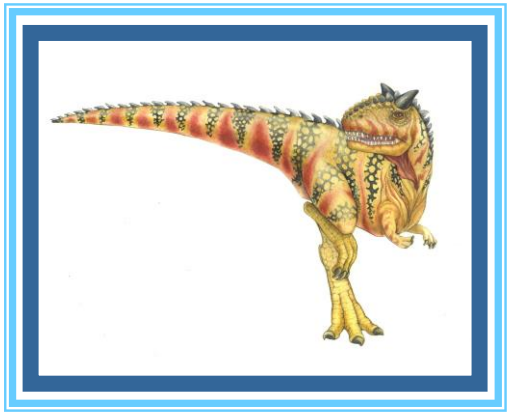
Bölüm 3: Süreçler



Mehmet Demirci tarafından çevrilmiştir.

Bölüm 3: Süreçler

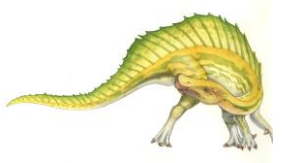
- Süreç Kavramı
- Süreç Planlama
- Süreç İşlemleri
- Süreçler Arası İletişim
- Sunucu-İstemci Sistemlerinde İletişim

Hedefler

- Süreç kavramını tanıtmak – yürütülen bir program, bilgisayar işlemlerinin temeli
- Süreçlerin çeşitli özelliklerini açıklamak (planlama, oluşturma, sonlandırma, iletişim)
- Süreçler arası iletişimi (paylaşılan bellek ve mesajlaşma) incelemek
- Sunucu-istemci sistemlerinde iletişimi açıklamak

Süreç Kavramı

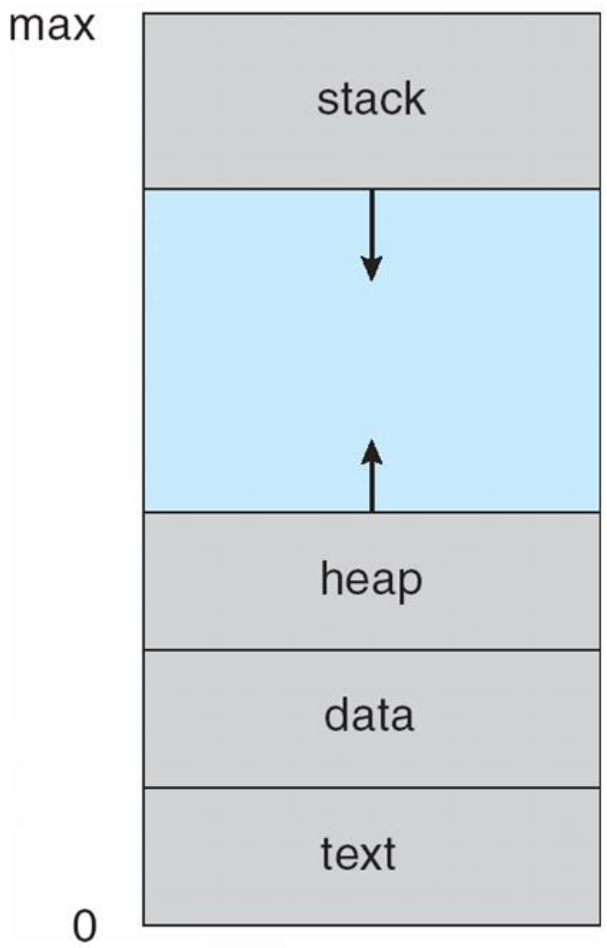
- İşletim sistemi çeşitli programlar yürütür:
  - Toplu komut sistemi (Batch system) – **işler (jobs)**
  - Zaman paylaşımlı sistemler – **kullanıcı programları** veya **görevler (tasks)**
- **Süreç (process)** – yürütülmekte olan program; süreç yürütme sıralı biçimde ilerlemeli.
- Birkaç parçadan oluşur
  - Program kodu, **metin bölümü (text section)**
  - Mevcut etkinlik – **program sayacı (program counter)**, işlemci registerları
  - **Stack** - geçici veri
    - Fonksiyon parametreleri, dönüş adresleri, yerel değişkenler
  - **Veri bölümü (data section)** - global değişkenler
  - **Heap** – çalışma zamanında dinamik olarak ayrılan bellek



Süreç Kavramı

- Program ***pasif*** varlık, diskte saklanır (**yürütülebilir dosya** - **executable file**), süreç ***aktif***
  - Yürütülebilir dosya belleğe yüklenince program süreç haline gelir.
- Yürütme, GUI'den fare ile, CLI'dan ismiyle vs. başlayabilir.
- Bir program birkaç süreçten oluşabilir.
  - Aynı programı yürüten birkaç kullanıcı örneği

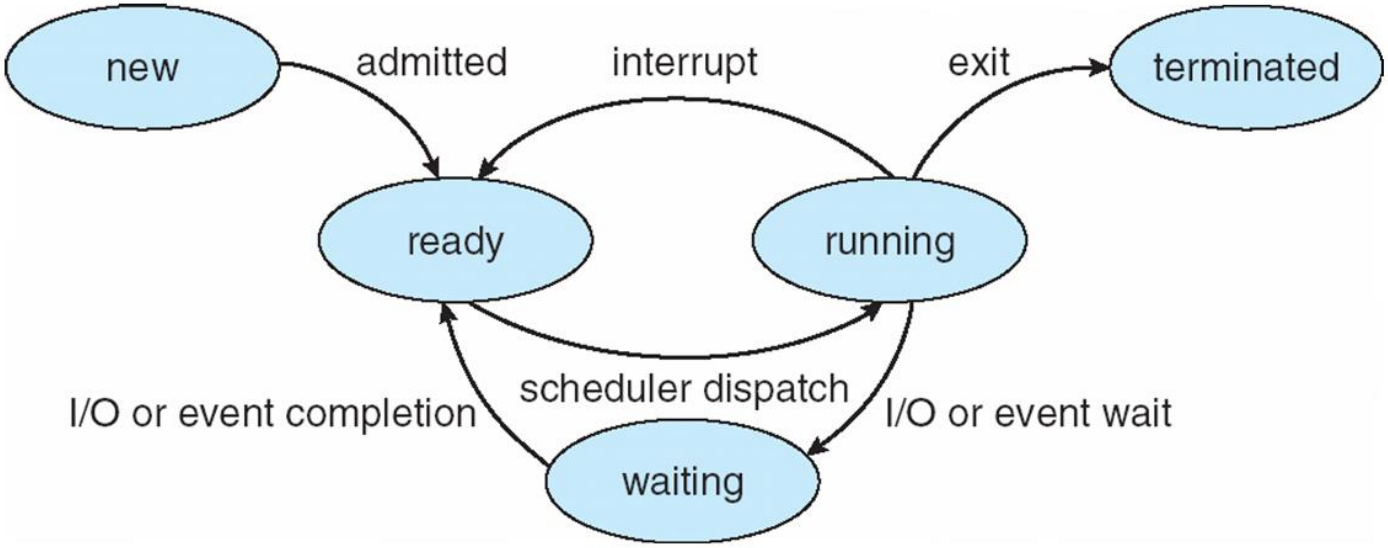
Bellekte Süreç



Süreç Durumu

- Süreç yürütülürken durumu (**state)** değişir,
  - **yeni**: Süreç oluşturuluyor
  - **çalışıyor**: Komutlar yürütülüyor
  - **bekliyor**: Süreç bir olayın gerçekleşmesini bekliyor
  - **hazır**: Süreç işlemciye atanmayı bekliyor
  - **sonlanmış**: Sürecin yürütülmesi bitti

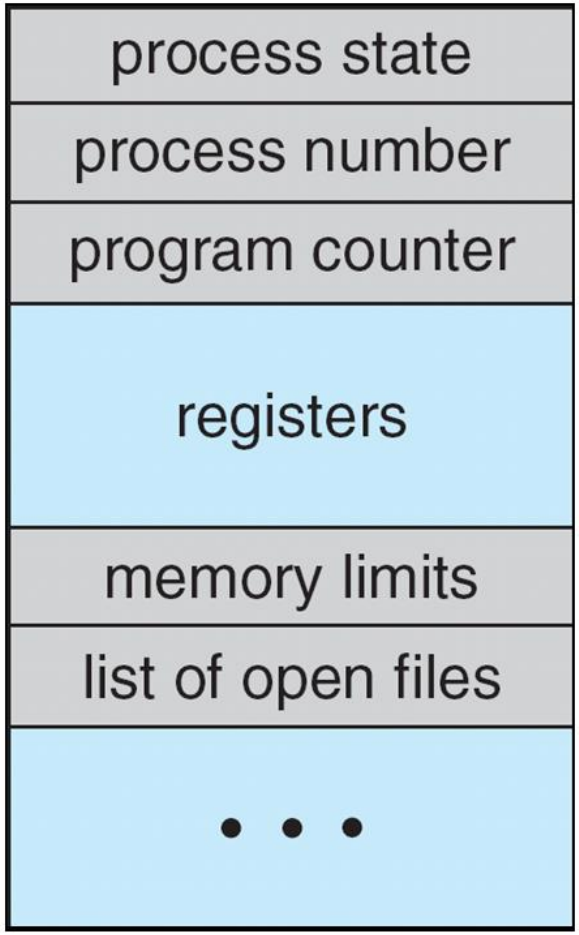
Süreç Durum Diyagramı



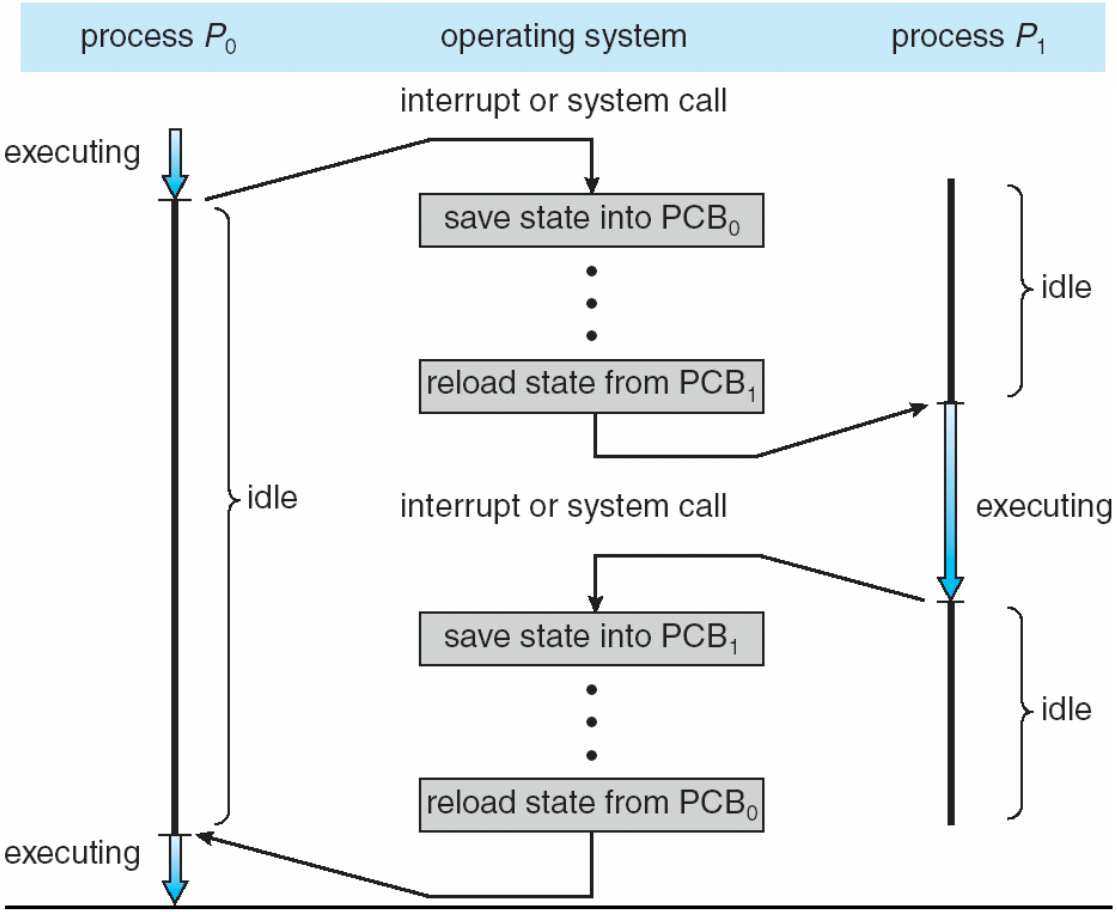
Süreç Denetim Bloğu (PCB)

Süreçle ilgili bilgi (**task control block – görev denetim bloğu**)

- Süreç durumu – çalışıyor, bekliyor, vb.
- Program sayacı – bundan sonra yürütülecek komutun konumu
- CPU registerları – bütün süreç-merkezli registerların içerikleri
- CPU planlama bilgisi - öncelikler, planlama kuyruğu işaretçileri
- Bellek yönetim bilgisi – sürece ayrılan bellek
- Muhasebe bilgisi – CPU kullanımı, başlangıçtan beri geçen süre, süre sınırları
- I/O durum bilgisi – sürece ayrılan I/O cihazları, açık dosya listesi



CPU'nun Süreçten Sürece Geçişi



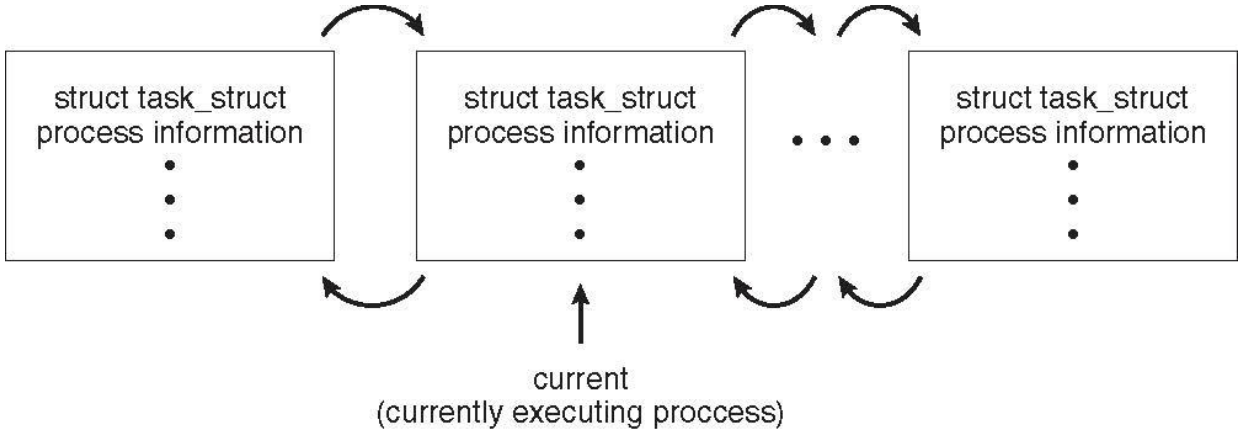
İş Parçaları (Threads)

- Şimdiye kadar, süreç tek parça halinde yürütülüyor.
- Süreç başına birden çok program sayacı olsa
  - Birden fazla konum aynı anda yürütülebilir
    - Birden fazla denetim ayağı -> **threads**
- İş parça ayrıntıları için depo, birden çok program sayacı vs. gerekir.
- Sonraki bölüm…

Süreç Temsili (Linux)

C structure `task_struct`

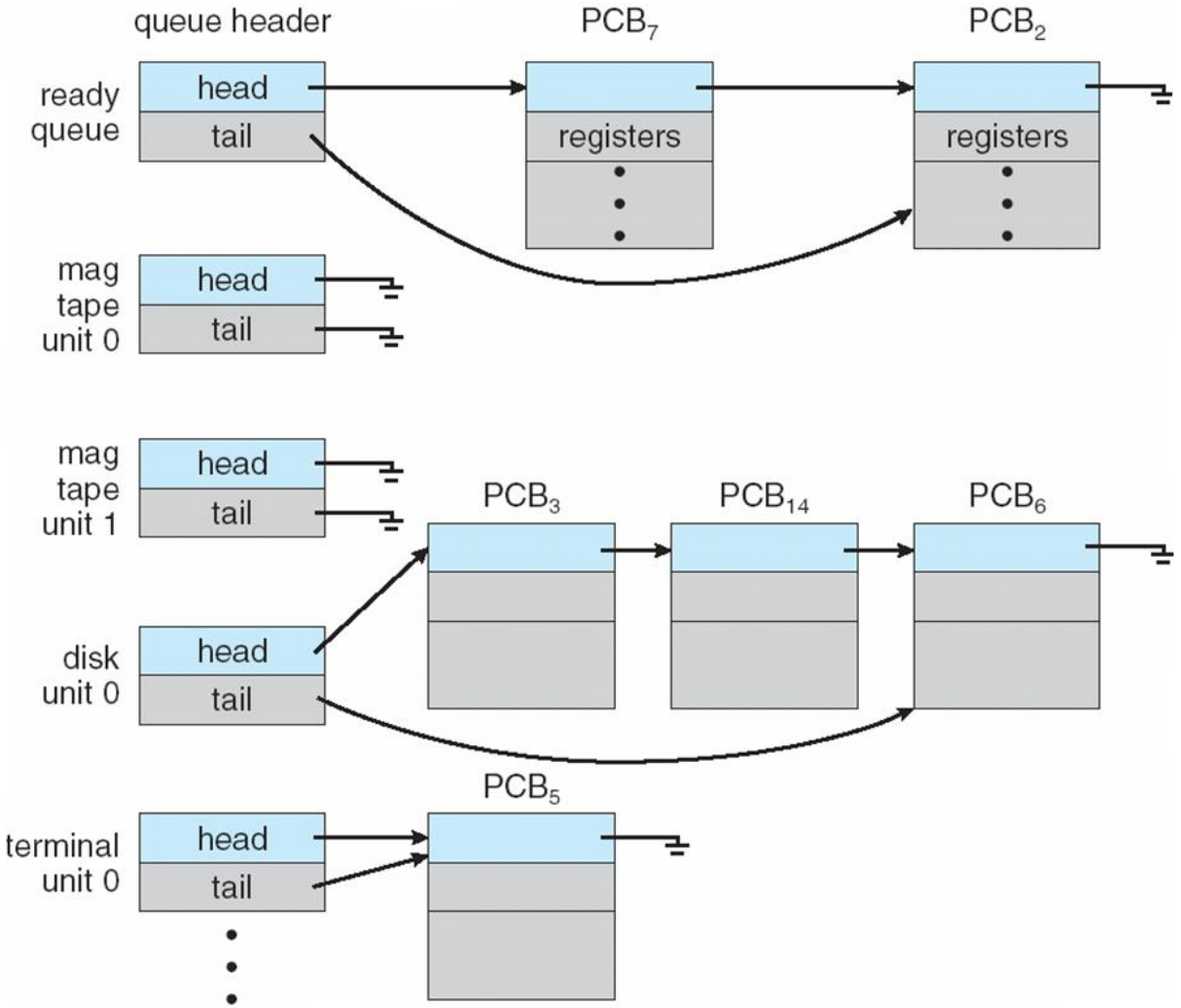
```
pid t_pid; /* süreç id */
long state; /* süreç durumu */
unsigned int time_slice /* planlama bilgisi */
struct task_struct *parent; /* sürecin ebeveyni*/
struct list_head children; /* sürecin çocukları*/
struct files_struct *files; /* açık dosya listesi */
struct mm_struct *mm; /* sürecin adres alanı */
```



Süreç Planlama

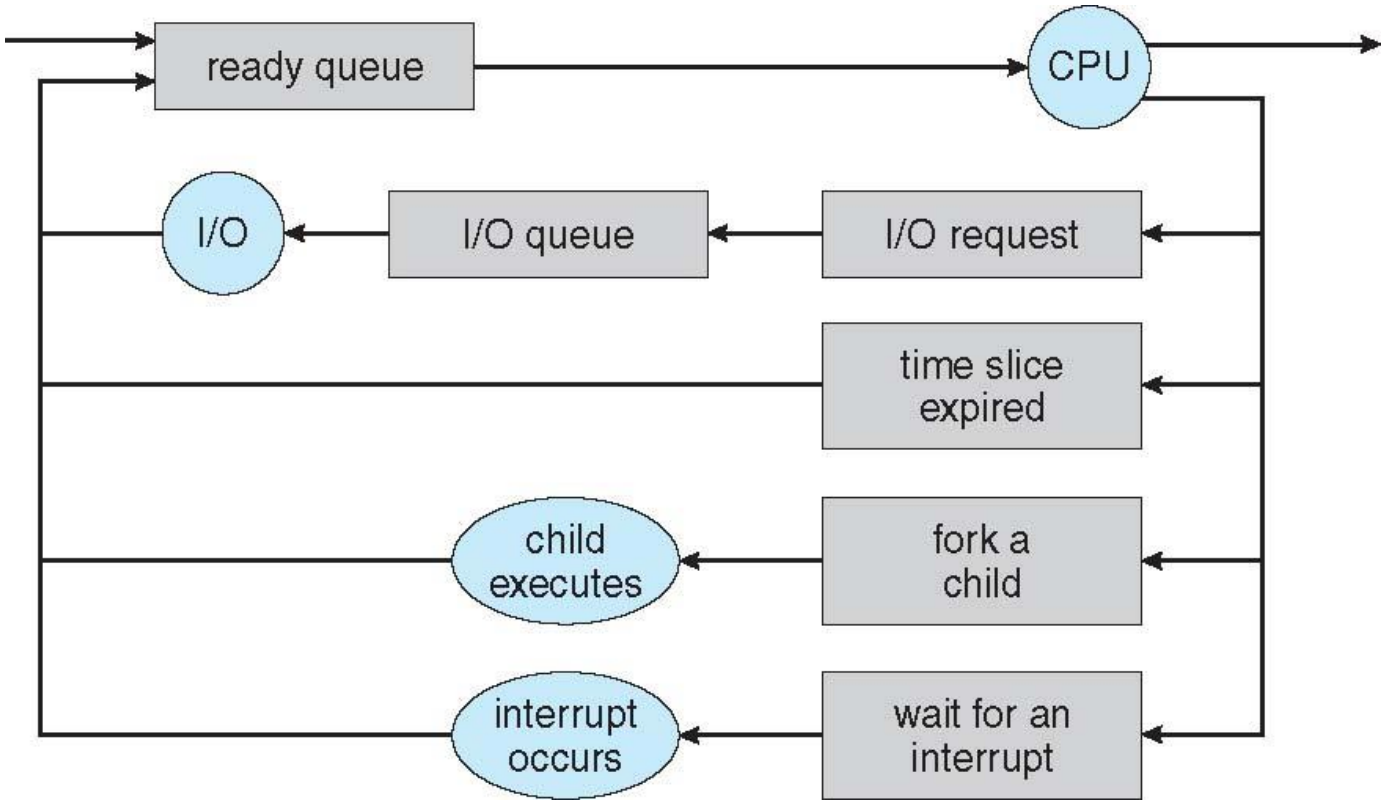
- **Amaç:** CPU kullanımını en yükseğe çıkarmak için CPU'da çalışan süreçleri hızlı hızlı değiştirip zaman paylaşımı sağlamak
- **Süreç planlayıcı (process scheduler)** uygun süreçler arasından seçim yaparak CPU'da yürütülecek sonraki süreci belirler.
- Süreçlere ait **planlama kuyrukları (scheduling queues)** tutulur.
  - **İş kuyruğu** – sistemdeki tüm süreçlerin kümesi
  - **Hazır kuyruğu** – ana bellekte duran ve yürütülmeyi bekleyen tüm süreçlerin kümesi
  - **Cihaz kuyrukları** – I/O cihazı bekleyen süreçlerin kümesi
  - Süreçler farklı kuyruklar arasında hareket halindedir.

Hazır Kuyruğu ve Çeşitli I/O Cihaz Kuyrukları



Süreç Planlama

- **Kuyruk diyagramı** kuyrukları, kaynakları, akışları temsil eder.

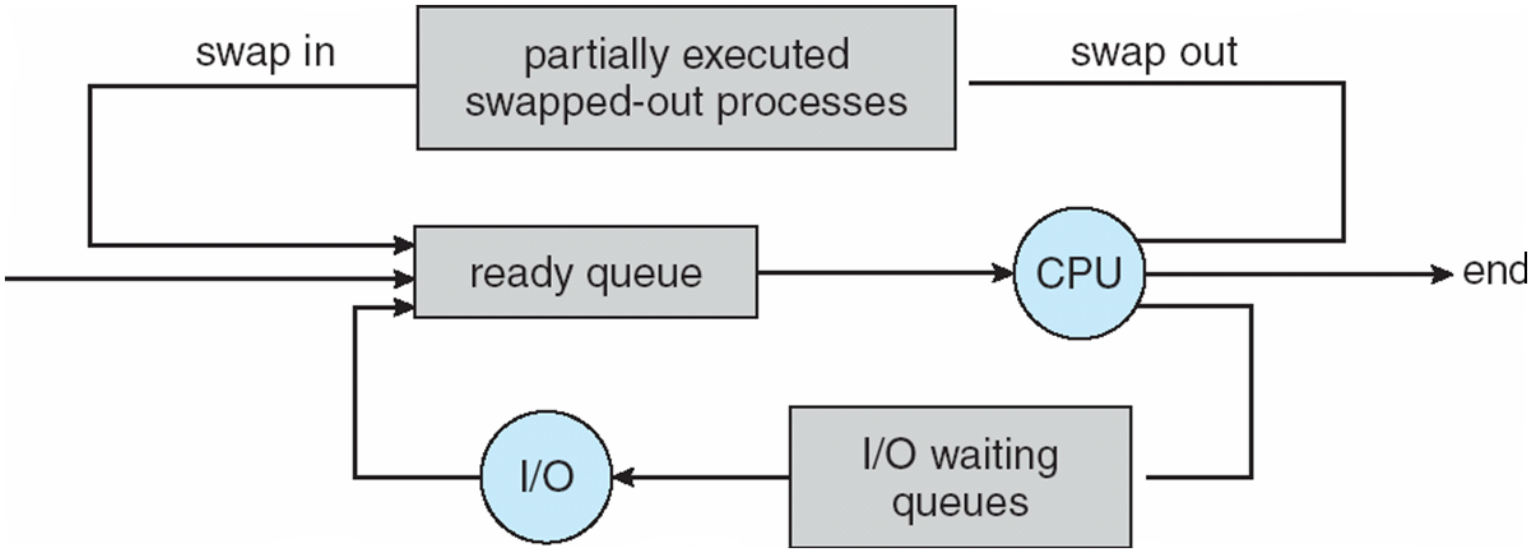


Planlayıcılar

- **Kısa vadeli planlayıcı (Short-term scheduler)** (veya **CPU planlayıcı**) – yürütülecek bir sonraki süreci seçer ve CPU ayırır.
  - Bazen sistemdeki tek planlayıcıdır
  - Kısa vadeli planlayıcı sıkça çağrılır (milisaniye) ⇒ (hızlı olmalı)
- **Uzun vadeli planlayıcı** (veya **iş planlayıcı**) – hangi süreçlerin hazır kuyruğuna sokulacağını seçer.
  - Uzun vadeli planlayıcı daha az sıklıkla çağrılır (saniye, dakika) ⇒ (yavaş olabilir)
  - Uzun vadeli planlayıcı **çoklu programlama derecesini kontrol eder.**
- Süreçler iki şekilde tarif edilebilir:
  - **I/O-sınırlamalı süreç** –I/O için hesaplamadan daha fazla zaman harcar, birçok kısa CPU burst
  - **CPU-sınırlamalı süreç** – hesaplama için daha fazla zaman harcar; az sayıda uzun CPU burst
- Uzun vadeli planlayıcı iyi bir ***süreç karışımı*** için uğraşır.

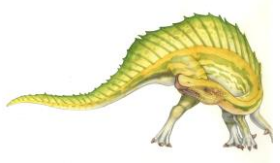
Ekleme - Orta Vadeli Planlama

- **Orta vadeli planlayıcı** - çoklu programlama derecesi azaltılmak istenirse eklenebilir.
  - Süreci bellekten çıkar, diske koy, diskten geri getirerek yürütmeyi sürdür: **swapping (değiş-tokuş)**



Bağlam Değiştirme (Context Switch)

- CPU bir sürece geçtiğinde, sistem eski sürecin durumunu (state) kaydedip yeni sürecin kayıtlı durumunu yüklemeli - **context switch (bağlam değiştirme)**
- **Sürecin bağlamı** PCB'de temsil edilir.
- Bağlam değiştirme zamanı bir ek yüktür. Sistem bu sırada başka bir iş yapmaz.
  - OS ve PCB karmaşıklaştıkça bağlam değiştirme daha uzun sürer.
- Zaman donanım desteğine de bağlıdır.
  - Bazı donanımlar her CPU'ya birkaç küme register verir ➔ böylece aynı anda birden fazla bağlam yüklenebilir.

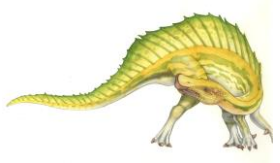


Süreç İşlemleri

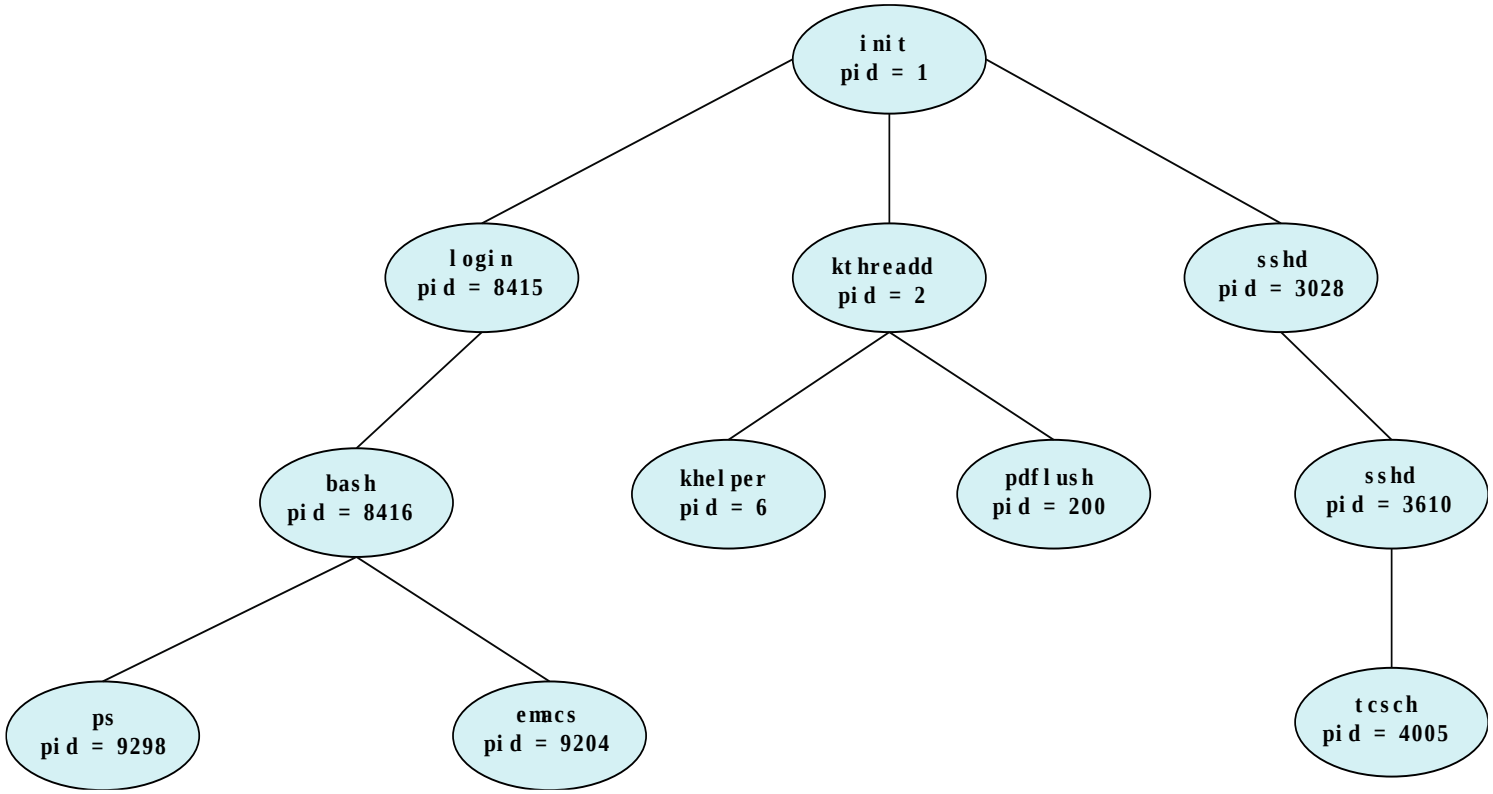
- Sistem aşağıdakiler için mekanizma sunmalı:
  - süreç oluşturma,
  - süreç sonlandırma,
  - ve ayrıntıları daha sonra verilecek işlerl

Süreç Oluşturma

- **Ebeveyn** süreç **çocuk** süreçleri oluşturur, onlar da başka süreçler oluşturur, böylece bir **süreç ağacı** oluşur.
- Genellikle, süreç bir **süreç tanımlayıcı (process identifier – pid)** ile yönetilir.
- Kaynak paylaşım seçenekleri
  - Ebeveyn ve çocuk bütün kaynakları paylaşır.
  - Çocuklar ebeveynin kaynaklarının bir kısmını paylaşır.
  - Ebeveyn ve çocuk hiçbir kaynağı paylaşmaz.
- Yürütme seçenekleri
  - Ebeveyn ve çocuk eşzamanlı olarak yürütülürler.
  - Ebeveyn çocuğun sonlanmasını bekler.

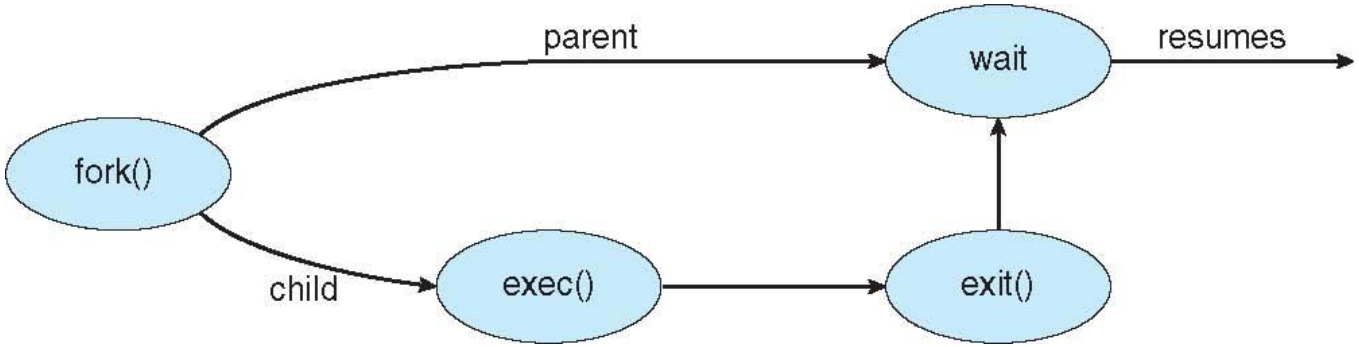


Linux'ta bir Süreç Ağacı



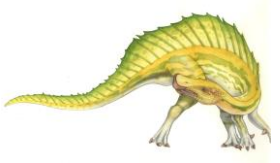
Süreç Oluşturma

- Adres alanı
  - Çocuk ebeveynin kopyası, veya
  - Çocuk içine bir program yüklenir
- UNIX örnekleri
  - **`fork()`** sistem çağrısı yeni süreç oluşturur.
  - **`exec()`** sistem çağrısı **`fork()`** sonrasında sürecin bellek alanını yeni programla değiştirir.



C Programının Ayrı Süreç Oluşturması

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>

int main()
{
pid_t pid;

    /* fork a child process */
    pid = fork();

    if (pid < 0) { /* error occurred */
       fprintf(stderr, "Fork Failed");
       return 1;
    }
    else if (pid == 0) { /* child process */
       execlp("/bin/ls","ls",NULL);
    }
    else { /* parent process */
       /* parent will wait for the child to complete */
       wait(NULL);
       printf("Child Complete");
    }

    return 0;
}
```

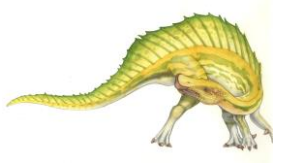


Windows API ile Ayrı Süreç Oluşturma

```
#include <stdio.h>
#include <windows.h>

int main(VOID)
{
STARTUPINFO si;
PROCESS_INFORMATION pi;

    /* allocate memory */
    ZeroMemory(&si, sizeof(si));
    si.cb = sizeof(si);
    ZeroMemory(&pi, sizeof(pi));

    /* create child process */
    if (!CreateProcess(NULL, /* use command line */
     "C:\\WINDOWS\\system32\\mspaint.exe", /* command */
     NULL, /* don't inherit process handle */
     NULL, /* don't inherit thread handle */
     FALSE, /* disable handle inheritance */
     0, /* no creation flags */
     NULL, /* use parent's environment block */
     NULL, /* use parent's existing directory */
     &si,
     &pi))
    {
      fprintf(stderr, "Create Process Failed");
      return -1;
    }
    /* parent will wait for the child to complete */
    WaitForSingleObject(pi.hProcess, INFINITE);
    printf("Child Complete");

    /* close handles */
    CloseHandle(pi.hProcess);
    CloseHandle(pi.hThread);
}
```

Süreç Sonlandırma

- Süreç son deyimini yürütür ve OS'e kendisini silmesini söyler - **`exit()`** sistem çağrısı ile.
  - Ebeveyn sürece durum verisi döndürür (**`wait()`** ile)
  - Süreç kaynakları OS tarafından geri alınır.
- Ebeveyn, çocuk süreçlerin yürütülmesini **`abort()`** sistem çağrısı ile sonlandırabilir. Bazı olası sebepler şunlardır:
  - Çocuk kendine ayrılmış kaynakları aştıysa,
  - Çocuğa verilen iş artık gerekli değilse,
  - Ebeveyn exit ile çıkış yapıyorsa (OS çocuğun devam etmesine izin vermez).

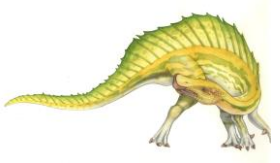


Süreç Sonlandırma

- Bazı OS'iler ebeveyn sonlanmışsa çocuğun devam etmesine izin vermez. Süreç sonlanırsa bütün çocukları da sonlanmalıdır.
  - **Zincirleme sonlanma.** Bütün çocuklar, torunlar vs. sonlanır.
  - Sonlandırmayı OS başlatır.
- Ebeveyn süreç çocuk sürecin bitişini **`wait()`** sistem çağrısı kullanarak bekleyebilir. Çağrı, durum bilgisini ve sonlanan sürecin pid'sini döndürür.

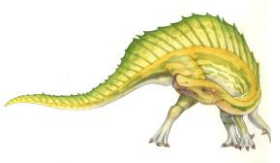
```
pid = wait(&status);
```

- Çocuk süreç bittiğinde bekleyen ebeveyn yoksa (**`wait()`** çağrılmamışsa), süreç **zombi.**
- Ebeveyn **`wait`** yapmadan sonlanmışsa, süreç **yetim.**



Çoklu Süreç Mimarisi – Chrome Browser

- Birçok web tarayıcı tek süreç olarak çalışırdı.
  - Bir web sitesinde sorun çıksa tüm tarayıcı takılabilir/çökebilir.
- Google Chrome'da 3 farklı tür süreç vardır:
  - **Tarayıcı** süreci kullanıcı arayüzünü, disk ve ağ I/O'sunu yönetir.
  - **İşleyici (renderer)** süreci web sayfalarını görüntülenmeye hazır hale getirir, HTML ve Javascript ile ilgilenir. Açılan her web sayfası için yeni bir işleyici oluşturulur.
    - **Sandbox** içinde çalışması disk ve ağ I/O'sunu kısıtlar, güvenlik istismarlarının etkisini azaltır.
  - **Eklenti (plug-in)** süreci her eklenti türü için ayrıdır.

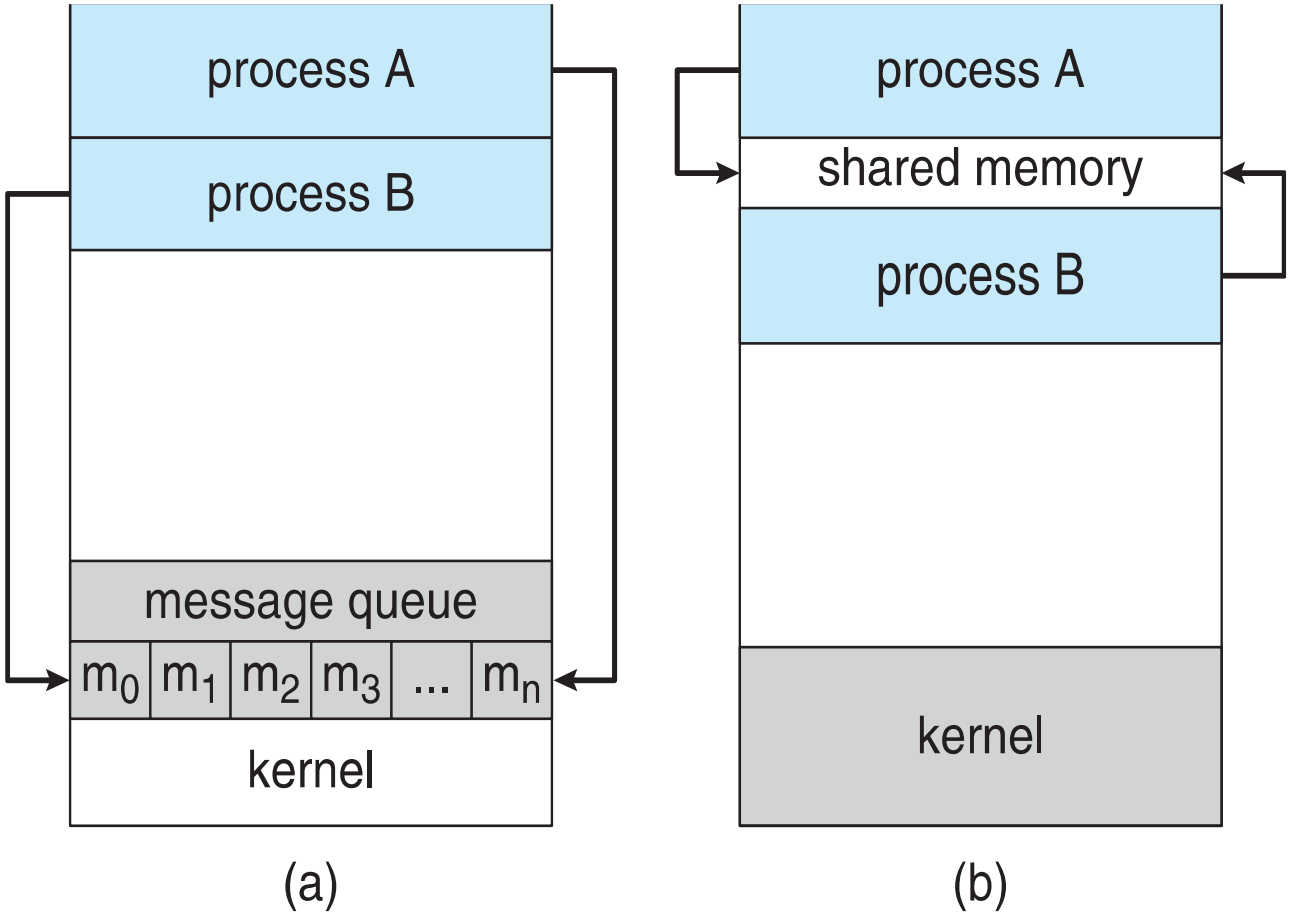


Süreçler Arası İletişim

- Sistemdeki süreçler ***bağımsız*** veya ***işbirliği içerisinde*** olabilir.
- İşbirliği yapan süreç diğerlerin(i)/(den) etkileyebilir/etkilenebilir, örneğin veri paylaşımıyla
- Süreçler neden işbirliği yapar:
  - Bilgi paylaşımı
  - Hesaplamayı hızlandırma
  - Modülerlik
  - Kolaylık
- İşbirliği yapan süreçler, **süreçler arası iletişim** (**IPC**) gereksinimi duyar.
- İki IPC modeli
  - **Paylaşılan bellek**
  - **Mesajlaşma**

İletişim Modelleri

(a) Mesajlaşma. (b) Paylaşılan bellek.



İletişim Modelleri

- Mesajlaşma gerçekleştirimi daha kolay, ama daha yavaş, sistem çağrılarıyla kernel işin içine giriyor.
- Paylaşılan bellek daha hızlı, alan kurulduktan sonra içinde sistem çağrısı yok, normal bellek erişimi.
- Multicore sistemlerde mesajlaşma daha iyi, paylaşılan bellekte cache tutarlılığı sorunu var.

İşbirliği Yapan Süreçler

- ***Bağımsız*** süreç bir başka sürecin yürütülmesini etkilemeyen ve ondan etkilenmeyen süreçtir.
- ***İşbirliği yapan*** süreç bir başka sürecin yürütülmesini etkileyen ve ondan etkilenen süreçtir.
- Süreç işbirliğinin avantajları:
  - Bilgi paylaşımı
  - Daha hızlı hesaplama
  - Modülerlik
  - Kolaylık

Üretici-Tüketici Problemi

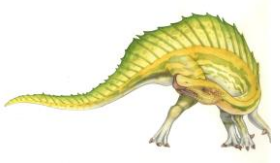
- Üretici sürecin ürettiği bilgiyi tüketici süreç tüketir.
  - **Sınırsız arabellek (unbounded-buffer)** arabellek boyutuna sınır koymaz.
  - **Sınırlı arabellek (bounded-buffer)** sabit bir arabellek boyutu olduğunu varsayar.

Sınırlı Arabellekli Paylaşılan Bellek

- Paylaşılan veri

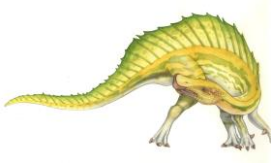
```
#define BUFFER_SIZE 10
typedef struct {
  . . .
} item;

item buffer[BUFFER_SIZE];
int in = 0;
int out = 0;
```

- Çözüm doğru ancak en fazla BUFFER_SIZE-1 eleman kullanılabilir.



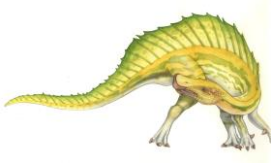
Sınırlı Arabellek – Üretici

```
item next_produced;
while (true) {
        /* produce an item in next produced */
        while (((in + 1) % BUFFER_SIZE) == out)
                ; /* do nothing */
        buffer[in] = next_produced;
        in = (in + 1) % BUFFER_SIZE;
}
```



Sınırlı Arabellek – Tüketici

```
item next_consumed;

while (true) {
        while (in == out)
                ; /* do nothing */
        next_consumed = buffer[out];

        out = (out + 1) % BUFFER_SIZE;

        /* consume the item in next consumed */

}
```



Süreçler Arası İletişim – Paylaşılan Bellek

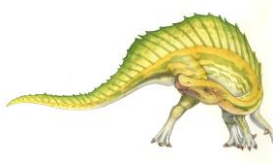
- İletişim kurmak isteyen süreçler arasında paylaşılan bellek alanı
- İletişim kullanıcı süreçlerinin denetimindedir, OS'in değil.
- Kullanıcı süreçlerinin paylaşılan belleğe erişirken eylemlerini senkronize etmesine yarayan mekanizmalar sunmak önemli bir problemdir.
- Senkronizasyon 5. Bölümde anlatılacak.

Süreçler Arası İletişim – Mesajlaşma

- Süreçlerin iletişim kurması ve eylemlerini senkronize etmesi için bir mekanizma.
- Mesaj sistemi – süreçler paylaşılan değişkenlere ihtiyaç duymadan haberleşebilir.
- IPC iki işlem sunar:
  - **`send`**(*mesaj*) - gönder
  - **`receive`**(*mesaj*) - al
- Mesaj boyutu sabit veya değişken olabilir.

Mesajlaşma

- *P* ve *Q* süreçleri haberleşmek isterse, şunları yapmaları gerekir:
  - Aralarında ***iletişim bağı*** kurmak
  - Gönder/al ile mesajlaşmak
- Gerçekleştirim sorunları:
  - Bağlar nasıl kurulur?
  - Bir bağı ikiden fazla süreç kullanabilir mi?
  - Her süreç çifti arasında kaç bağ olabilir?
  - Bağın kapasitesi nedir?
  - Bağda gönderilen mesaj boyutu sabit mi değişken mi?
  - Bağ tek yönlü mü çift yönlü mü?

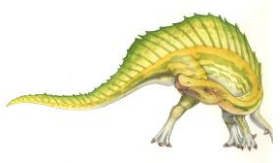


Mesajlaşma

- İletişim bağının gerçekleştirimi
  - Fiziksel:
    - Paylaşılan bellek
    - Donanım veri yolu
    - Ağ
  - Mantıksal:
    - Doğrudan veya dolaylı
    - Eşzamanlı veya eşzamansız
    - Otomatik veya açık arabellek

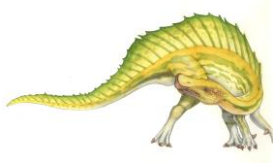
Doğrudan İletişim

- Süreçler birbirlerinin adlarını açıkça kullanırlar:
  - **`send`** (*P*, *mesaj*) –P sürecine mesaj gönder
  - **`receive`**(*Q*, *mesaj*) – Q sürecinden mesaj al
- İletişim bağının özellikleri:
  - Bağlar otomatik kurulur.
  - Her bağ tam olarak bir çift sürece aittir.
  - Her süreç çifti tam olarak bir bağla bağlıdır.
  - Bağ tek yönlü olabilir ama çoğunlukla çift yönlüdür.



Dolaylı İletişim

- Mesajlar mesaj kutularına yollanır ve oradan alınır (bunlara kapı veya port da denir)
  - Her mesaj kutusunun eşsiz id'si vardır.
  - Süreçler ancak bir mesaj kutusu paylaşırlarsa haberleşebilir.
- İletişim bağının özellikleri:
  - Bağ ancak süreçler arasında paylaşılan bir mesaj kutusu varsa kurulur.
  - Bir bağ birçok süreçle ilişkili olabilir.
  - Her süreç çifti birden fazla bağla bağlı olabilir.
  - Bağ tek yönlü veya çift yönlü olabilir.



Dolaylı İletişim

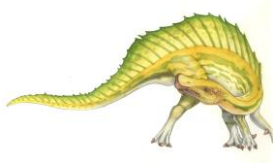
- İşlemler
  - Yeni bir mesaj kutusu (port) oluştur
  - Mesaj kutusu yoluyla mesajları gönder/al
  - Mesaj kutusunu yok et
- Temel işlemler:

  `send`(*A, mesaj*) – A kutusuna mesaj gönder

  `receive`(*A, mesaj*) – A kutusundan mesaj al

Dolaylı İletişim

- Mesaj kutusu paylaşımı
  - P_1, P_2, ve P_3 mesaj kutusu A'yı paylaşıyor.
  - P_1 gönderme; P_2 ve P_3 alma işlemi yapıyor.
  - Mesajı kim alır?
- Çözümler
  - Bir bağ en fazla iki sürece ait olsun
  - Bir anda sadece bir süreç alma işlemi yapabilsin
  - Sistem alıcıyı rastgele seçsin ve göndericiye bilgi versin



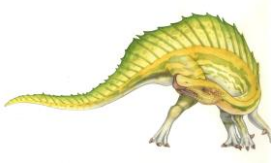
Senkronizasyon

- Mesajlaşma, engelleyen veya engellemeyen nitelikte olabilir.
- **Engelleyen (blocking), eşzamanlı (synchronous)** kabul edilir.
  - **Engelleyen gönderim** – Gönderici, mesaj alınana kadar engellenir.
  - **Engelleyen alım** – Alıcı, mesaj hazır olana kadar engellenir.
- **Engellemeyen (non-blocking), eşzamansız (asynchronous)** kabul edilir.
  - **Engellemeyen gönderim** – Gönderici mesajı gönderir ve devam eder.
  - **Engellemeyen alım** -- Alıcı:
    - Geçerli bir mesaj alır, veya
    - Geçersiz/tanımsız (null) mesaj alır.
- Farklı kombinasyonlar mümkün
  - Eğer hem gönderici hem alıcı engelleyen ise, **randevu (rendezvous)** olur.

Senkronizasyon

- Randevu durumunda üretici-tüketici problemi çok kolay hale gelir:

```
message next_produced;
while (true) {
    /* produce an item in next produced */
send(next_produced);
}

message next_consumed;
while (true) {
   receive(next_consumed);

   /* consume the item in next consumed */
}
```

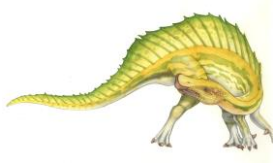


Arabellek Kullanımı

- Bağa mesaj kuyruğu eklenir.
- Üç yoldan biriyle gerçekleştirilir:
  1. Sıfır kapasite – Hiçbir mesaj kuyrukta tutulmaz. Gönderici alıcıyı beklemelidir (randevu)
  2. Sınırlı kapasite – Sonlu uzunluklu kuyruk (*n* mesaj) Bağ doluysa gönderici bekler
  3. Sınırsız kapasite – Sonsuz uzunluklu kuyruk Gönderici hiç beklemez

IPC Sistem Örneği - POSIX

- POSIX'te paylaşılan bellek
  - Süreç ilk olarak paylaşılan bellek bölümünü oluşturur.
    ```
    shm_fd = shm_open(name, O CREAT | O RDWR, 0666);
    ```
  - Var olan bir bölümü paylaşıma açmak için de kullanılır.
  - Nesne boyutu belirlenir
    ```
    ftruncate(shm fd, 4096);
    ```
  - Şimdi süreç paylaşılan belleğe yazabilir
    ```
    sprintf(shared memory, "Writing to shared
    memory");
    ```

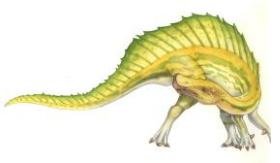


IPC - POSIX Üreticisi

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <sys/shm.h>
#include <sys/stat.h>

int main()
{
/* the size (in bytes) of shared memory object */
const int SIZE = 4096;
/* name of the shared memory object */
const char *name = "OS";
/* strings written to shared memory */
const char *message_0 = "Hello";
const char *message_1 = "World!";

/* shared memory file descriptor */
int shm_fd;
/* pointer to shared memory obect */
void *ptr;

   /* create the shared memory object */
   shm_fd = shm_open(name, O_CREAT | O_RDWR, 0666);

   /* configure the size of the shared memory object */
   ftruncate(shm_fd, SIZE);

   /* memory map the shared memory object */
   ptr = mmap(0, SIZE, PROT_WRITE, MAP_SHARED, shm_fd, 0);

   /* write to the shared memory object */
   sprintf(ptr,"%s",message_0);
   ptr += strlen(message_0);
   sprintf(ptr,"%s",message_1);
   ptr += strlen(message_1);

   return 0;
}
```

IPC - POSIX Tüketicisi

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/shm.h>
#include <sys/stat.h>

int main()
{
/* the size (in bytes) of shared memory object */
const int SIZE = 4096;
/* name of the shared memory object */
const char *name = "OS";
/* shared memory file descriptor */
int shm_fd;
/* pointer to shared memory obect */
void *ptr;

    /* open the shared memory object */
    shm_fd = shm_open(name, O_RDONLY, 0666);

    /* memory map the shared memory object */
    ptr = mmap(0, SIZE, PROT_READ, MAP_SHARED, shm_fd, 0);

    /* read from the shared memory object */
    printf("%s",(char *)ptr);

    /* remove the shared memory object */
    shm_unlink(name);

    return 0;
}
```



IPC Sistem Örneği - Mach

- Mach iletişimi mesaj tabanlıdır.
  - Sistem çağrıları dahi mesajdır.
  - Her görev (task) oluşturulunca iki mesaj kutusu alır - Kernel ve Notify
  - Mesaj iletimi için sadece üç sistem çağrısı gereklidir:

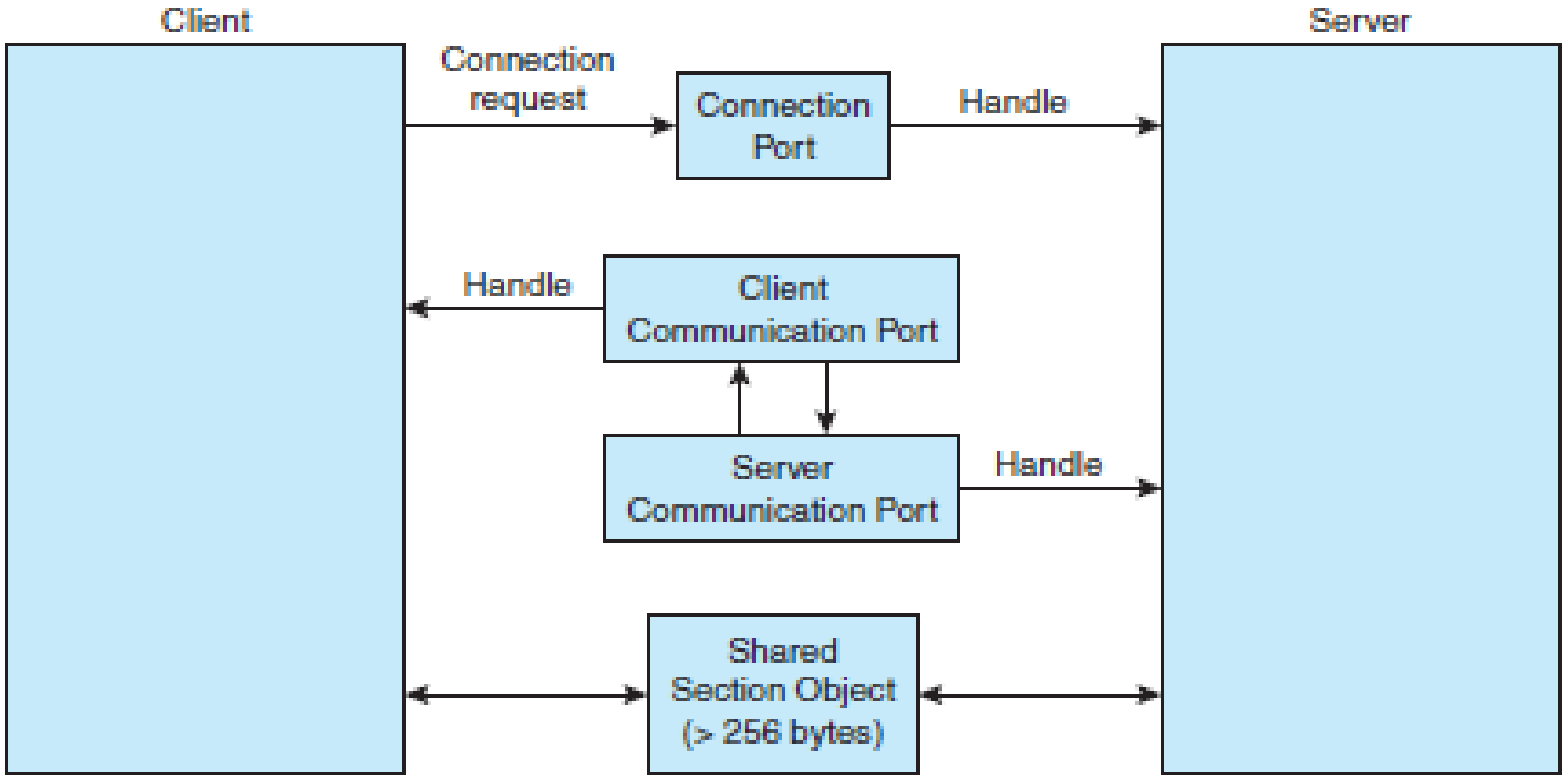
    `msg_send(), msg_receive(), msg_rpc()`
  - Mesaj kutularına ihtiyaç vardır,

    `port_allocate()` çağrısı ile oluşturulur.
  - Gönderme ve alma esnektir, örneğin mesaj kutusu doluysa dört seçenek vardır:
    - Süresiz bekle
    - En çok n milisaniye bekle
    - Hemen dön (return)
    - Mesajı önbellek (cache) içinde sakla.

IPC Sistem Örneği - Windows

- Mesajlaşma merkezli – **ileri yerel yordam çağrısı (advanced local procedure call - ALPC)** yoluyla
  - Sadece aynı sistemdeki süreçler arasında çalışır.
  - İletişim kanalı kurup sürdürmek için port kullanır.
  - İletişim şöyle sağlanır:
    - İstemci, altsistemin **bağlantı kapısı (connection port)** nesnesine handle açar.
    - İstemci bağlantı isteği gönderir.
    - Sunucu iki özel **iletişim kapısı (communication ports)** oluşturur ve birine ait handle'ı istemciye verir.
    - İstemci ve sunucu port handle üzerinden mesaj gönderir ve yanıtları dinler.

Yerel Yordam Çağrıları - Windows



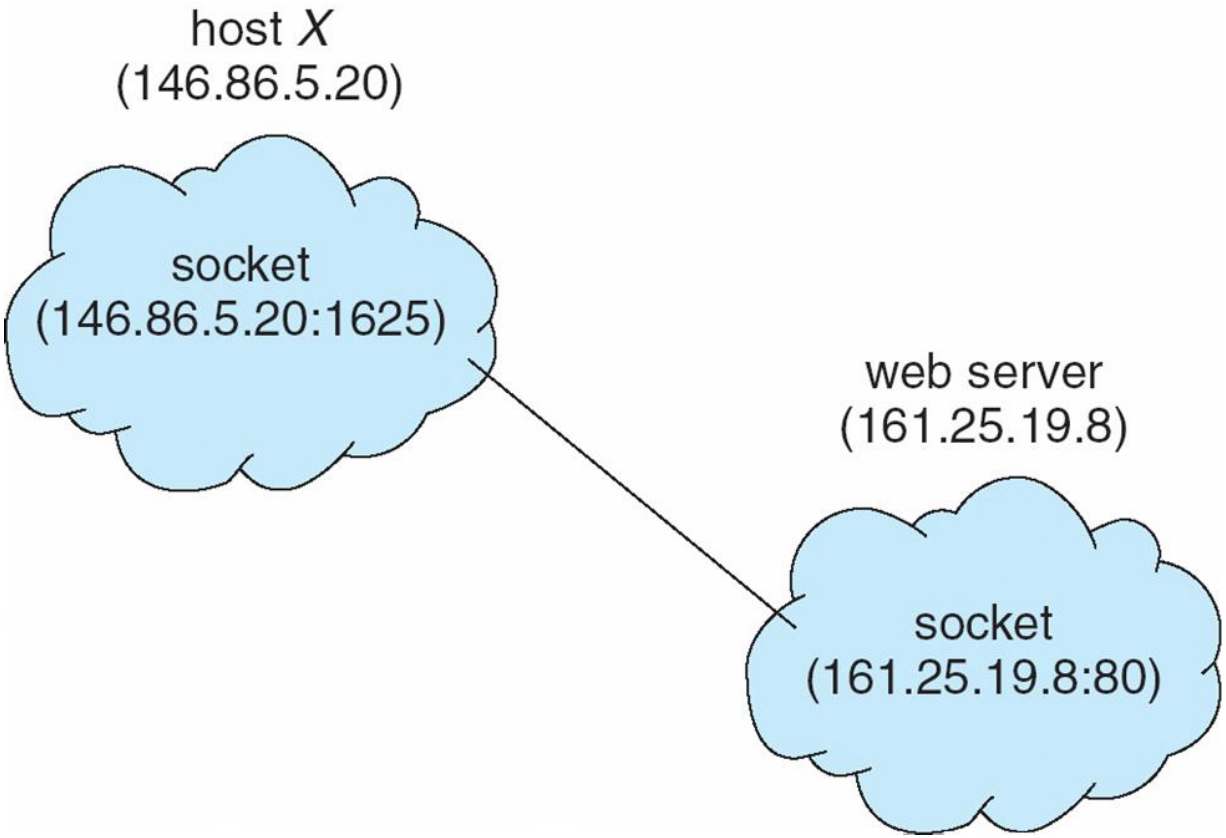
Sunucu-İstemci Sistemlerinde İletişim

- Soketler (sockets)
- Uzaktan yordam çağrıları (Remote procedure calls – RPC)
- Borular (pipes)
- Uzaktan metod çağırma (Java)

Soketler

- Bir **soket** iletişimin uç noktasıdır.
- IP adresi ve **port** – mesajın başlığında yer alan ve bilgisayardaki farklı ağ servislerini ayırt etmeye yarayan sayı
- **161.25.19.8:1625** soketi **161.25.19.8** IP adresine sahip bilgisayardaki **1625** numaralı portu belirtir.
- İletişim soket çifti arasında gerçekleşir.
- 1024'ten küçük portlar standart servisler için kullanılır.
- Özel IP adresi 127.0.0.1 (**loopback**) sürecin üzerinde çalıştığı sistemi belirtir.

Soket İletişimi



Java'da Soketler

- Üç tür soket
  - **Bağlantı tabanlı** (**TCP**)
  - **Bağlantısız** (**UDP**)
  - **`MulticastSocket`** sınıfı – veri birden çok alıcıya gönderilebilir.
- "Date" sunucusu:

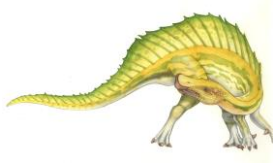
```
import java.net.*;
import java.io.*;

public class DateServer
{
   public static void main(String[] args) {
      try {
         ServerSocket sock = new ServerSocket(6013);

         /* now listen for connections */
         while (true) {
            Socket client = sock.accept();

            PrintWriter pout = new
             PrintWriter(client.getOutputStream(), true);

            /* write the Date to the socket */
            pout.println(new java.util.Date().toString());

            /* close the socket and resume */
            /* listening for connections */
            client.close();
         }
      }
      catch (IOException ioe) {
         System.err.println(ioe);
      }
   }
}
```

Uzaktan Yordam Çağrıları

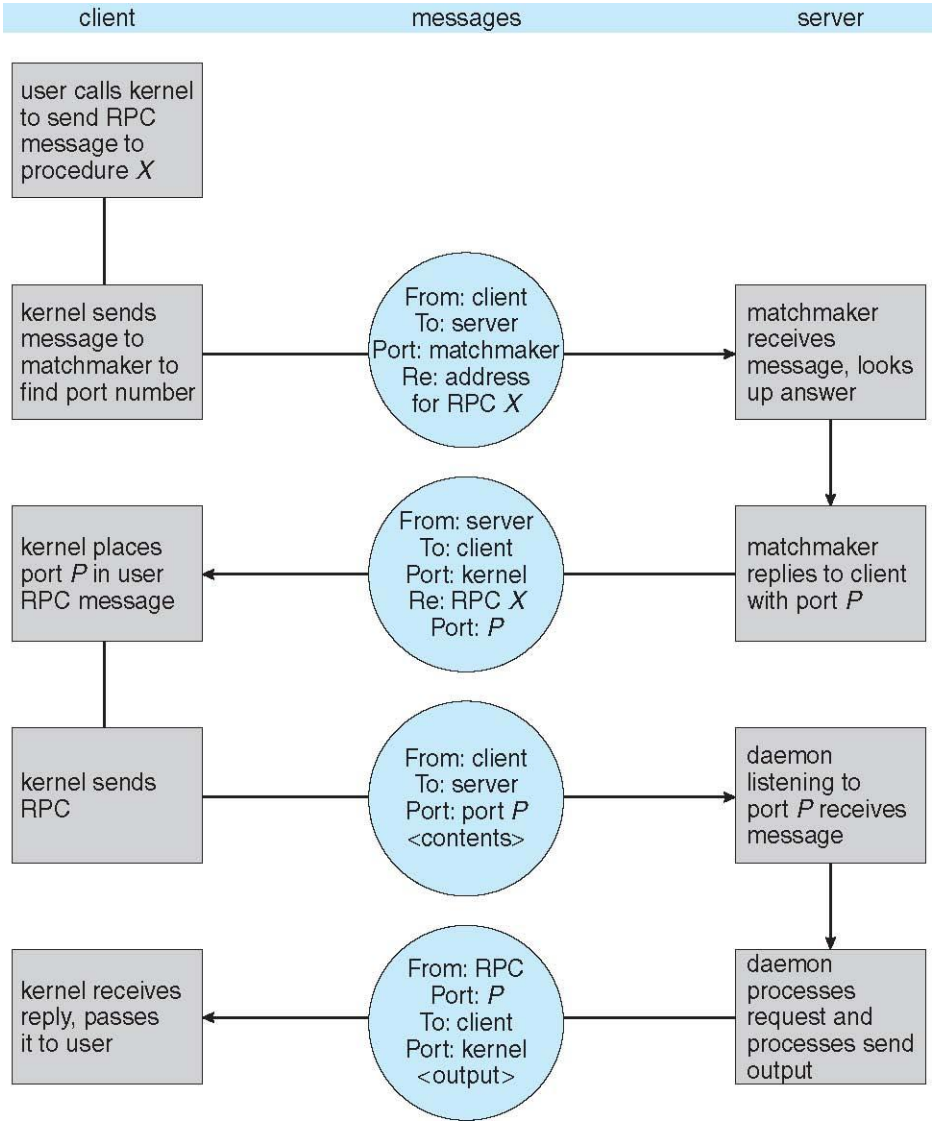
- Remote procedure call (RPC) ağ sistemlerinde süreçler arasındaki yordam çağrılarını soyutlar.
  - Hizmet ayrımı için portlar kullanılır.
- **Stub (koçan)** – sunucudaki asıl yordam için istemci tarafındaki vekil (proxy)
- İstemci tarafındaki koçan, sunucuyu bulur ve parametreleri sıralayarak iletir.
- Sunucu tarafındaki koçan mesajı alır, parametreleri çıkartır, and sunucudaki yordamı çalıştırır.
- Windows'ta, **Microsoft Arayüz Tanımlama Dilinde (Interface Definition Language)** (**MIDL**) yazılı şartnameden koçan kodu derlenir.

Uzaktan Yordam Çağrıları

- Veri temsili Dış Veri Temsili (**External Data Representation - XDL**) formatıyla yapılır ve farklı mimarilere uyumludur.
  - **Big-endian** ve **little-endian**
- Uzaktan iletişimde yerele göre daha çok arıza senaryosu vardır.
  - Mesajların ***tam bir kere (exactly once)*** iletileceği mekanizma gerekir.
- OS genellikle istemciyle sunucuyu bir randevu (**matchmaker**) yoluyla bağlar.



RPC’nin Yürütülmesi

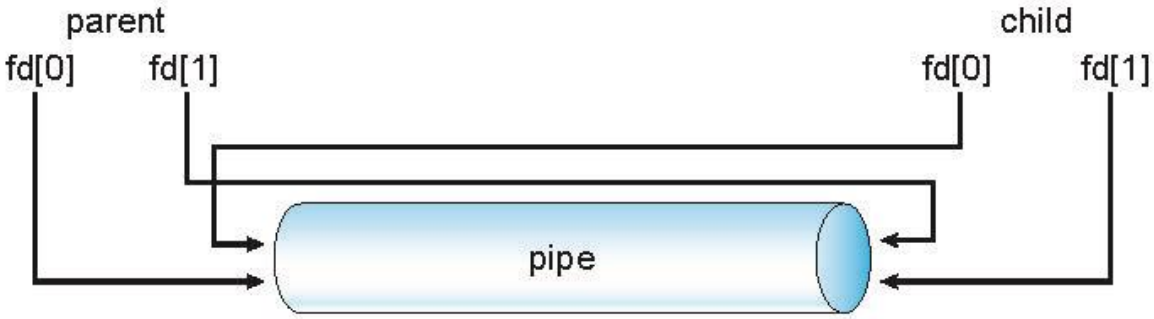


Borular (Pipes)

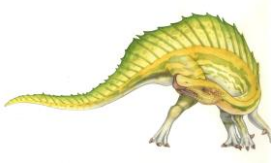
- İki sürecin iletişimini sağlayan bir kanal
- Sorunlar:
  - İletişim tek yönlü mü çift yönlü mü?
  - İki yönlüyse, half-duplex mi full-duplex mi (aynı anda her iki yönde veri akabilir mi)?
  - Haberleşen süreçler arasında bir ilişki (örn. ***ebeveyn-çocuk***) bulunmalı mı?
  - Borular ağ üzerinden kullanılabilir mi?
- Sıradan borular – kendisini oluşturan sürecin dışından erişilemez. Genelde, ebeveyn süreç boruyu oluşturur ve kendi doğurduğu çocuk süreç ile haberleşmede kullanır.
- Adlı borular – ebeveyn-çocuk ilişkisi olmadan erişilebilir.

Sıradan Borular

- Standart üretici-tüketici tarzında iletişim sağlarlar.
- Üretici bir uca yazar (**yazma ucu**)
- Tüketici diğer uçtan okur (**okuma ucu**)
- Sıradan borular tek yönlüdür.
- Ebeveyn-çocuk ilişkisi gereklidir.



- Windows bunlara **anonim borular (anonymous pipes)** adını verir.
- Bkz. Kitaptaki Unix ve Windows kod örnekleri



Adlı Borular

- Sıradan borularda daha güçlü
- Çift yönlü iletişim
- Ebeveyn-çocuk ilişkisine gerek yok
- Birkaç süreç birlikte kullanabilir
- UNIX ve Windows sistemlerinde bulunur

Bölüm 3 Sonu