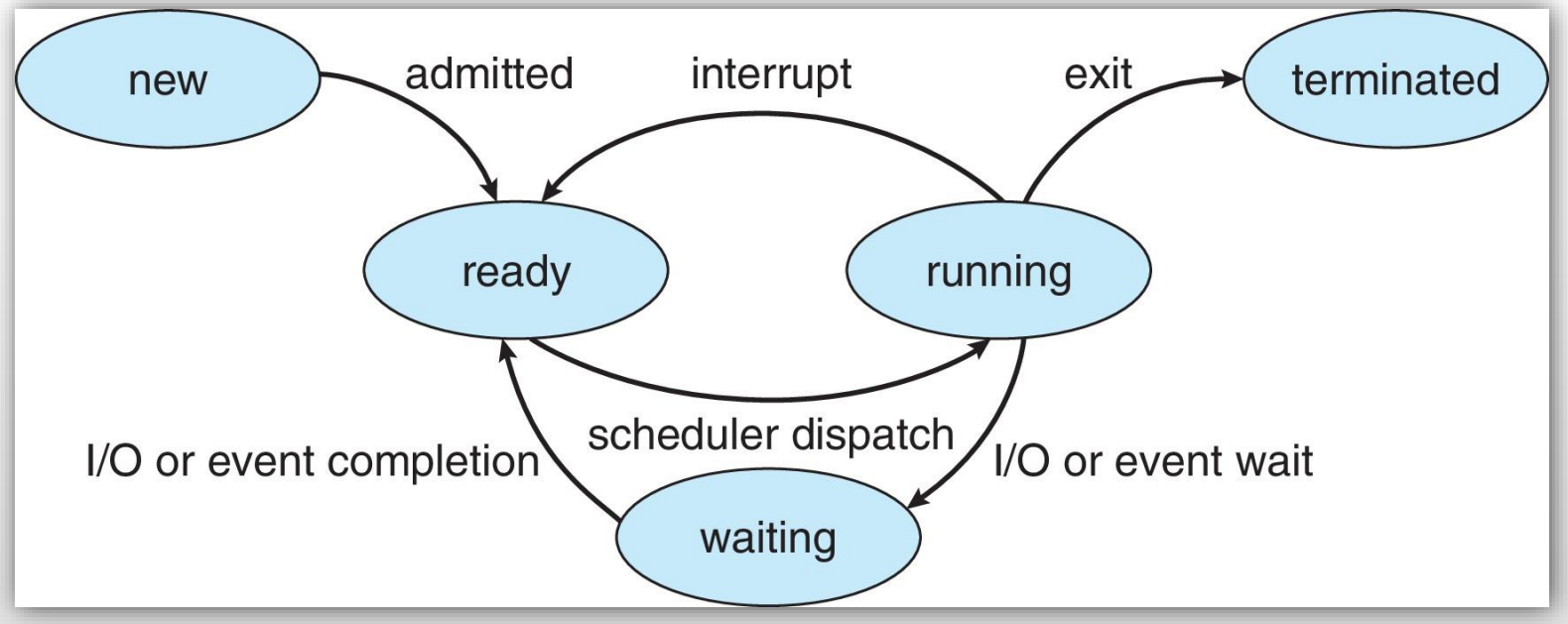
Süreç Sonlandırma

- Sürecin sonlanmasını bekleyen bir ebeveyn süreç yoksa (`wait()` çağrılmamışsa) **zombie** olarak adlandırılır
- Ebeveyn süreç `wait()` çağırmadan sonlanmışsa, **orphan** olarak adlandırılır.
- `exit()` sistem çağrısı ile süreç sonlanabilir.
- `abort()` sistem çağrısı ile süreç sonlandırılabilir.

Süreç Durumları

- Bir süreç yürütülürken durum değiştirir
- **Yeni**: Süreç oluşturuldu
- **Çalışıyor**: Talimatlar yürütülüyor
- **Bekliyor**: Bir olayın gerçekleşmesi bekleniyor
- **Hazır**: Bir işlemciye atanma bekleniyor
- **Sonlandırıldı**: Yürütme tamamlandı

Süreç Durumları

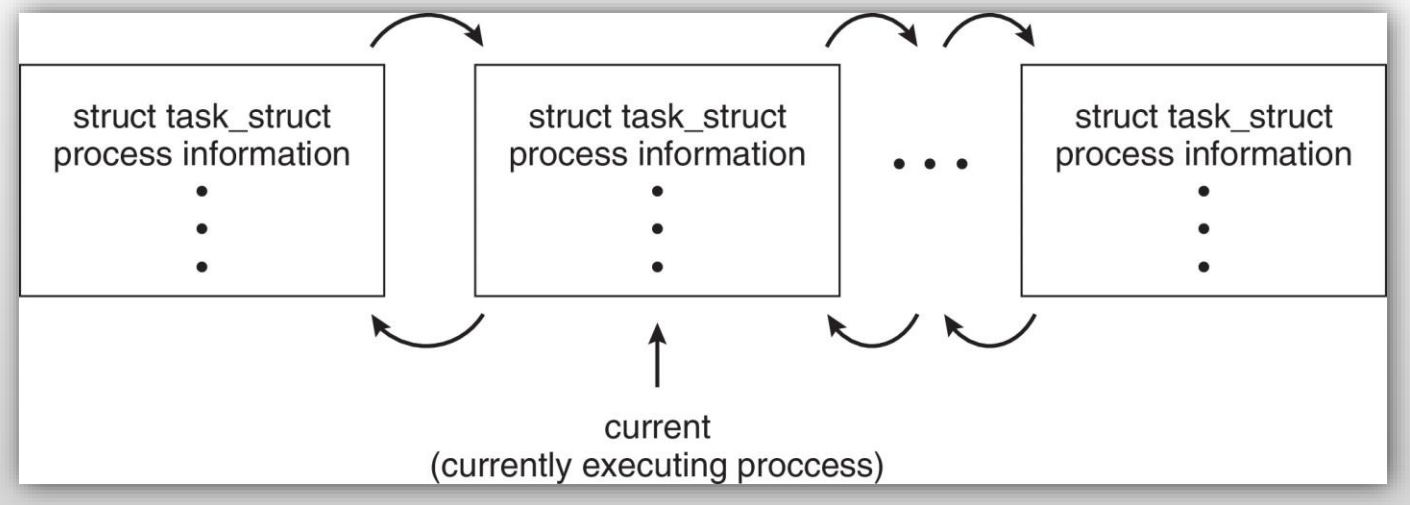


Süreç Tablosu

- Süreç tablosunda (process control block) bulunan bazı alanlar.
- **Süreç yönetimi**
  - Registers, Program counter, Program status Word, Stack pointer, Process state, 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
- **Bellek yönetimi**
  - Pointer to {text, data, stack} segment info
- **Dosya yönetimi**
  - Root directory, Working directory, File descriptors, User ID, Group ID

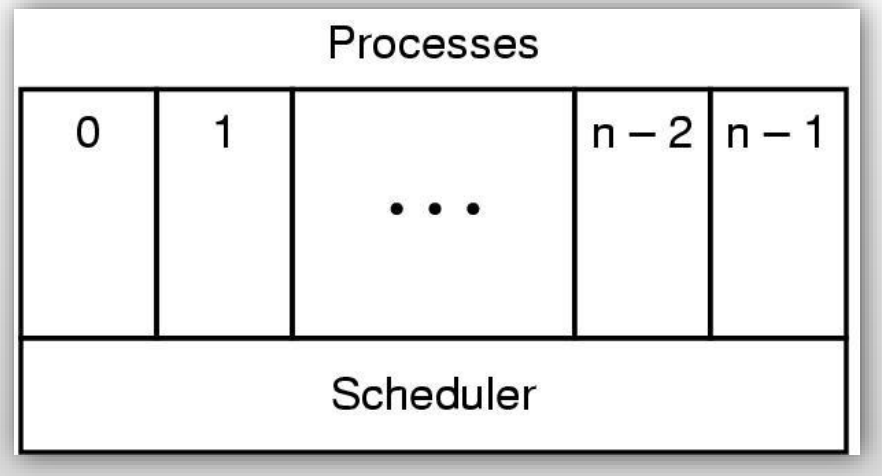
Süreçlerin Çizelgelenmesi

```
pid t_pid;                      /* process identifier */
long state;                     /* state of the process */
unsigned int time_slice         /* scheduling information */
struct task_struct *parent;     /* this process's parent */
struct list_head children;      /* this process's children */
struct files_struct *files;     /* list of open files */
struct mm_struct *mm;           /* address space of this process */
```

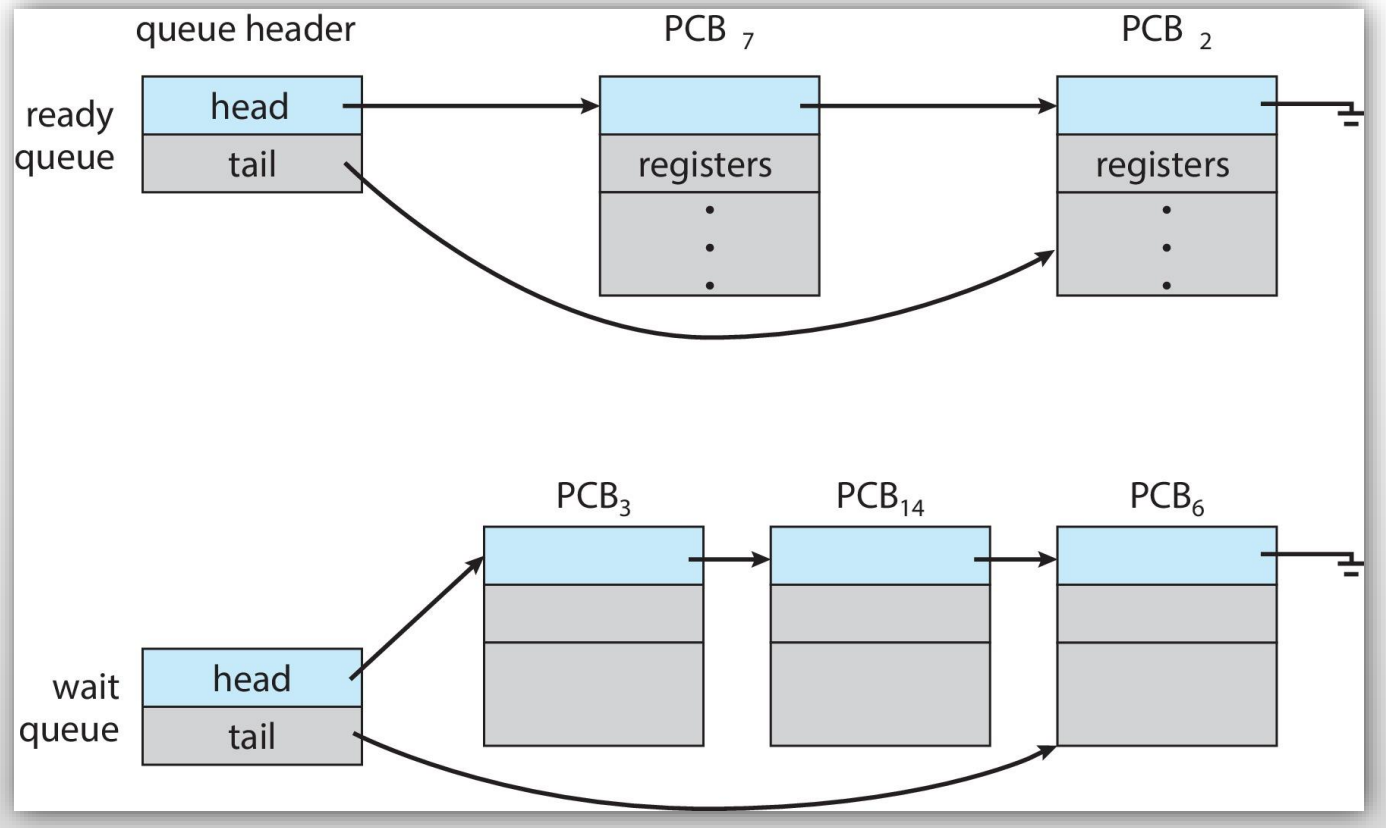


Süreçlerin Çizelgelenmesi

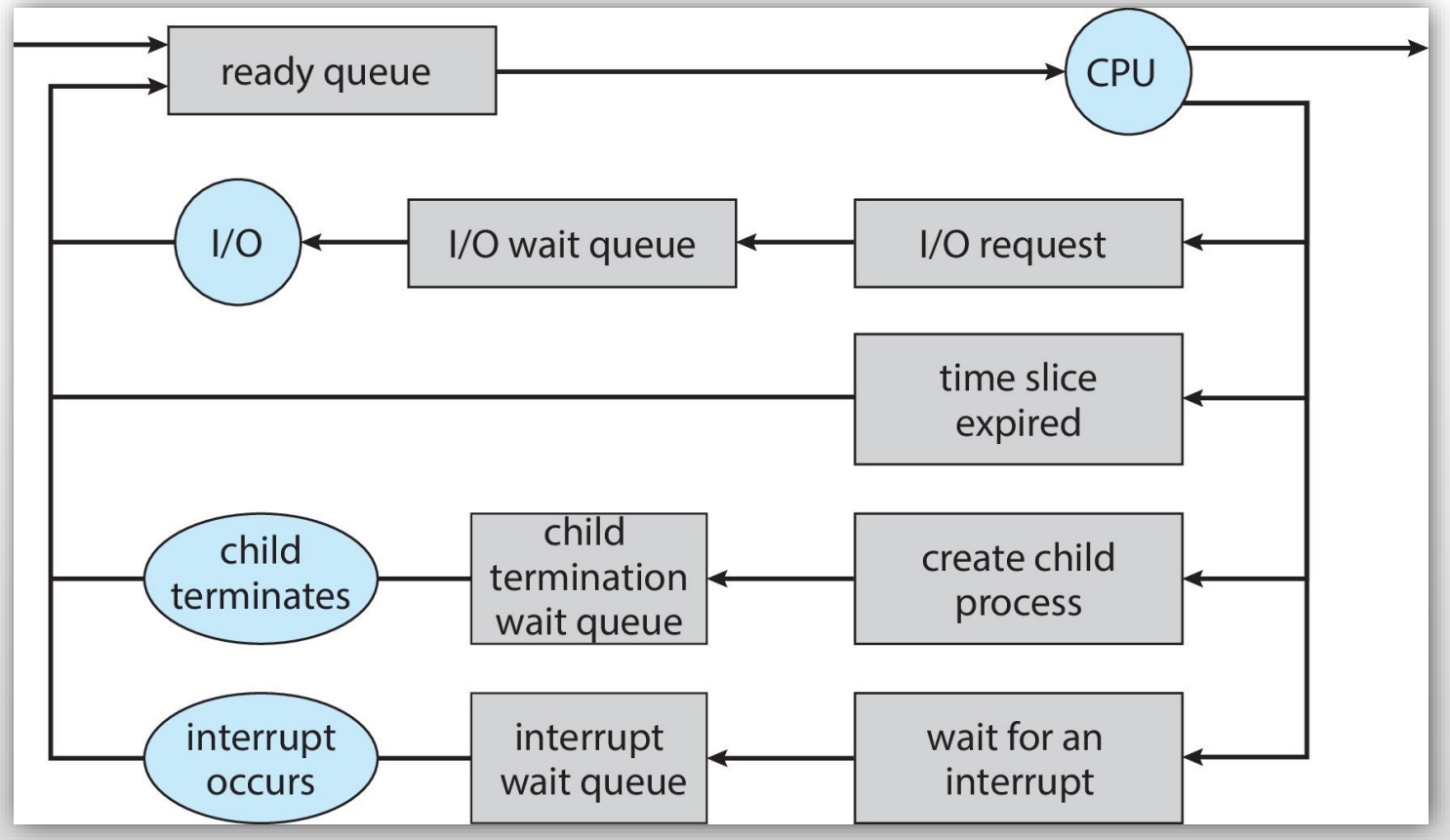
- İşletim sisteminin kesilme ve çizelgelemeyi yönetir.
- Çizelgeleyici, bir sonraki yürütme için mevcut süreçler arasından seçim yapar.
- **Ready queue**, ana bellekte bulunan, hazır ve yürütmeyi bekleyen süreçler
- **Wait queue**, bir olayı bekleyen süreçler kümesi (G/Ç gibi)



Hazır ve Bekleme Kuyrukları

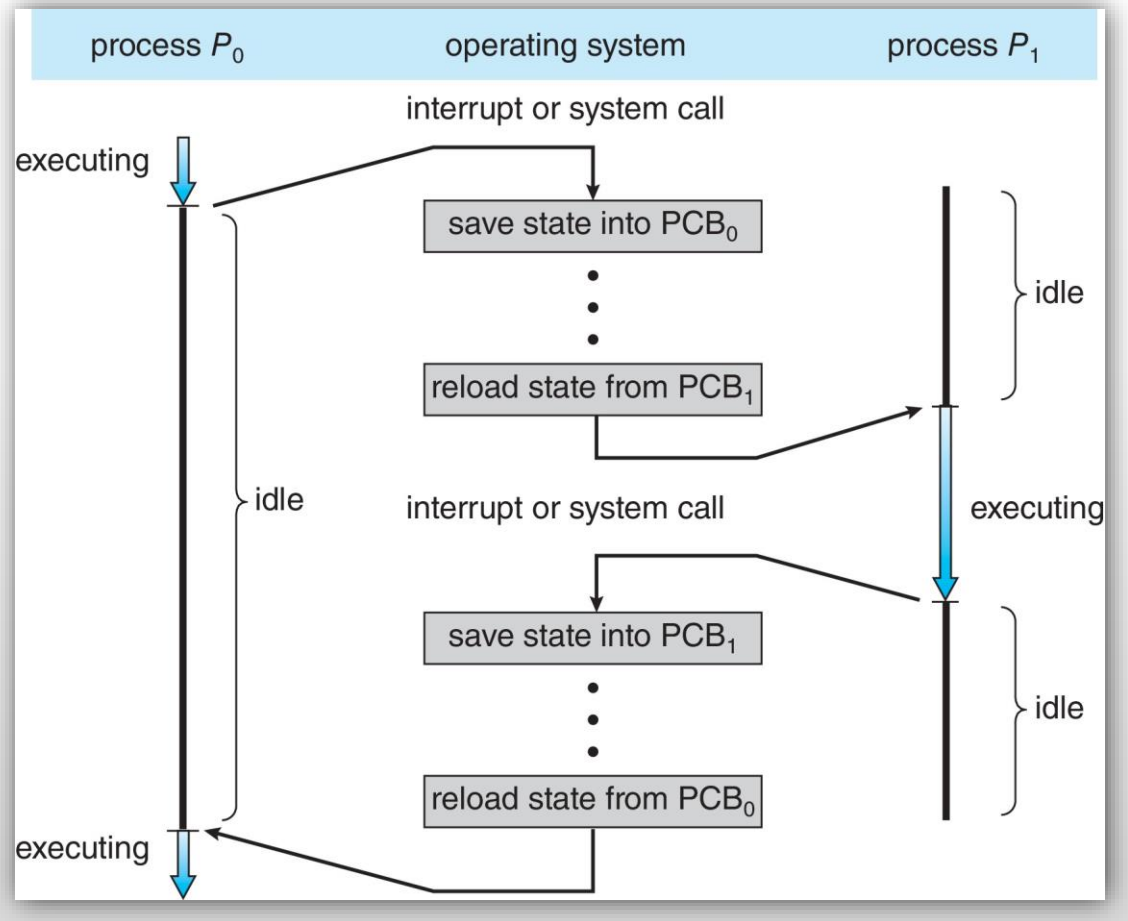


Süreçlerin Çizelgelenmesi



Bağlam Anahtarlama

- .



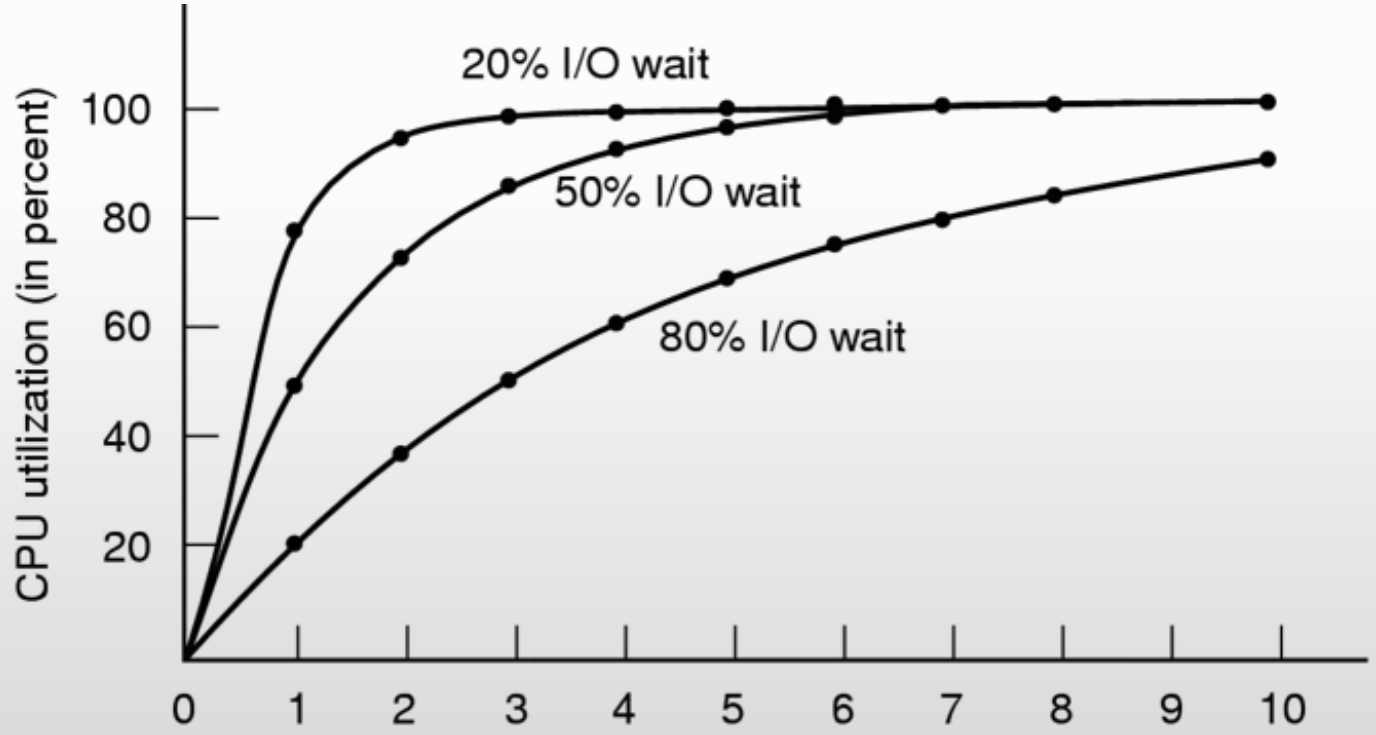
Kesilmenin Ele Alınması

Bir kesilme oluştuğunda işletim sisteminin en alt seviyesi ne yapar.

1. Donanım program sayacı vb. verileri kaydeder
2. Donanım, kesme vektöründen yeni program sayacı yükler.
3. Assembly dili prosedürü yazmaçları kaydeder.
4. Assembly dili prosedürü yeni yığın hazırlar.
5. C kesme hizmeti çalışır (genellikle girişi okur ve ara belleğe alır).
6. Çizelgeleyici hangi sürecin çalıştırılacağına karar verir.
7. C kesme hizmeti Assembly dili prosedürüne geri döner.
8. Assembly dili prosedürü seçilen süreci başlatır.

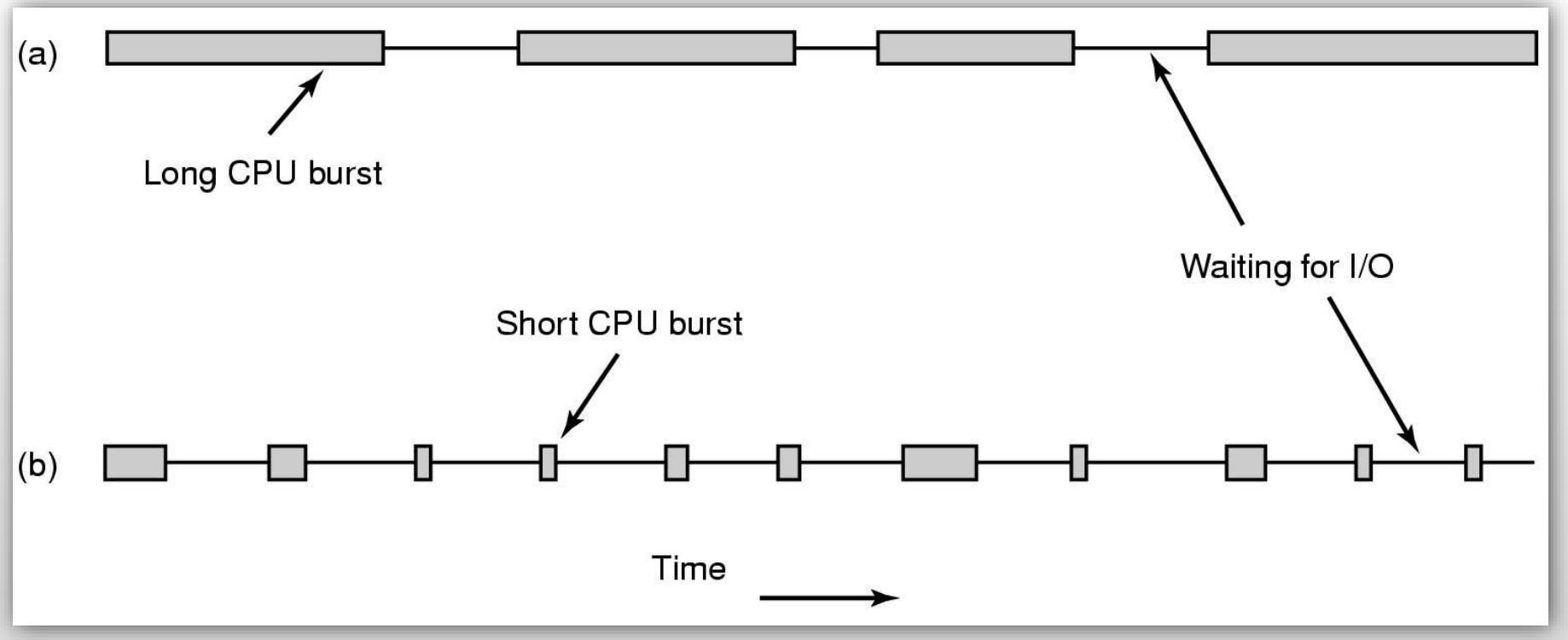
Çoklu Programlama Modellemesi

- Bellekte bulunan süreç sayısının bir fonksiyonu olarak CPU kullanımı grafiği.



İşlemci Kullanımı

(a) CPU'ya bağlı (bound) bir işlem. (b) G/Ç'ye bağlı bir işlem.

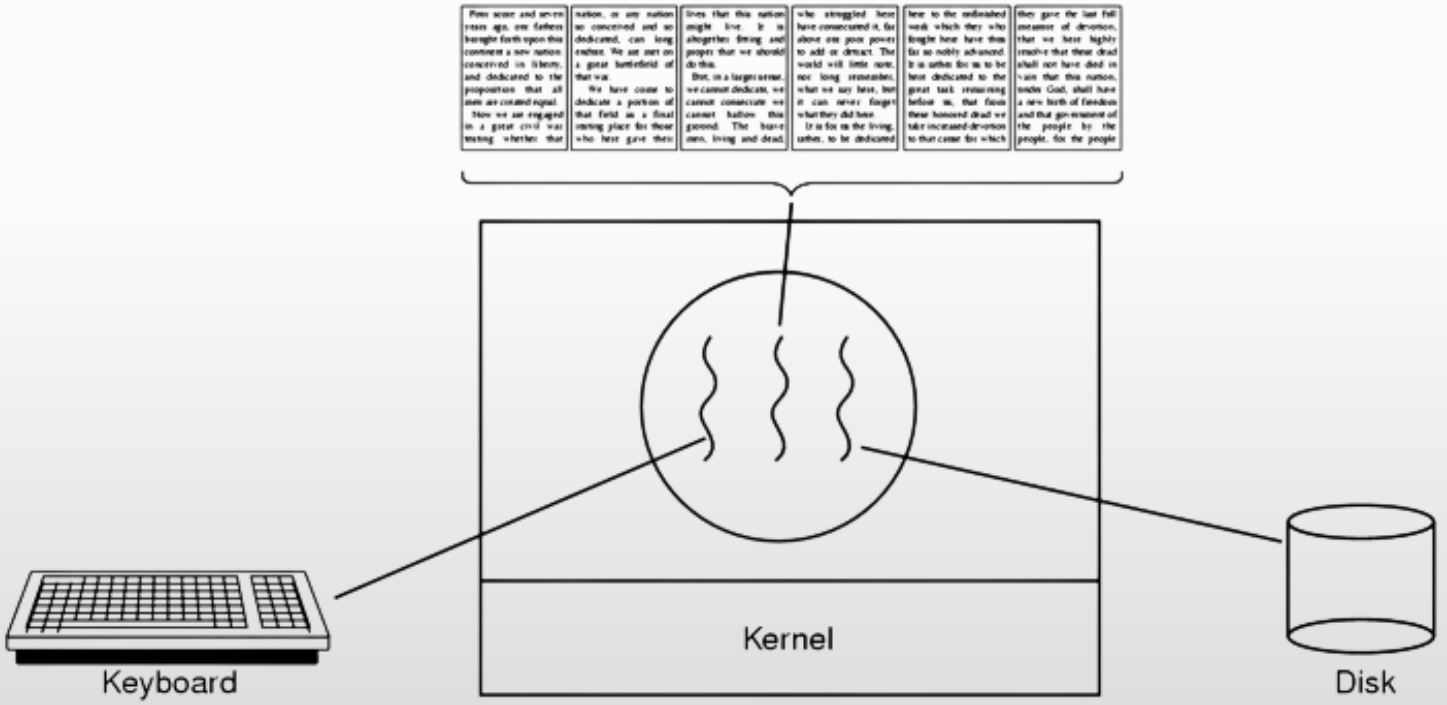


İşlemci Kullanımı

- CPU hızlandığında, süreçler daha fazla G/Ç bağlı olurlar
- Bir G/Ç bağlı süreç çalışmak istediğinde, hızlı bir şekilde bağlam anahtarlama yapmak gerekir.

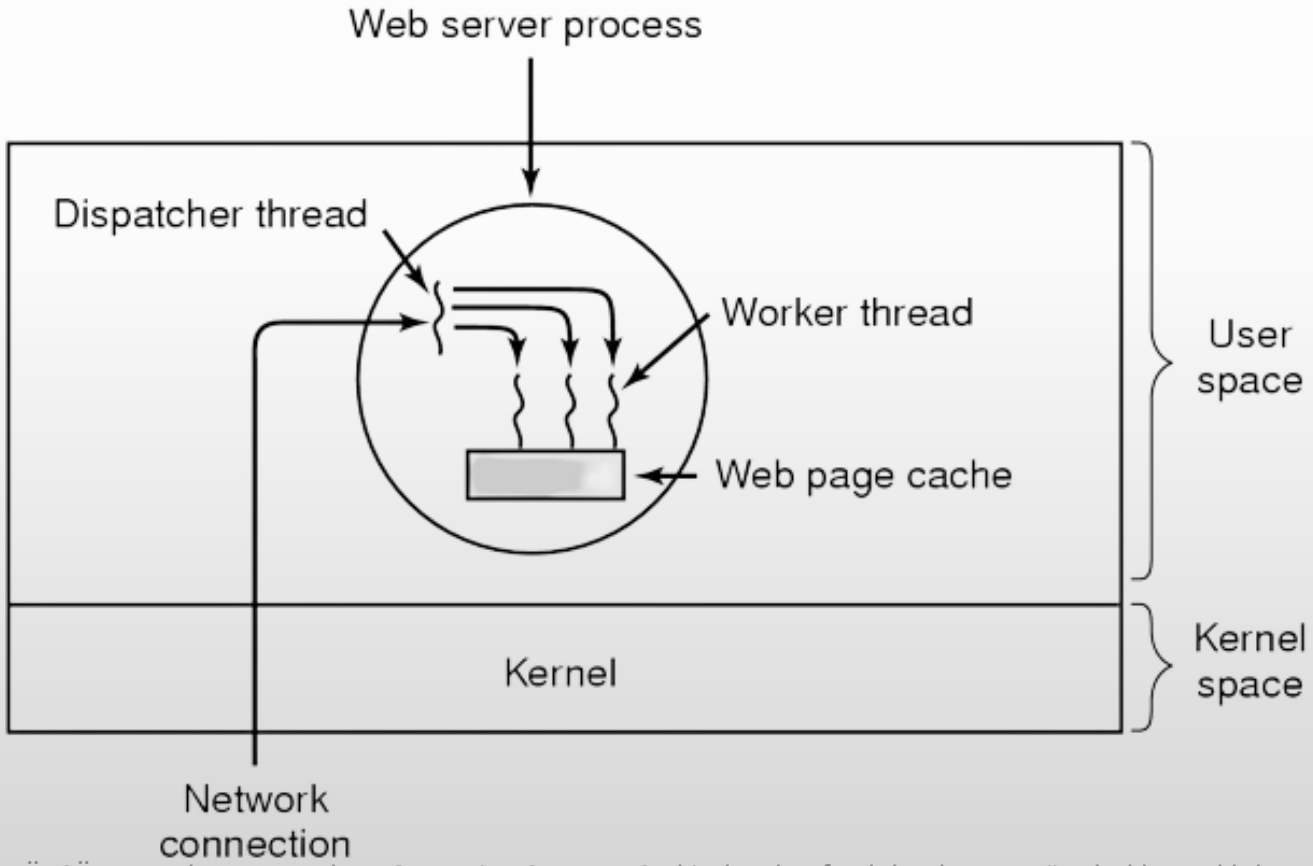
İş Parçacığı

- 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

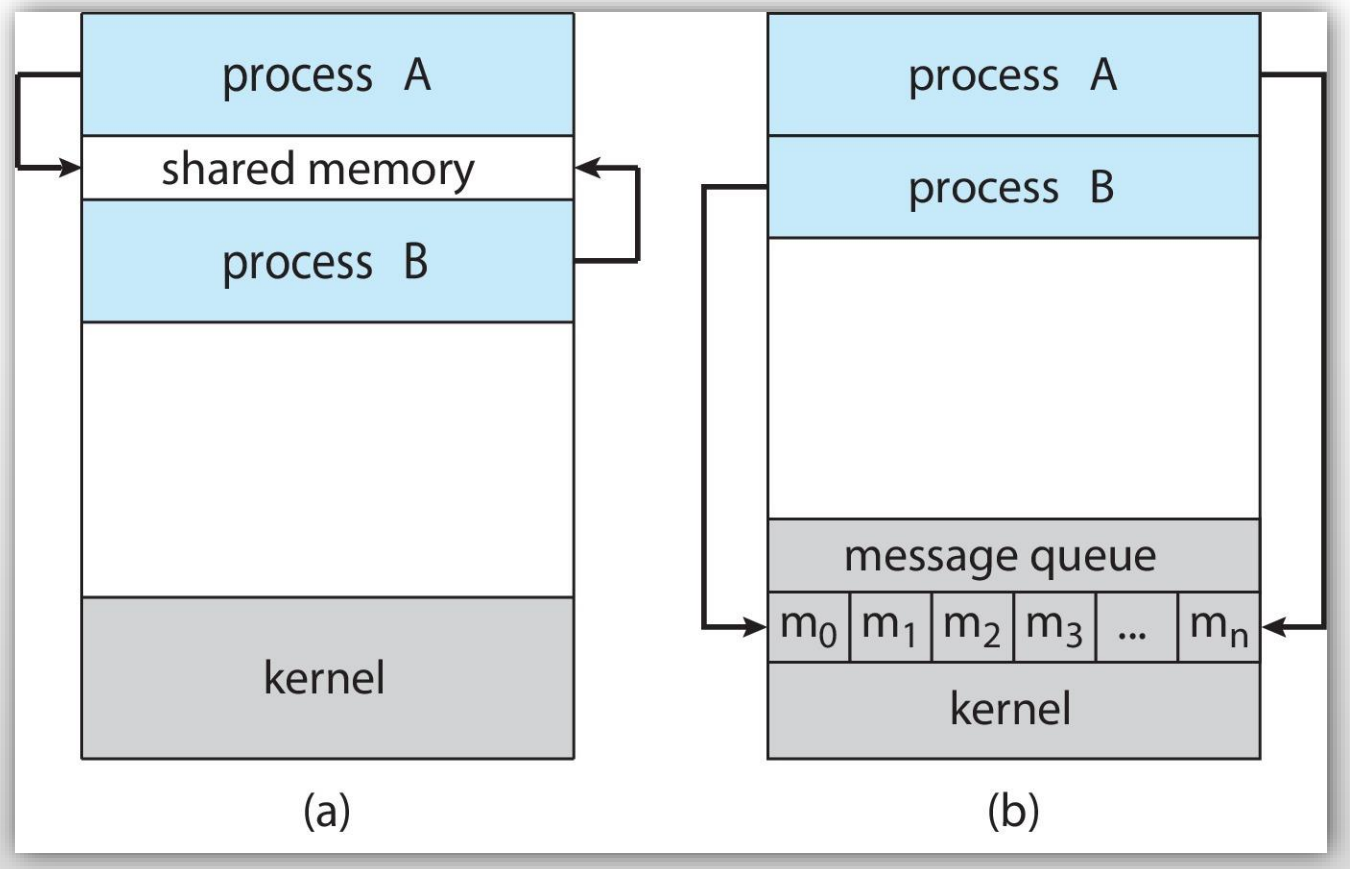
```
while (TRUE) {
  get_next_request(&buf);
  handoff work(&buf);
}
```

```
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);
}
```

Süreçler Arası İletişim Problemler

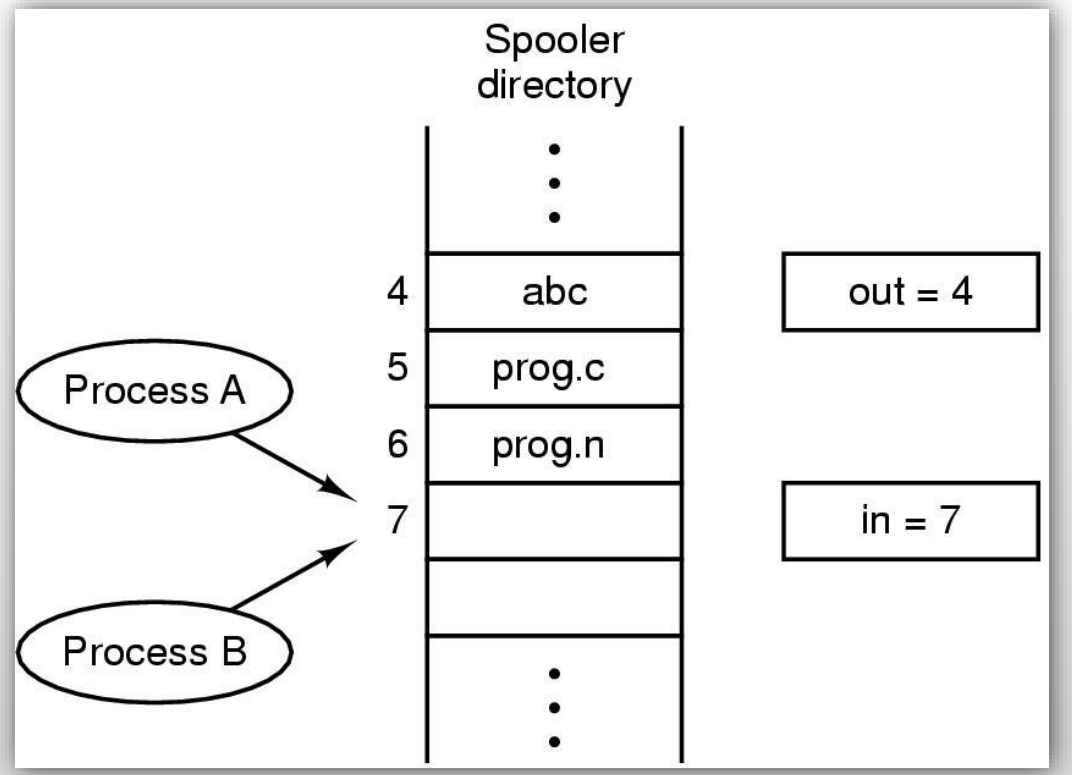
- Süreçler bağımsız ya da işbirlikçi olabilir.
- İşbirlikçi süreçler birbirleriyle veri paylaşımı yaparlar.
- Süreç çakışmalarıyla nasıl başa çıkılır (aynı koltuk için 2 havayolu rezervasyonu)
- Bağımlılıklar mevcutken doğru sıralama nasıl yapılır, silahı ateşlemeden önce nişan alınması
- İki yöntem var
  - Paylaşımlı bellek
  - Mesaj kuyrukları

Süreçler Arası İletişim



Süreçler Arası İletişim

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



Bağlam Anahtarlama

- İşlemci başka bir sürece geçtiğinde, eski sürecin durumunu kaydetmeli ve yeni işlem için durum bilgisini yüklemelidir.
- Bağlam süreç için PCB'de tutulan bilgilerdir.
- Bağlam anahtarlama süresi sisteme ek yük oluşturur.
- Sistem, geçiş yaparken bir iş yapmaz
- İşletim sistemi ve PCB ne kadar karmaşık olursa, bağlam anahtarlama o kadar uzun sürer.
- Donanım desteği bu süreyi kısaltabilir. (ekstra yazmaçlar gibi)

Mobil Sistemlerde Çoklu Görev

- İlk versiyonlarda tek süreç çalışabiliyordu.
- Kullanıcı arayüzü kısıtlarından dolayı
  - Ön planda tek bir süreç çalışabilir
  - Arka planda bir çok süreç çalışabilir
  - Ses oynatma gibi özelliklerde kısıtlamalar olabilir.
- Android süreçlerin kullanımı için service arayüzü vardır.
  - Service süreç suspend olsa bile çalışmaya devam edebilir.

Android Süreç Öncelikleri

- Yüksekten düşüğe göre
  - Ön planda çalışan süreç (foreground)
  - Görünür süreç (visible)
  - Hizmet süreci (service)
  - Arka planda çalışan süreçler (background)
  - Boş süreçler (empty)

Yarış Durumu

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

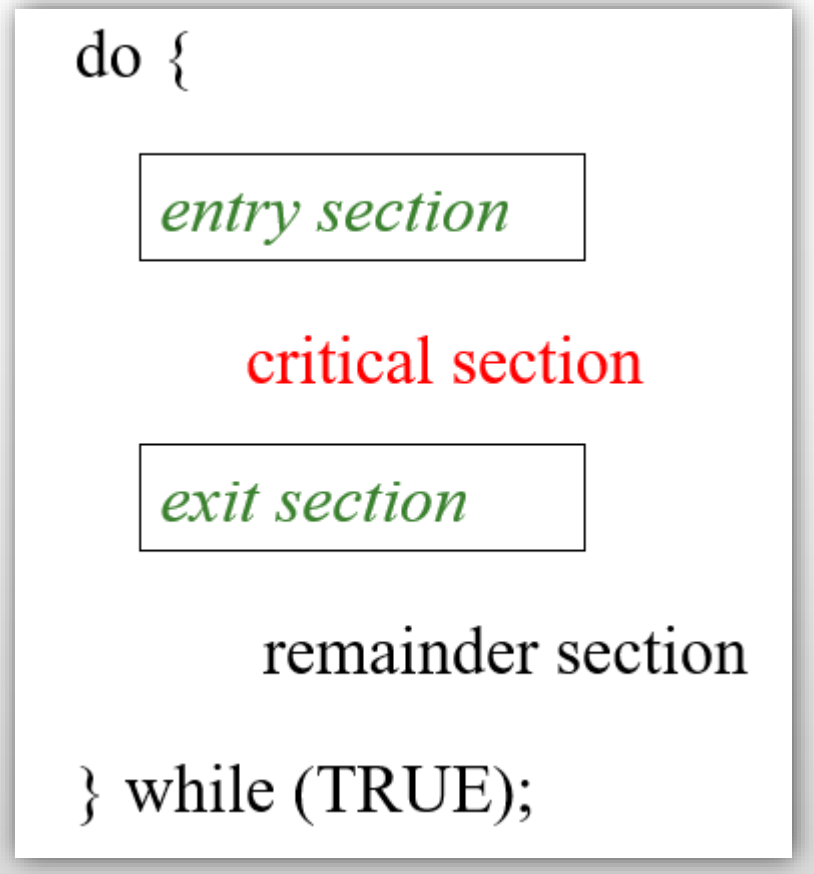
- Karşılıklı dışlama
  - Birden fazla işlemin paylaşılan verileri aynı anda okumasını ve yazmasını engelleme
- Kritik bölge
  - Programın paylaşılan alana erişim yaptığı kod bölümü

Karşılıklı Dışlama

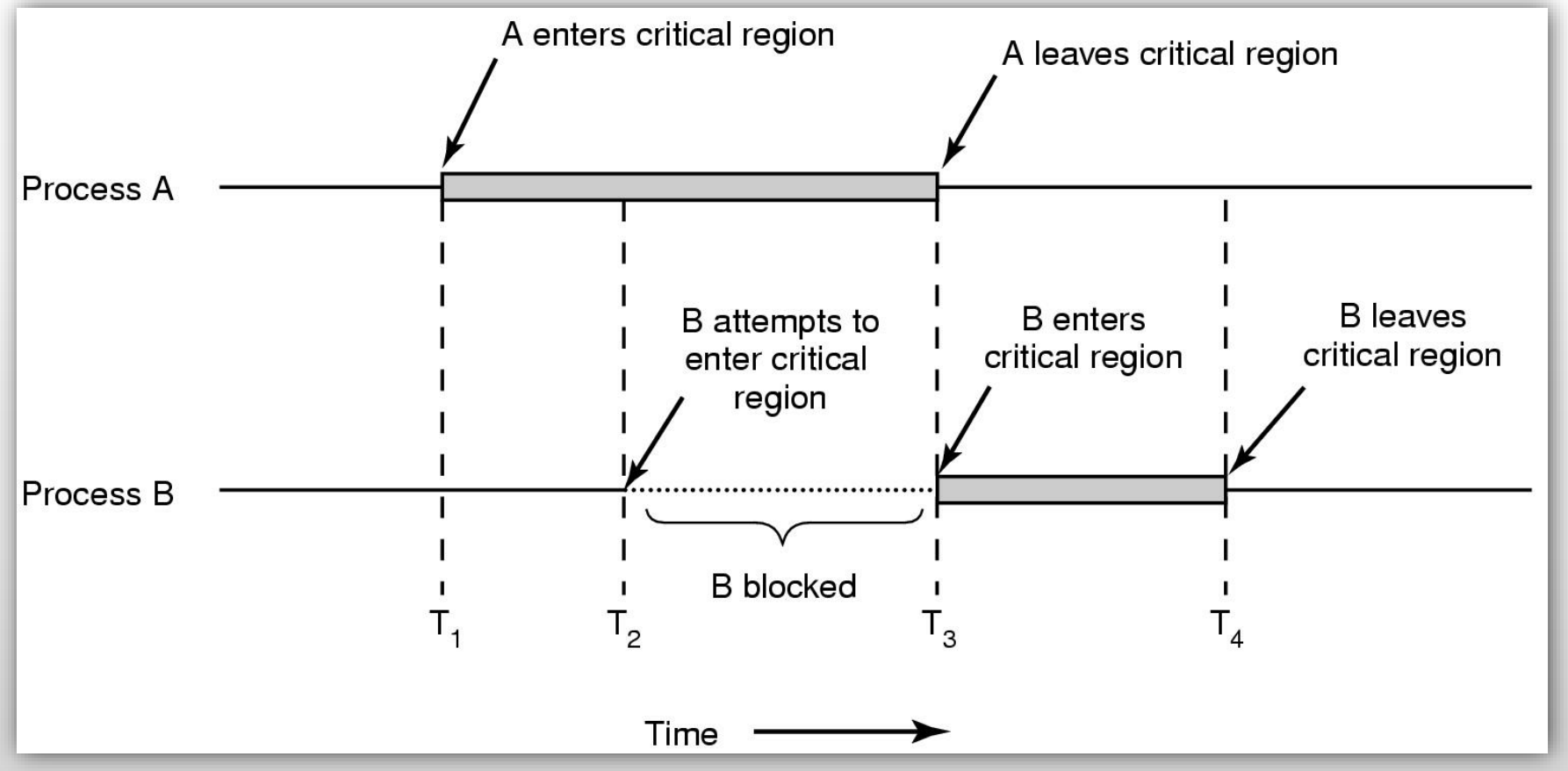
Karşılıklı dışlama sağlamak için dört koşul

- 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



Kritik Bölge Kullanarak Karşılıklı Dışlama



Peterson’un Çözümü

- Algoritma, hangi işlemin kritik bölüme girmesi gerektiğini belirtmek için "turn" ve "flag[2]" olmak üzere iki değişken kullanır.
- flag, süreç tarafından kritik bölüme girme niyetini belirtir.
- turn, daha sonra hangi sürecin gireceğini belirtir.
- Algoritma, her iki işlemin de kritik bölüme aynı anda girmesini önlemek için bir meşgul bekleme döngüsü ve bir dizi koşul kullanır.
- Algoritma, karşılıklı dışlamayı sağlar ve süreçlerin sonsuz bir bekleme döngüsüne girmesini engeller.
- Meşgul bekleme döngüsü önemli miktarda CPU zamanı tüketebilir!

Peterson'un Çözümü

```
private static final int N = 2; // Number of threads
private static volatile boolean[] flag = new boolean[N];
private static volatile int turn = 0;
private static int counter;

private static void incrementCounter() {
     int i = (int) (Thread.currentThread().getId() % N);
     int j = (i + 1) % N;
     flag[i] = true;
     turn = j;
     while (flag[j] && turn == j) {} // Spin loop
     // Critical region
     counter++;
     System.out.println("Counter: " + counter + " i: " + i + " j: " + j);
     flag[i] = false;
}
```

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 bloke etme
  - Önce uyandır, sonra uyut (wake up, sleep)

Üretici Tüketici Problemi

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

- İki versiyon
  - Sınırsız tampon bellek
    - Üretici beklemez, tüketici tampon boş ise bekler.
  - Sınırlı tampon bellek
    - Tampon dolu ise üretici bekler
    - Tampon boş ise tüketici bekler

Ölümcül Yarış Durumu - Producer

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

void producer()
{
  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? */
  }
}
```

Ölümcül Yarış Durumu - Consumer

```
void consumer()
{
  while (true) { /* repeat forever */
    if (count == 0) sleep(); /* if buffer empty, 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 */
  }
}
```

Mesaj Gönderme/Alma

- İki süreç haberleşmek için iletişim bağı kurmalı
- Fiziksel
  - Paylaşımlı bellek
  - Donanım veriyolu
  - Ağ
- Mantıksal
  - Doğrudan, dolaylı
  - Senkron, asenkron
  - Otomatik, tamponlayarak

Doğrudan İletişim

- Süreçler açık olarak adlarını belirtmeliler
  - `send (P, message)` – P sürecine mesaj gönder
  - `receive (Q, message)` – Q sürecinden mesaj al
- Bağlantı otomatik kurulur
- Bir bağlantı bir çift (pair) süreçle ilişkilidir.
- Her bir çift süreç arasında sadece bir bağlantı vardır
- Bağlantı genellikle çift yönlüdür.

Doğrudan İletişim

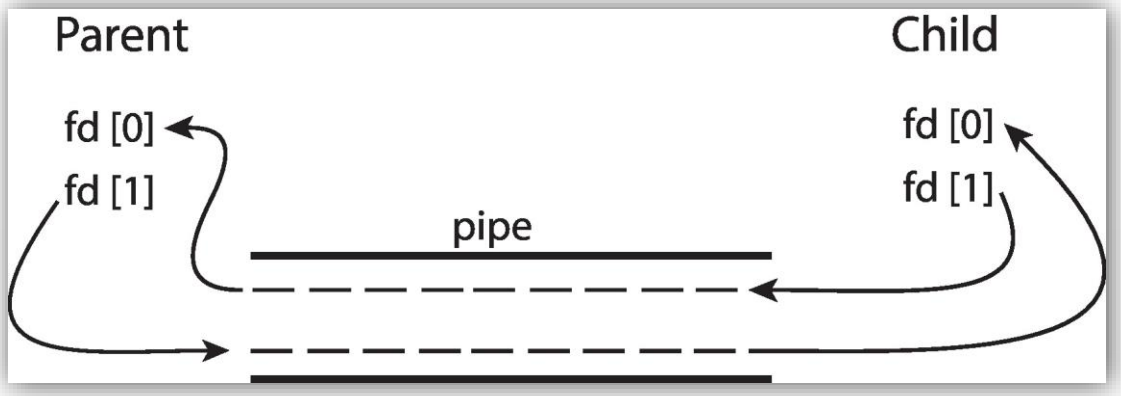
- Posta kutusu aracılığıyla iletişim sağlanır
  - Her posta kutusu tekil tanımlayıcıya sahip
  - Süreçler aynı posta kutusunu paylaşıyorsa haberleşebilir
  - `send (A, message)` – A posta kutusuna mesaj gönder
  - `receive (A, message)` – A posta kutusundan mesaj al
- Bir bağlantı bir çok süreç ile ilişkilendirilebilir
- Her bir süreç çifti bir çok bağlantı paylaşabilir.
- Bağlantılar tek yönlü ve çift yönlü olabilir

Boru Hattı (pipe)

- İki sürecin iletişim kurmasına izin veren bir kanal görevi görür
- Sorunlar:
  - İletişim tek yönlü mü yoksa çift yönlü mü?
  - İki yönlü iletişim durumunda, yarı mı yoksa tam çift yönlü mü?
  - İletişim süreçleri arasında bir ilişki (yani ebeveyn-çocuk) olmalı mı?
  - Bir ağ üzerinden kullanılabilir mi?
- Sıradan boru hattı
  - Süreç dışından erişelemez
  - Ebeveyn çocuk süreç ile haberleşmek için yaratır
- Adlandırılmış boru hattı
  - Tüm ssüreçler tarafından erişilebilir

Boru Hattı (pipe)

- Sıradan boru hattı
  - Üretici bir sona yazar
  - Tüketici bir sondan okur.
  - Tek yönlüdür.
  - Ebeveyn-çocuk ilişkisi
  - Adsız boru hattı olarak da geçer
- Adlandırılmış boru hattı
  - Çift yönlü iletişim
  - Ebeveyn-çocuk ilişkisi gerektirmez
  - Windows, UNIX destekler

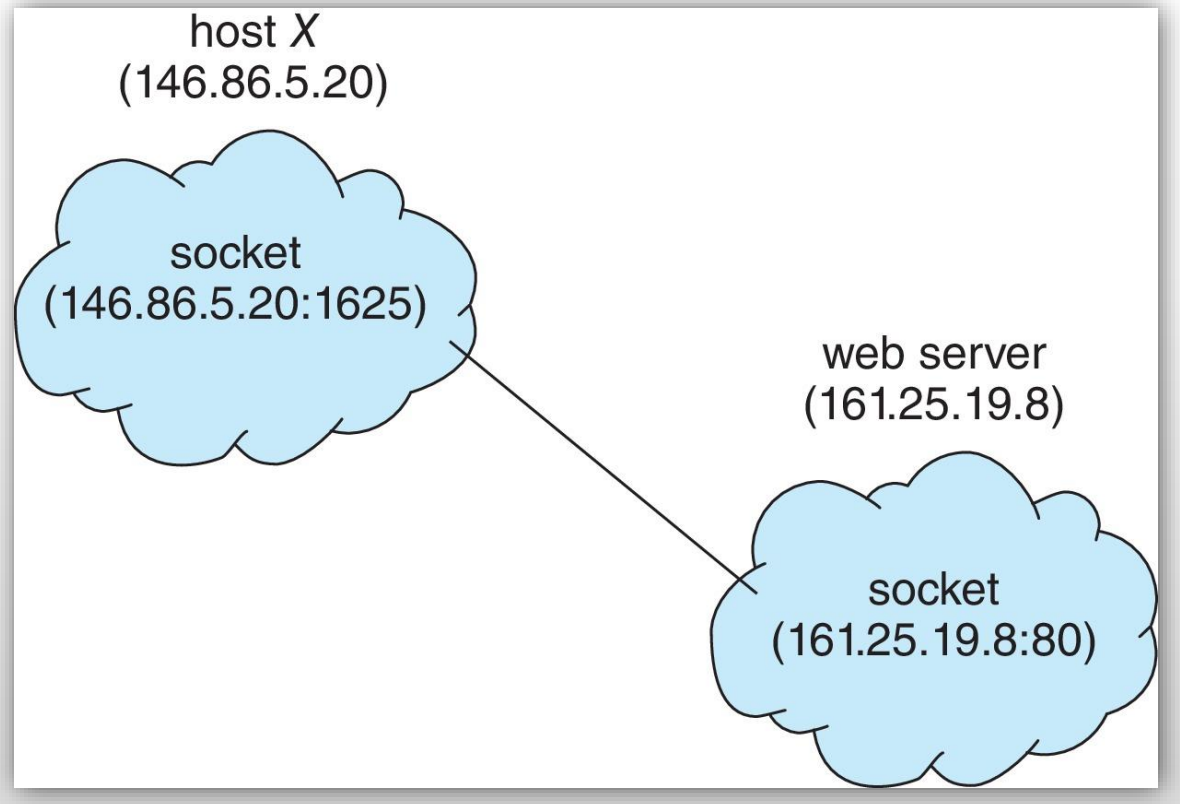


Soketler

- Soket, iletişim için uç nokta olarak tanımlanır
- IP adresi ve bağlantı noktasının birleştirilmesi (IP + port)
- Port: ağ hizmetlerini ayırt etmek için kullanılır
- 161.25.19.8:1625, **IP**:161.25.19.8 **port**:1625
- İletişim bir çift soket arasında oluşur
- 1024'ün altındaki tüm bağlantı noktaları iyi bilinir ve standart hizmetler için kullanılır
- Sürecin üzerinde çalıştığı sisteme atıfta bulunmak için özel IP adresi 127.0.0.1 (geri döngü)

Soketler

- .

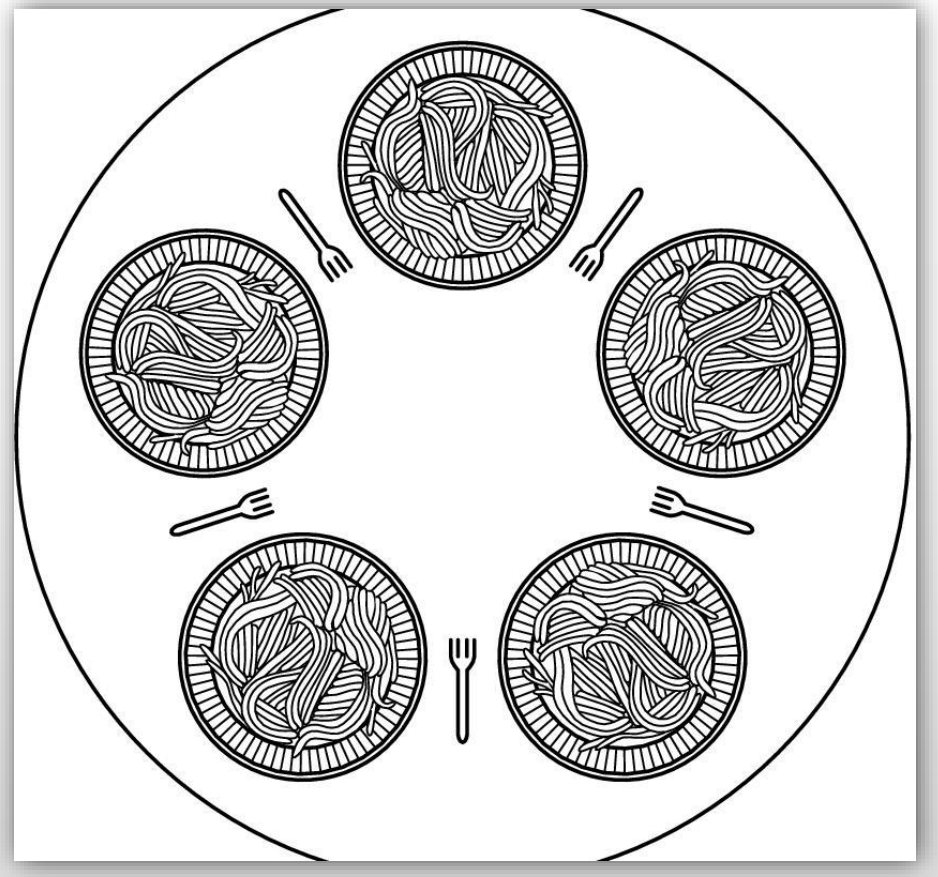


Süreçler Arası İletişim Problemleri

- Dining philosopher problemi
  - Bir filozof ya yer ya da düşünür
  - Aç kalırsa, iki çatal alıp yemeye çalış

- Okur-Yazar problemi
  - Bir veritabanına erişimi modeller

Dining Philosophers Problemi



Dining Philosophers

```
while(true) {
// Initially, thinking
  think();
// Take a break from thinking, hungry now
  pick_up_left_fork();
  pick_up_right_fork();
  eat();
  put_down_right_fork();
  put_down_left_fork();
// Not hungry anymore. Back to thinking!
}
```

Dining Philosophers - loop

```
#define LEFT (i+N-1)%N /* number of i's left neighbor */
#define RIGHT (i+1)%N /* number of i's right neighbor */

void philosopher(int i) /* i: philosopher number, from 0 to N-1 */
{
    while (TRUE) { /* repeat forever */
        think(); /* philosopher is thinking */
        take_forks(i); /* acquire two forks or block */
        eat(); /* philosopher is eating */
        put_forks(i); /* put both forks back on table */
    }
}
```

Dining Philosophers – take forks

```
void take_forks(int i) /* i: philosopher number, from 0 to N-1 */
{
  down(&mutex); /* enter critical region */
  state[i] = HUNGRY; /* record fact that philosopher i is hungry */
  test(i); /* try to acquire 2 forks */
  up(&mutex); /* exit critical region */
  down(&s[i]); /* block if forks were not acquired */
}
```

Dining Philosophers – put forks

```
void put_forks (i) /*i: philosopher number, from 0 to N-1 */
{
  down(&mutex); /* enter critical region */
  state[i]= THINKING; /* philosopher has finished eating */
  test(LEFT); /* see if left neighbor can now eat */
  test(RIGHT); /* see if right neighbor can now eat */
  up(&mutex); /* exit critical region */
}
```

Dining Philosophers – test state

```
void test(i) /* i: philosopher number, from 0 to N-1 */
{
  if (state[i] == HUNGRY &&
      state[LEFT] != EATING &&
      state[RIGHT] != EATING) {
   state[i]= EATING;
   up(&s[i]);
   }
}
```

Okur-Yazar Problemi - writer

```
semaphore mutex = 1; /* controls access to 'rc' */
semaphore db = 1; /* controls access to the database */
int rc = 0; /* # of processes reading or wanting to */

void writer(void)
{
    while (TRUE) { /* repeat forever */
      think_up_data(); /* noncritical region */
      down(&db); /* get exclusive access */
      write_data_base(); /* update the data */
      up(&db); /* release exclusive access */
  }
}
```

Okur-Yazar Problemi - reader

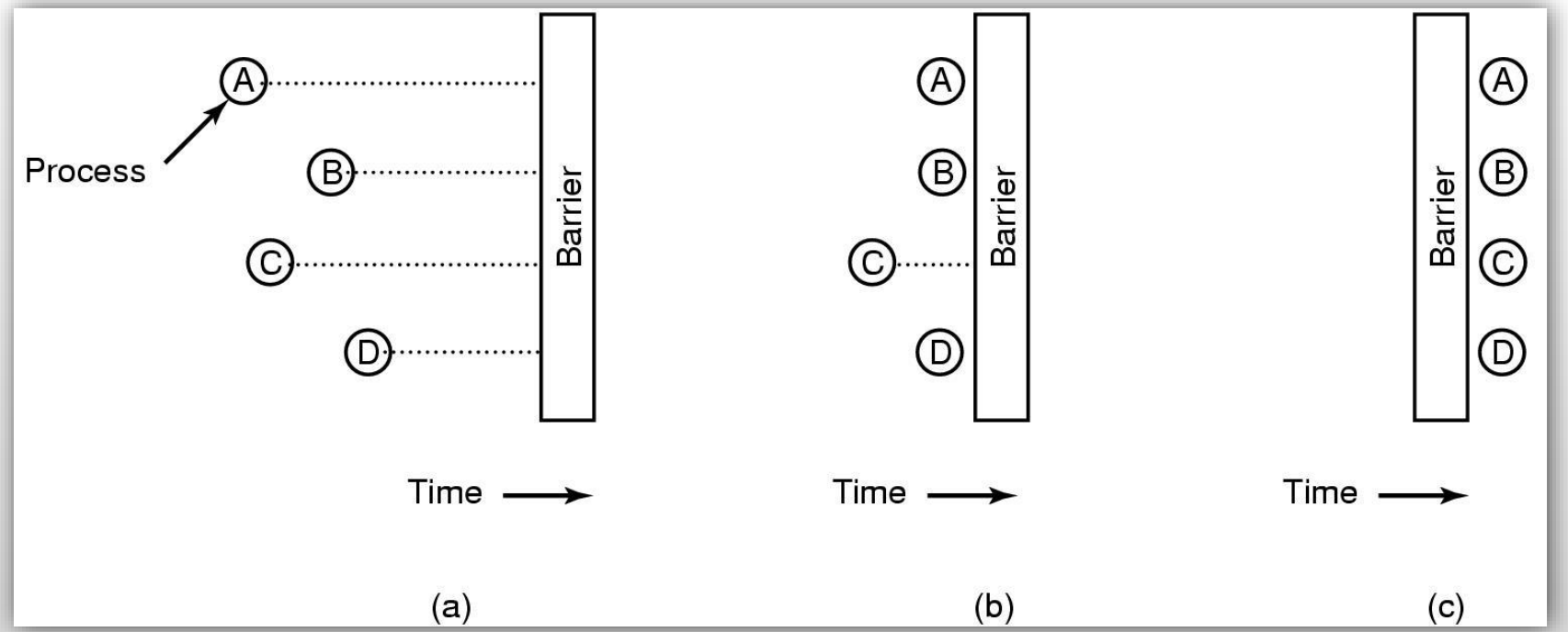
```
void reader(void)
{
  while (TRUE) { /* repeat forever */
    down(&mutex); /* get exclusive access to 'rc' */
    rc = rc + 1; /* one reader more now */
    if (rc == 1) down(&db); /* if this is the first reader ... */
    up(&mutex); /* release exclusive access to 'rc' */
    read_data_base(); /* access the data */
    down(&mutex); /* get exclusive access to 'rc' */
    rc rc - 1; /* one reader fewer now */
    if (rc == 0) up(&db); /* if this is the last reader ... */
    up(&mutex); /* release exclusive access to 'rc' */
    use_data_read(); /* noncritical region */
  }
}
```

Okur-Yazar Problemi

- Çözümün dezavantajı nedir?
  - Yazar süreci açlık (starvation) tehlikesiyle karşı karşıya

Bariyer (Barriers)

- Bariyerler, süreç gruplarını senkronize etmek için tasarlanmıştır.
- Genellikle bilimsel hesaplamalarda kullanılır.



Çizelgeleme

- Bir sonraki adımda hangi süreç çalıştırılacak?
- Bir süreç çalışırken, işlemci süreç sonuna kadar çalışmalı mı yoksa farklı süreçler arasında geçiş yapmalı mı?

- Süreç değiştirme pahalı
  - Kullanıcı modu ve çekirdek modu arasında geçiş
  - Geçerli süreç kaydedilir
  - Bellek haritası (memory map) kaydedilir
  - Önbellek temizlenir ve yeniden yüklenir

Kavramlar

- Önleyici (preemptive) algoritma
  - Bir süreç, zaman aralığının sonunda hala çalışıyor durumunda ise, askıya alınır ve başka bir süreç çalıştırılır.
- Önleyici olmayan (non-preemptive) algoritma
  - Çalıştırmak için bir süreç seçilir ve bloke olana kadar veya gönüllü olarak işlemciyi serbest bırakana kadar çalışmasına izin verilir.

Çizelgeleme Kategorileri

- Farklı ortamlar farklı çizelgeleme algoritmalarına ihtiyaç duyar
- Toplu (batch)
  - Hala yaygın olarak kullanılıyor
  - Önleyici olmayan algoritmalar süreç geçişlerini azaltır
- Etkileşimli
  - Önleyici algoritma gerekli
- Gerçek zamanlı
  - Süreçler hızlı çalışır ve bloke olur

Çizelgelemenin Hedefleri

- Tüm sistemler
  - Adalet - her işleme CPU'dan adil bir pay vermek
  - Politika uygulama - belirtilen politikanın yürütüldüğünü görme
  - Denge - sistemin tüm parçalarını meşgul tutmak
- Toplu sistemler
  - Verim – birim zamanda yapılan işi maksimize etmek
  - Geri dönüş süresi – başlatma ve sonlandırma arasındaki süreyi en aza indirmek
  - CPU kullanımı - CPU'yu her zaman meşgul tutmak

Çizelgelemenin Hedefleri

- Etkileşimli sistemler
  - Yanıt süresi - isteklere hızla yanıt verilmeli
  - Orantılılık - kullanıcıların beklentilerini karşılamalı
- Gerçek zamanlı sistemler
  - Son teslim zamanı (deadline) - veri kaybı olmamalı
  - Öngörülebilirlik - multimedya sistemlerinde kalite düşüşünden kaçınmalı

SON