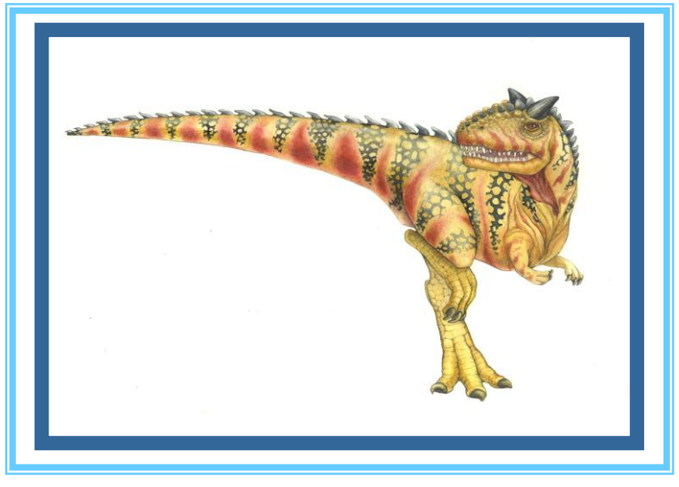
Bölüm 3: Prosesler



Bölüm 3: Prosesler

- Proses Kavramları
- Proses Çizelgeleme
- Proses Üzerinde İşlemler
- Prosesler Arası İletişim
- IPC (Prosesler Arası İletişim) Örnekleri
- İstemci-Sunucu Sistemleri Arasındaki İletişim

Bölüm Hedefleri

- Proses mantığını tanıtmak – çalışan bir program
- Proseslerin yapısını anlatmak (Proseslerin çizelgelenmesi, oluşturulması ve sonlandırılması, iletişimi vb.)
- Paylaşılmış hafıza ve Mesajlaşma ile prosesler arası iletişimi açıklamak
- İstemci – sunucu sistemlerin iletişimini tarif etmek

Proses Kavramları

- Bir işletim sistemi çeşitli programlar yürütür:
  - Batch sistemi – işler (jobs)
  - Zaman paylaşımlı sistemler – kullanıcı programları veya görevler.

- Ders kitaplarında iş/süreç (job / proses) terimleri neredeyse birbirlerinin yerine kullanılır.

- Proses, yürütülen bir programdır denilebilir ve prosesler sıralı bir biçimde yürütülmelidir. (**Process** – a program in execution)

- Bölümleri,
  - **program kodu** (text section),
  - **Mevcut aktiviteler**, **program sayacı** ve diğer kaydedicileri içeren mevcut faaliyetler (PCB).

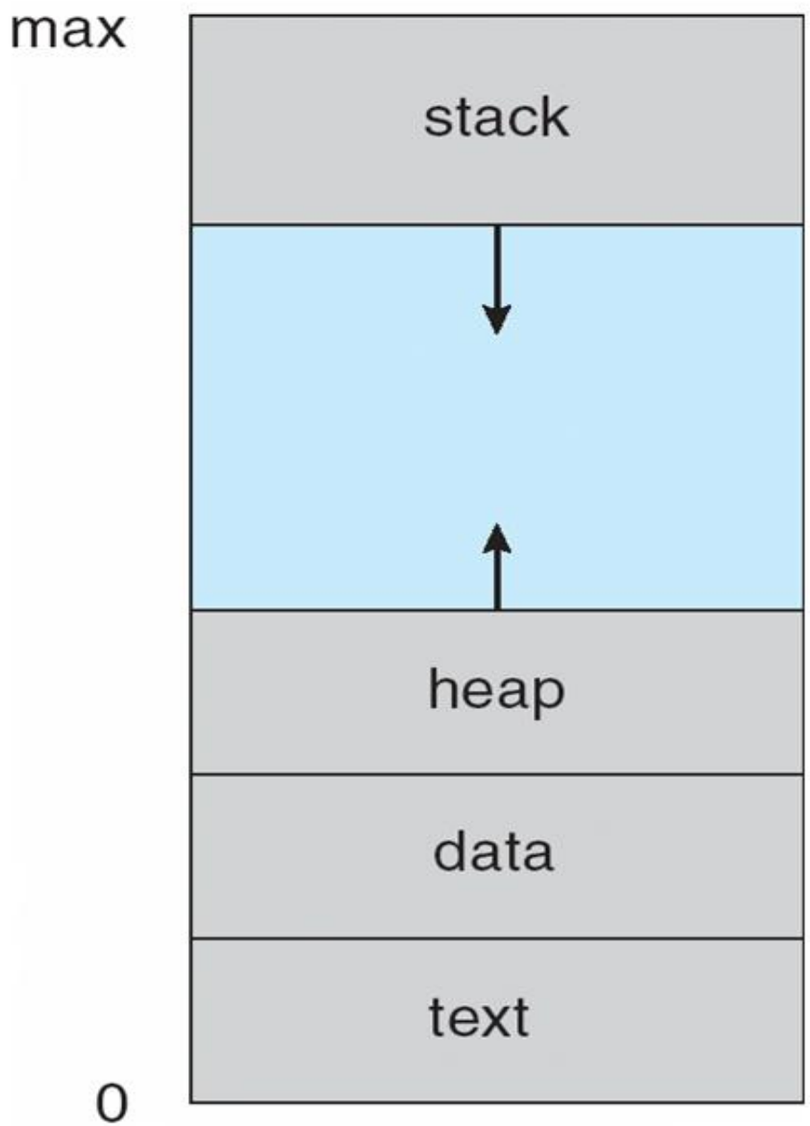
Proses

- **Yığın** (stack) geçici veriyi içerir.
  - Fonksiyon parametreleri, geri dönüş değerleri, yerel değişkenler
- **Veri bölümü** (data section) global değişkenleri içerir.
- **Bellek kümesi-Yığıt** (heap) çalışma zamanında dinamik olarak tahsis edilmiş belleği içerir.

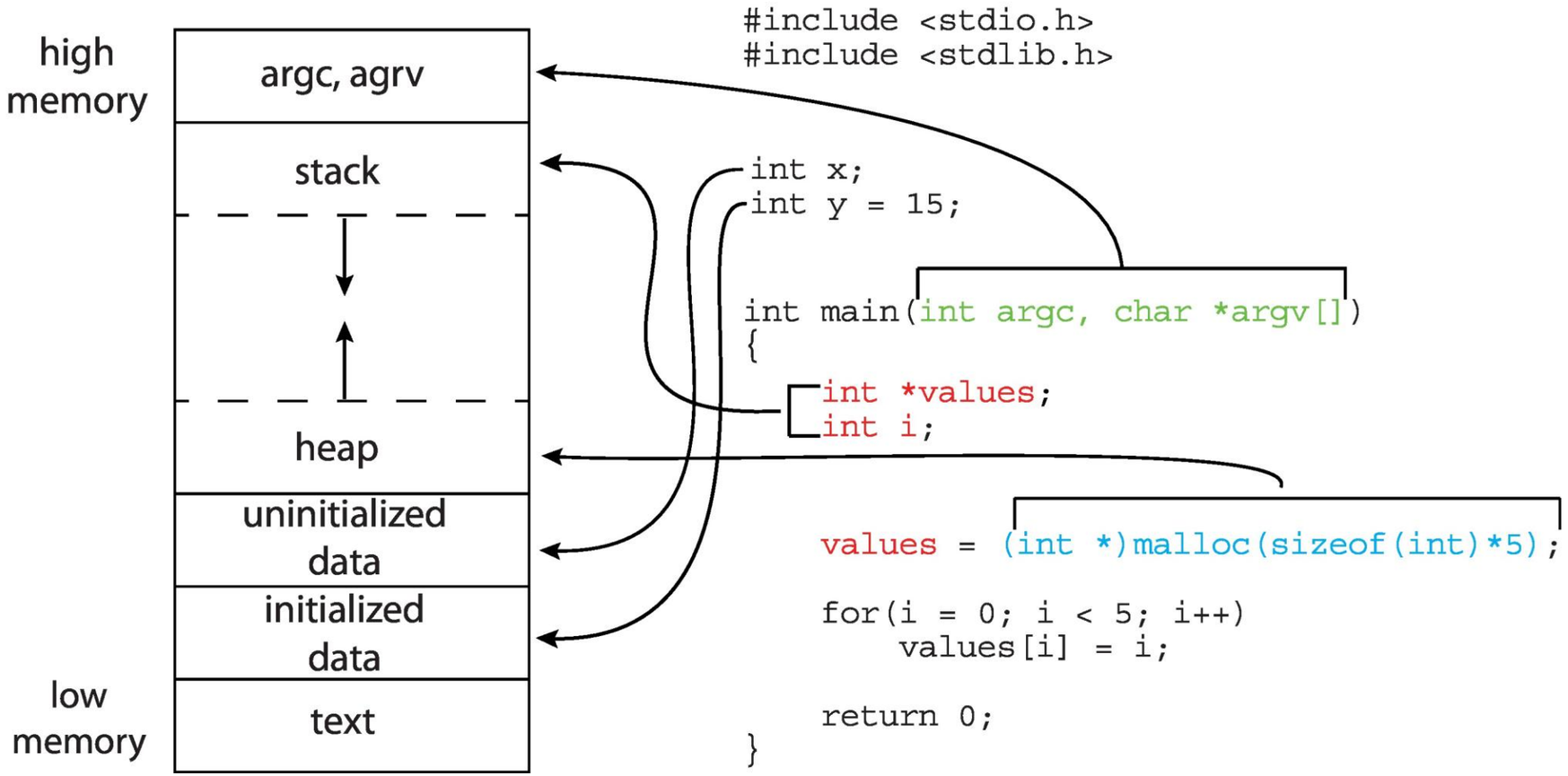
- Program pasif bir varlıktır(disk üzerinde depolanmış), Proses ise aktiftir
  - Program, çalıştırılabilir dosya belleğe yüklendiğinde proses halini alır.

- Programın çalıştırılması komut satırından komutun girilmesi, grafik arayüzde program ikonu üzerine tıklanması vb. ile başlar.
- Bir program birden fazla proses içerebilir.
  - Örneğin bir kullanıcının aynı programı bir çok kere çalıştırması (Birçok web browser açık)

Bellekteki Bir Proses



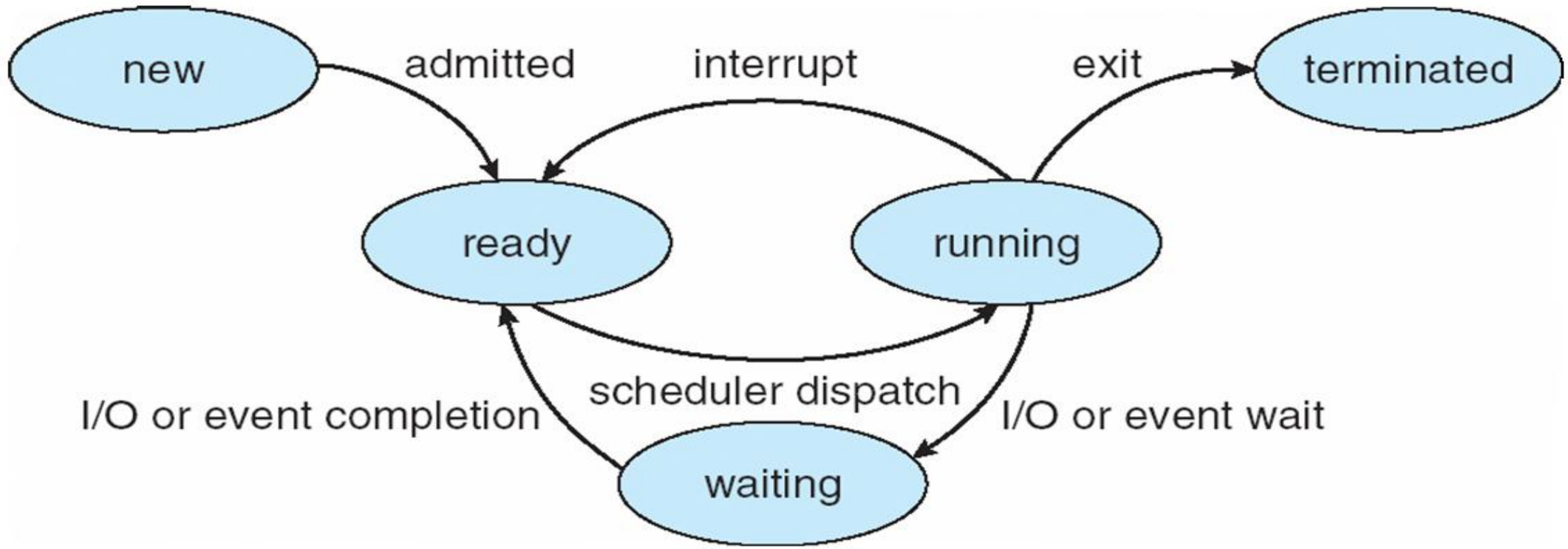
Memory Layout of a C Program



Proses Durumları

- Çalışma anında proseslerin durumları değişir.
  - **yeni**: Yeni bir proses oluşturuluyor.
  - **çalışıyor**: İşlemler gerçekleştiriliyor.
  - **bekleme**: Proses bazı olayların gerçekleşmesini beklemektedir.
  - **hazır**: Proses, işlemciye aktarılmayı beklemektedir.
  - **sonlandırılmış**: Prosesin yürütülmesi tamamlandığını belirtir.

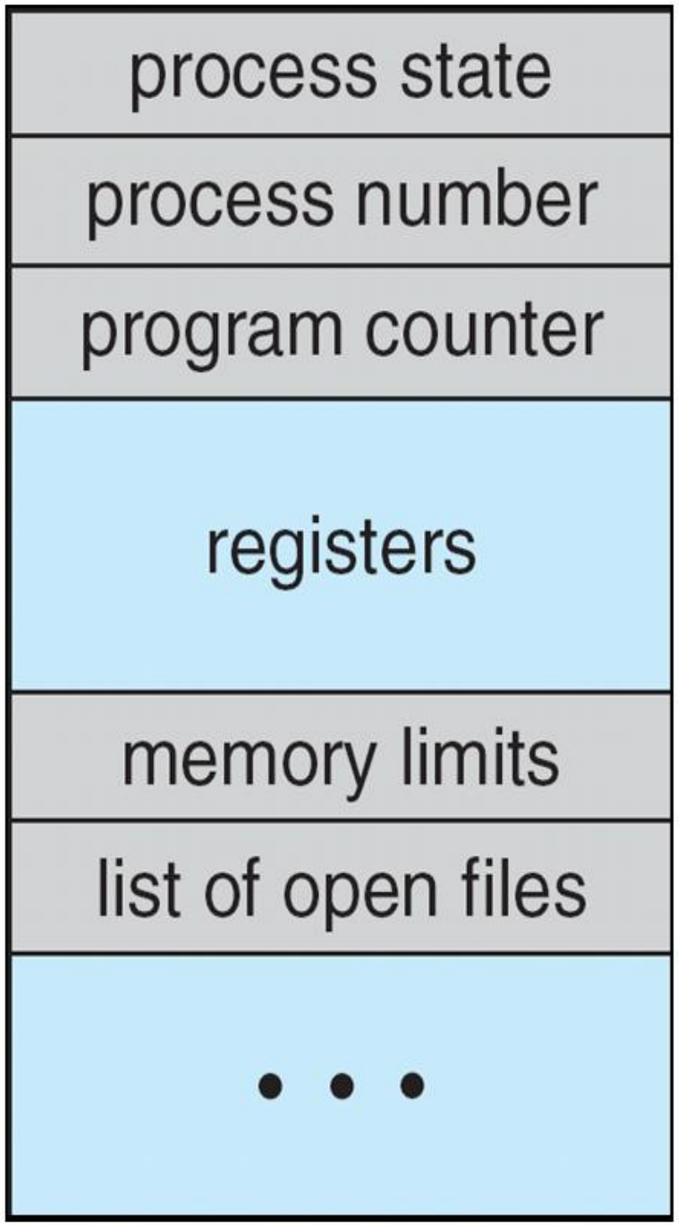
Proses Durum Diyagramı



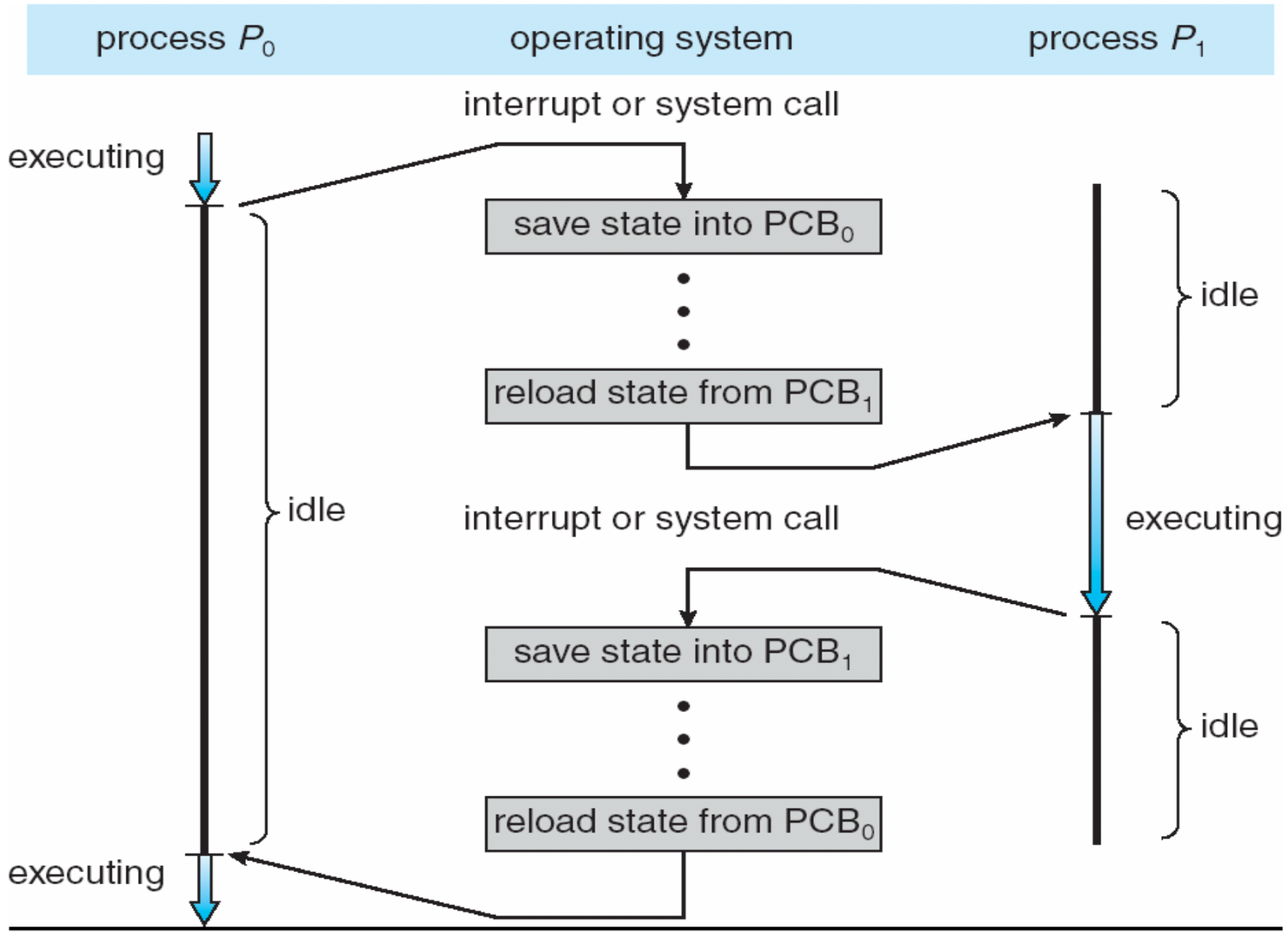
Proses Kontrol Bloğu (PCB)

Her proses ile ilgili bilgileri içerir :

- Proses durumu
  - Bekleme, Çalışıyor
- Program Sayacı
  - Bir sonraki çalıştırılacak komutun adresi
- CPU kaydedicileri
  - bütün proses-merkezli kaydedicilerin içerikleri
- CPU çizelgeleme bilgisi
  - Öncelikler, çizelgeleme kuyruk pointerı
- Bellek yönetim bilgisi
  - Prosese tahsis edilmiş bellek
- Hesap bilgileri
  - Zaman limiti, Kullanılan CPU, baş. itibaren harcanan zaman
- I/O durum bilgisi
  - Prosese tahsis edilmiş I/O aygıtları, açık dosyaların listesi



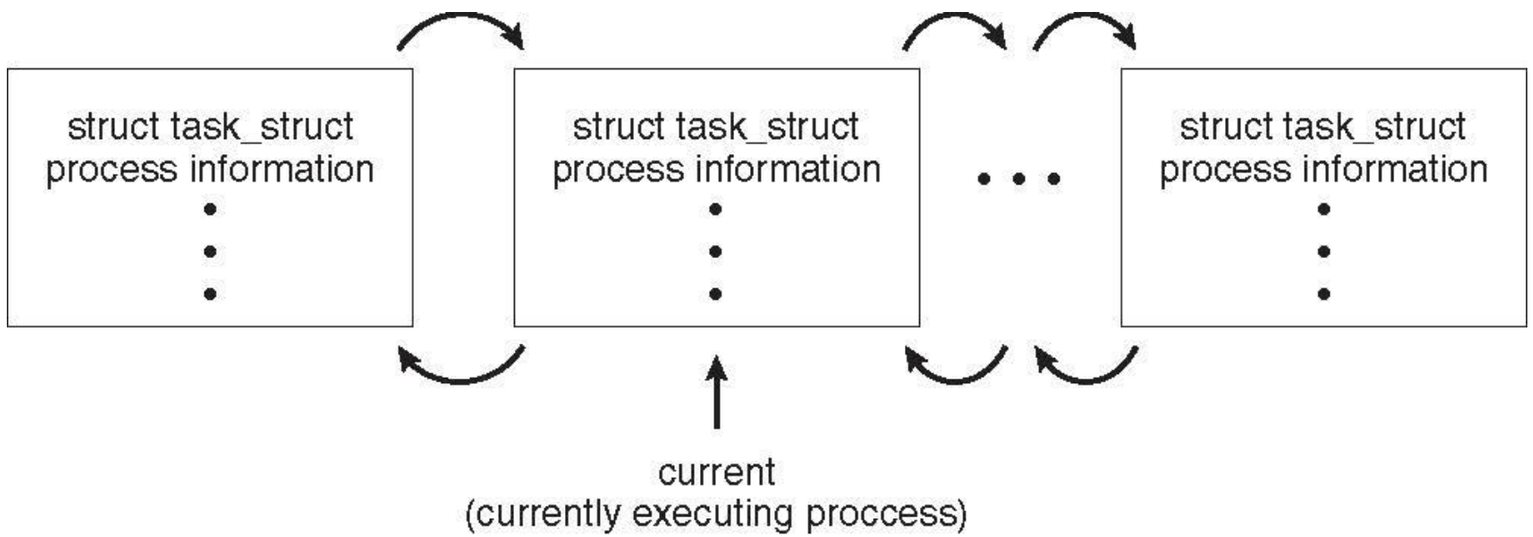
CPU'da Proses Değişimi



Proseslerin Linux'ta Temsili

- C yapısı ile gösterilir (<linux/sched.h> kütüphanesinde)
- task_struct

```
pid t pid; /* Proses tanımlayıcı*/
long state; /* Prosesin durumu*/
unsigned int time slice /* çizelgeleme bilgisi*/
struct task struct *parent; /* bu prosesin ebeveyn'i*/
struct list head children; /* bu prosesin çocuğu*/
struct files struct *files; /* açık dosyaların listesi*/
struct mm struct *mm; /* address space of this process */
```

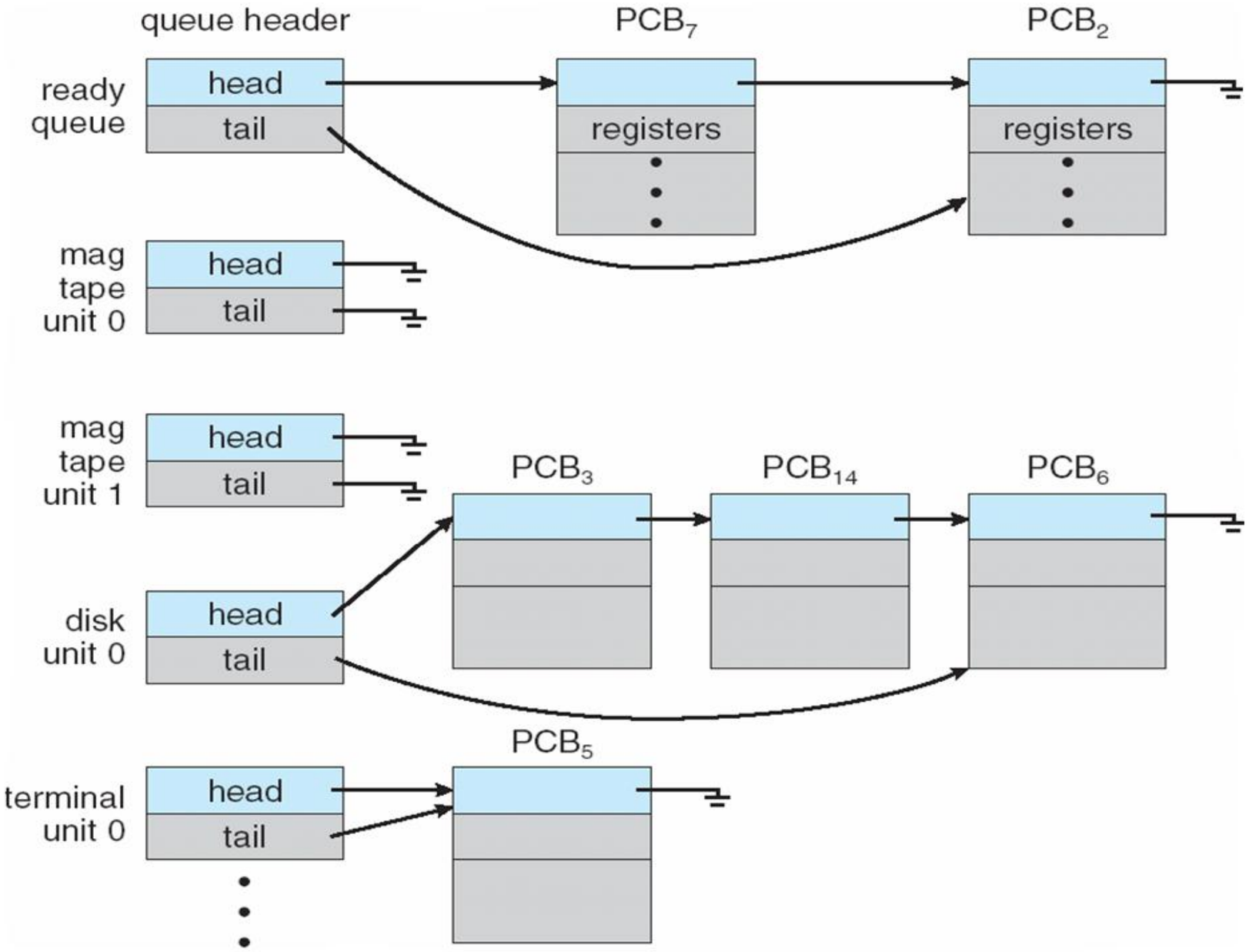


current->state = new state;

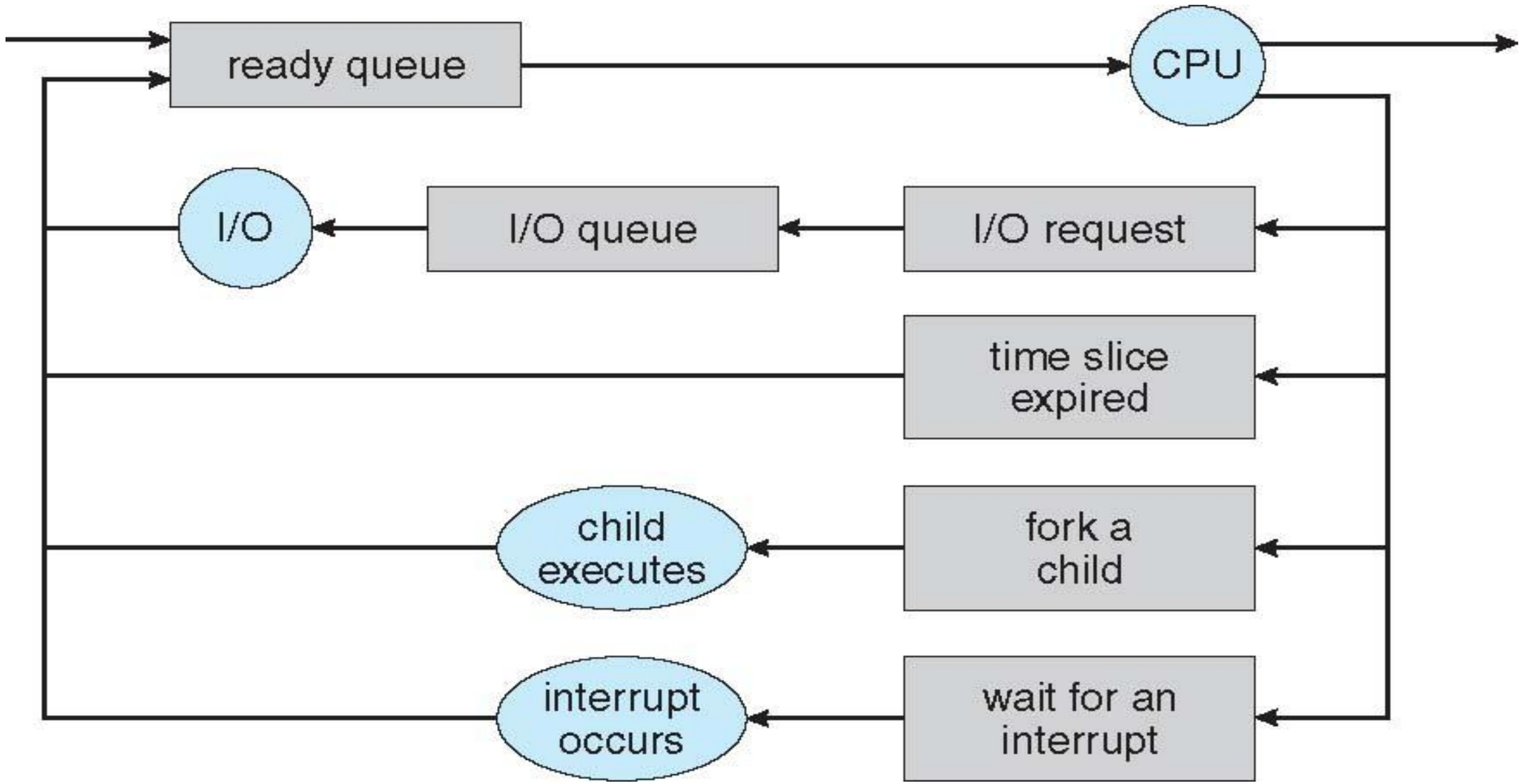
Proses Çizelgeleme

- İşlemcinin kullanımı maks. seviyeye çıkarmak için , zaman paylaşımıyla prosesler arasında çok hızlı geçişler gerçekleşir.
- **Proses çizelgeleyicisi,** işlenecek prosesi işleme hazır prosesler arasından seçer.
- Prosesler aşağıdaki **çizelgeleme kuyruklarında** tutulur.
  - **İş kuyruğu**– sistemdeki tüm prosesler kümesi
  - **Hazır kuyruğu**– ana bellekte bulunan, hazır ve işleme girmeyi bekleyen prosesler kümesi
  - **Aygıt kuyrukları**– I/O aygıtlarından gelecek mesajı bekleyen prosesler kümesi
  - Prosesler kuyruklar arasında geçiş yapabilir.

Hazır Kuyruğu ve Çeşitli I/O Aygıt Kuyrukları



Proses Çizelgeleme Diyagramı



Çizelgeleyiciler

- **Kısa vadeli çizelgeleyici / CPU çizelgeleyici** – Çalıştırılacak bir sonraki Prosesi seçer ve CPU tahsis eder.
  - Bazen sistemde sadece bir çizelgeleyici vardır
  - sık sık çağrılırlar. (milisaniyeler düzeyinde) ⇒ (hızlı olmalı)
- **Uzun vadeli çizelgeleyici / iş çizelgeleyici** – Hazır kuyruğuna getirilmesi gereken prosesleri seçer.
  - seyrek çağrılırlar (saniyeler, dakikalar düzeyinde) ⇒ (yavaş olabilir)
  - Uzun vadeli çizelgeleyiciler, *çoklu programlama* derecesini kontrol eder.

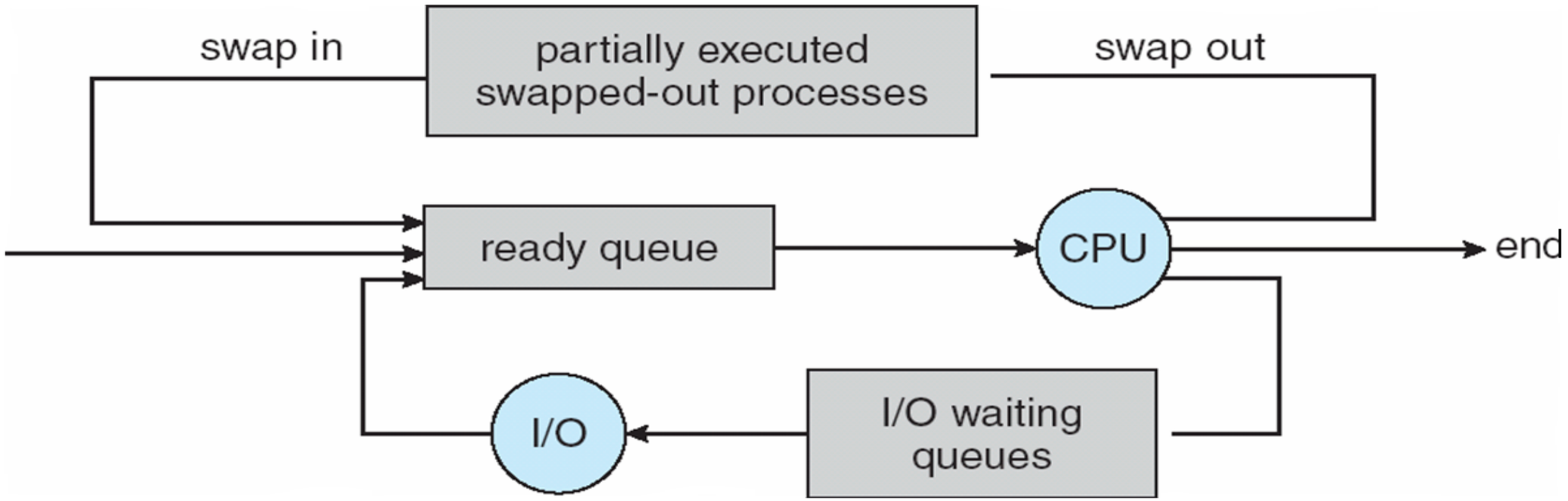
Çizelgeleyiciler

- Prosesler her iki biçimde de tanımlanmış olabilir:
  - **I/O-bağımlı Proses** –giriş/çıkış işlemlerinde daha fazla zaman harcar (hesaplamaya göre), çok az CPU sarfiyatı
  - **CPU-bağımlı Proses** – Hesaplamalar daha fazla zaman harcar, çok fazla CPU sarfiyatı

- Uzun vadeli çizelgeleyiciler iyi bir proses karışımı (I/O-bağımlı ve CPU-Bağımlılar arasından) oluşturmaya çalışırlar

- **Orta vadeli çizelgeleyiciler,** çoklu-programla derecesi düşürülmesi gerekiyorsa orta vadeli çizelgeleyiciler eklenebilir
  - Prosesin bellekten kaldırılması, diskte saklanması, yürütmeye devam etmek için diskten geri getirilmesi(swapping)

Orta vadeli çizelgeleyici eklenmesi



İçerik Değiştirme (Context Switch)

- CPU diğer prosese geçtiği zaman, sistem mutlaka eski prosesin durumunu kaydetmeli ve yeni prosesin daha önce kaydedilmiş durumunu yüklemeli

- Bir prosesin içeriği PCB'nin içinde gösterilir.
- Aşırı sayıda içerik değiştirme; sistem geçişler sırasında kullanımda olmaz
  - Daha karmaşık İşletim Sist. ve PCB -> daha uzun içerik değişimi

- Zaman ,Donanım desteğine bağlıdır.
  - Bazı donanımlar CPU başına birden fazla kaydedici kümesi sağlar -> birden fazla içerik bir kerede yüklenir

Proses İşlemleri

- Sistem proses işlemleri için bir mekanizması olması gerekir;
  - Proses oluşturma
  - Proses sonlandırma
  - Proses İletişimi
  - ..

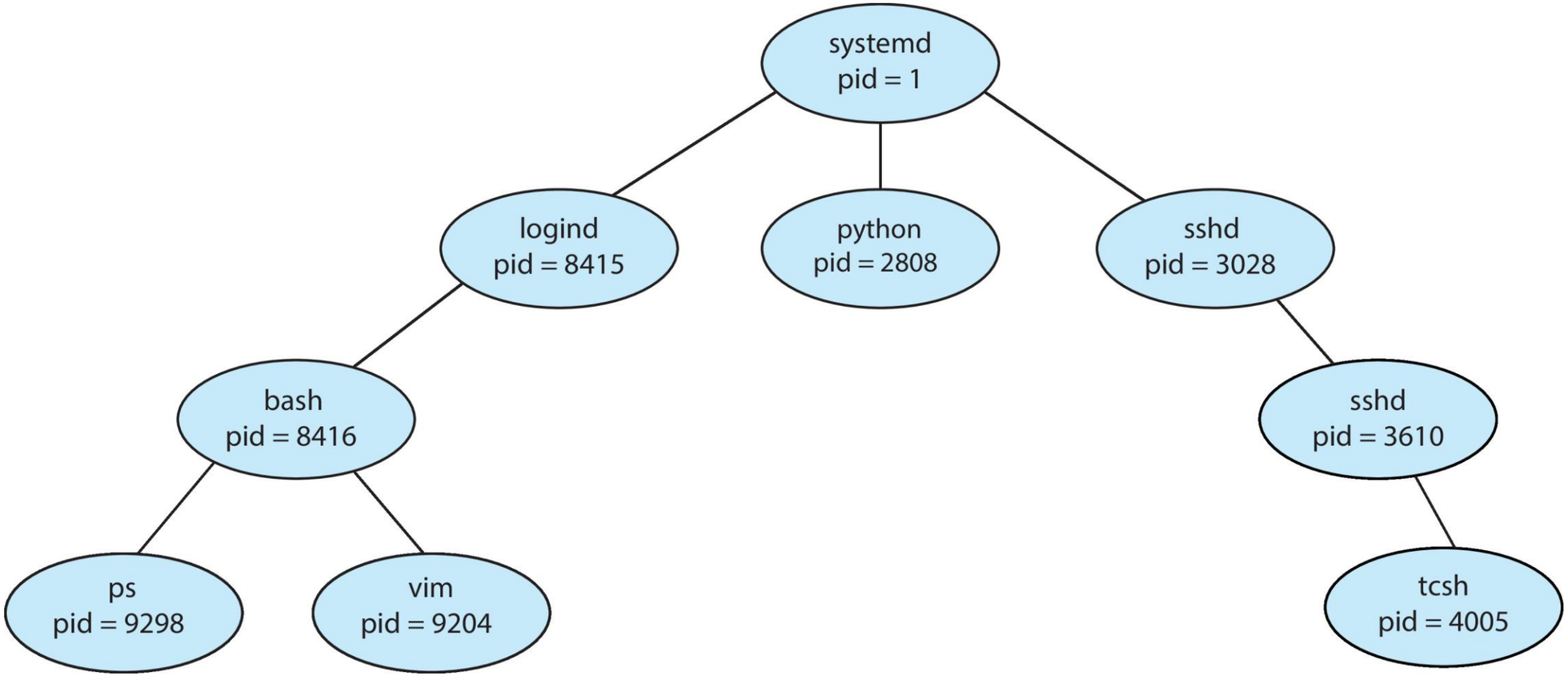
Proses Oluşturulması

- Ebeveyn Proses, çocuk prosesleri oluşturur. Bu şekilde ağaç yapısı meydana gelir.
- Genelde prosesler**, bir proses kimlik numarası** (Proses identifier - pid) ile tanımlanır ve yönetilir.

- Kaynak Paylaşımı seçenekleri:
  - Ebeveyn ve çocuk prosesler tüm kaynakları paylaşır.
  - Çocuk prosesler ebeveyn prosesin bir kısım kaynaklarını kullanır.
  - Ebeveyn ve çocuk hiçbir kaynağı paylaşmaz (Ayrı ayrı kaynakları vardır).

- Uygulama seçenekleri:
  - Ebeveyn ve çocuk proses eşzamanlı çalışır.
  - Ebeveyn proses, çocuk proses sonlanana kadar bekler.

A Tree of Processes in Linux



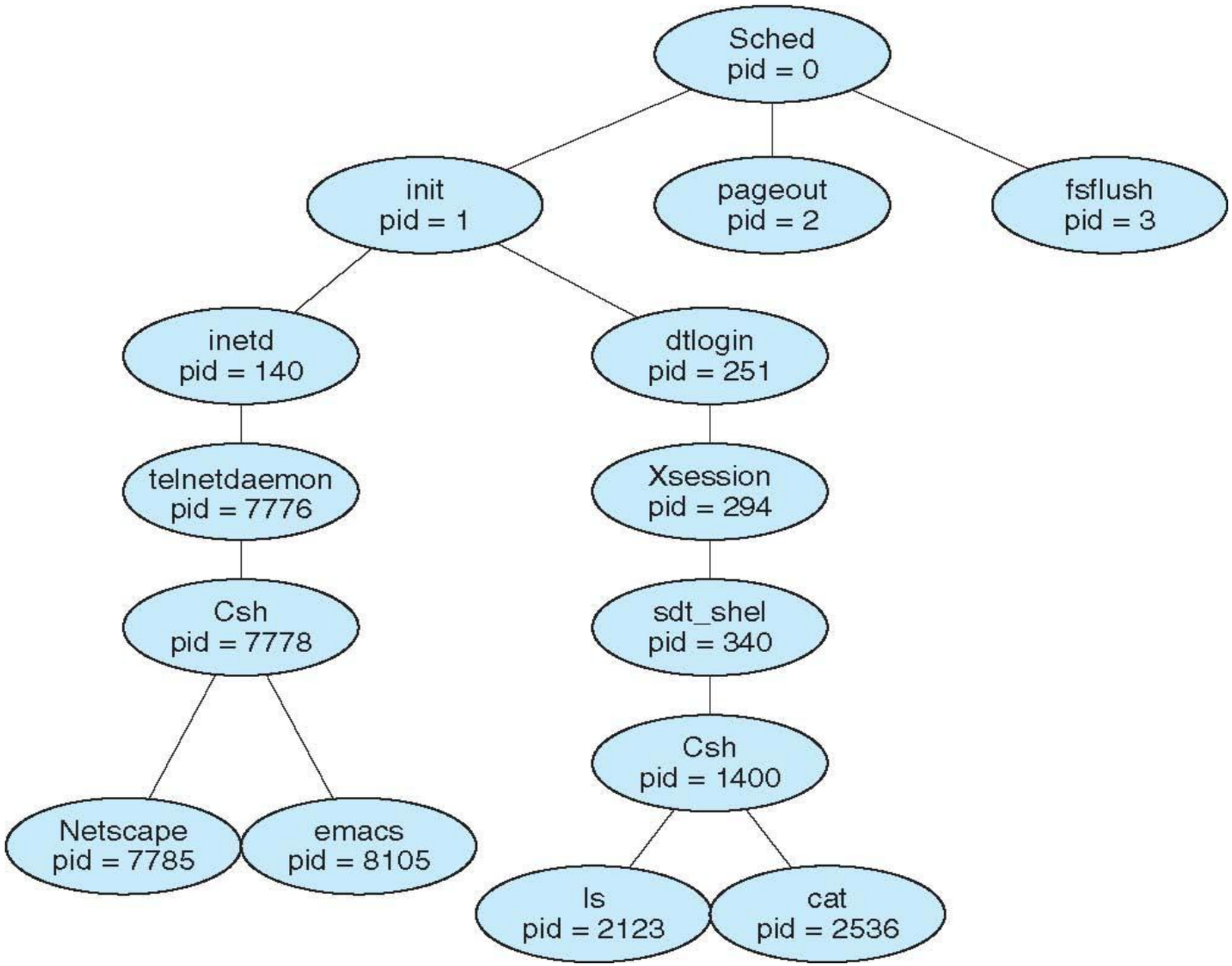
A Tree of Processes in Linux

```
abdullah@abdullah-VirtualBox: ~
Dosya  Düzenle  Görünüm  Ara  Uçbirim  Yardım
abdullah@abdullah-VirtualBox:~$ ps -ef
UID          PID  PPID  C STIME TTY          TIME CMD
root           1     0  0 10:14 ?        00:00:05 /sbin/init splash
root           2     0  0 10:14 ?        00:00:00 [kthreadd]
root           4     2  0 10:14 ?        00:00:00 [kworker/0:0H]
root           6     2  0 10:14 ?        00:00:00 [mm_percpu_wq]
root           7     2  0 10:14 ?        00:00:00 [ksoftirqd/0]
root           8     2  0 10:14 ?        00:00:00 [rcu_sched]
root           9     2  0 10:14 ?        00:00:00 [rcu_bh]
root          10     2  0 10:14 ?        00:00:00 [migration/0]
root          11     2  0 10:14 ?        00:00:00 [watchdog/0]
root          12     2  0 10:14 ?        00:00:00 [cpuhp/0]
root          13     2  0 10:14 ?        00:00:00 [cpuhp/1]
root          14     2  0 10:14 ?        00:00:00 [watchdog/1]
root          15     2  0 10:14 ?        00:00:00 [migration/1]
root          16     2  0 10:14 ?        00:00:00 [ksoftirqd/1]
root          18     2  0 10:14 ?        00:00:00 [kworker/1:0H]
root          19     2  0 10:14 ?        00:00:00 [kdevtmpfs]
root          20     2  0 10:14 ?        00:00:00 [netns]
```

A Tree of Processes in Linux

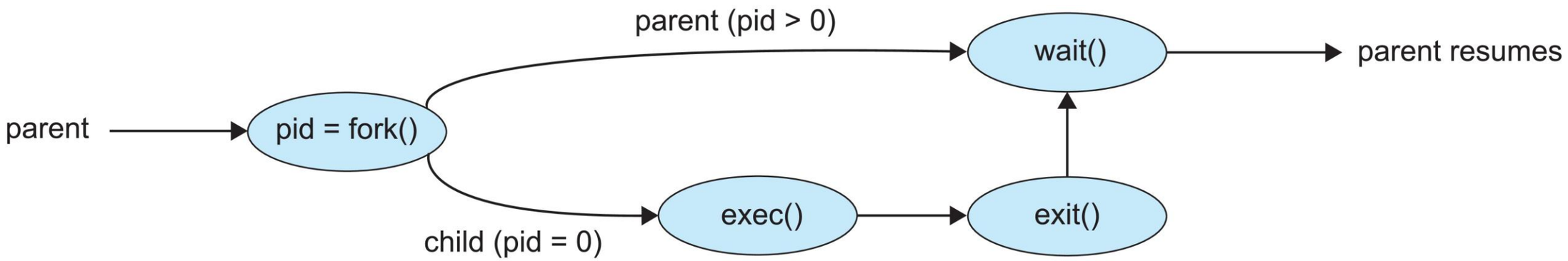
```
abdullah@abdullah-VirtualBox: ~
Dosya  Düzenle  Görünüm  Ara  Uçbirim  Yardım
abdullah@abdullah-VirtualBox:~$ pstree -ap 0
?,
  ├─(kthreadd,2)
  │     ├─(acpi_thermal_pm,85)
  │     ├─(ata_sff,33)
  │     ├─(charger_manager,121)
  │     ├─(cpuhp/0,12)
  │     ├─(cpuhp/1,13)
  │     ├─(crypto,30)
  │     ├─(devfreq_wq,36)
  │     ├─(ecryptfs-kthrea,42)
  │     ├─(edac-poller,35)
  │     ├─(ext4-rsv-conver,185)
  │     ├─(iprt-VBoxWQueue,288)
  │     ├─(ipv6_addrconf,95)
  │     ├─(jbd2/sda1-8,184)
  │     ├─(kauditd,22)
  │     ├─(kblockd,32)
  │     ├─(kcompactd0,27)
  │     ├─(kdevtmpfs,19)
  │     ├─(khugepaged,29)
  │     ├─(khungtaskd,24)
  │     ├─(kintegrityd,31)
  │     ├─(ksmd,28)
```

Solaris'te Proses Ağaç Yapısı



Proses Oluşturulması (Devam)

- Adres alanı
  - Çocuk proses, ebeveyn prosesin alanını kopyalar (Aynı prog., aynı veri).
  - Çocuk prosese ait bir program yüklenmiş olabilir.
- UNIX örnekleri :
  - **Fork()** sistem çağrısı, yeni bir proses oluşturur.
  - **Exec()** sistem çağrısı Prosesin bellek alanını yeni bir program ile yer değiştirmek için kullanılır. (fork() sistem çağrısından sonra)



- Başarılı bir şekilde tamamlandıktan sonra fork(), yavru prosese 0 döndürür ve yavru prosesin ID'sini ebeveyn prosese döndürür.

Fork İşlemi Yapan C Programı

```
#include <sys/types.h>
#include <studio.h>
#include <unistd.h>
int main()
{
pid_t  pid;
        /* fork another Proses */
        pid = fork();
        if (pid < 0) { /* error occurred */
              fprintf(stderr, "Fork Failed");
              return 1;
        }
        else if (pid == 0) { /* child Proses */
              execlp("/bin/ls", "ls", NULL);
        }
        else { /* parent Proses */
              /* parent will wait for the child */
              wait (NULL);
              printf ("Child Complete");
        }
        return 0;
}
```

```
int execlp (
 File,
 Argument0 [, Argument1
, ...], 0)
const char *File, *Argument0, *Argument
1, ...;
```

https://www.ibm.com/docs/en/aix/7.2?topic=e-exec-execl-execle-execlp-execv-execve-execvp-exect-fexecve-subroutine

Creating a Separate Process via Windows API

```
#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);
}
```

Process Creation in Java

```
import java.io.*;

public class OSProcess
{
 public static void main(String[] args) throws IOException {
  if (args.length != 1) {
   System.err.println("Usage: java OSProcess <command>");
   System.exit(0);
  }

  // args[0] is the command
  ProcessBuilder pb = new ProcessBuilder(args[0]);
  Process proc = pb.start();

  // obtain the input stream
  InputStream is = proc.getInputStream();
  InputStreamReader isr = new InputStreamReader(is);
  BufferedReader br = new BufferedReader(isr);

  // read what is returned by the command
  String line;
  while ( (line = br.readLine()) != null)
     System.out.println(line);

  br.close();
 }
}
```

Proses'lerin Sonlanması

- Proses son komutunu çalıştırdıktan sonra işletim sistemine silinip silinmeyeceğini sorar (exit() sistem çağrısı ile)
  - Durum verileri çocuktan ebeveyne geri döndürülür (**wait() ile**)
  - Prosese ayırılan alan işletim sistemi tarafından geri alınır.

- Ebeveyn Proses çocuk Prosesin çalışmasını sonlandırabilir (**abort() ile**). Bazı sebeplerden nedeniyle;
  - Çocuk proses tahsis edilmiş kaynakların dışına çıkmış ise
  - Artık çocuk prosese görev tayin etmek gerekmiyor ise.
  - Eğer ebeveyn proses sonlandırılırsa

Proses'lerin Sonlanması

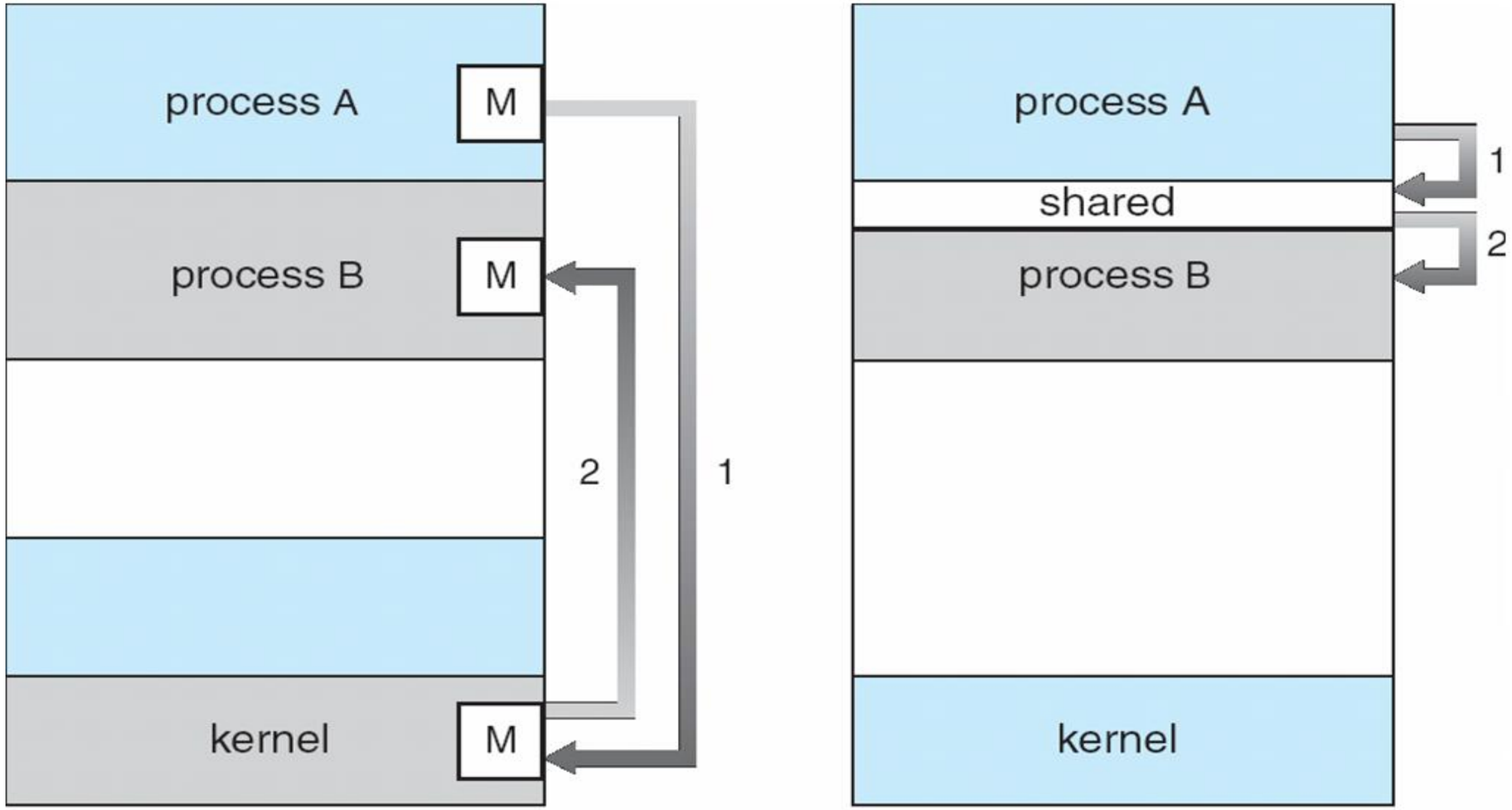
- Bazı işletim sistemi ebeveyn proses sonlandırıldıktan sonra çocuk prosesin çalışmasına izin vermez
  - Tüm çocuk Prosesler sonlandırılır.- **basamaklı sonlandırma**
  - Sonlandırma İşletim Sist. tarafından başlatılır

- Ebeveyn proses bir çocuk prosesin sonlanmasını bekleyebilir (wait() sistem çağrısı ile). Bu çağrıya durum bilgisi ile sonlandırılan prosesin pid'si geri döndürülür
  - `pid = wait(&status);`

Prosesler Arası İletişim

- Prosesler işletim sistemi içerisinde bağımsız ya da işbirliği içinde çalışabilirler.
- İşbirliği içerisindeki prosesler veri paylaşımı da dahil olmak üzere diğer prosesleri etkileyebilir ya da diğer proseslerden etkilenebilirler.
- Proseslerin işbirliği yapma nedenleri:
  - Bilgi paylaşımı
  - Daha hızlı hesaplama
  - Modülerlik
  - Konfor/Rahatlık- Birçok görevi aynı anda yapabilir
- İşbirliği içindeki prosesler prosesler arası haberleşmeye (Interproses communication - IPC) ihtiyaç duyarlar.
- 2 temel IPC modeli mevcuttur:
  - Paylaşılmış bellek
  - Mesajlaşma

Haberleşme Modelleri (Doğrudan)

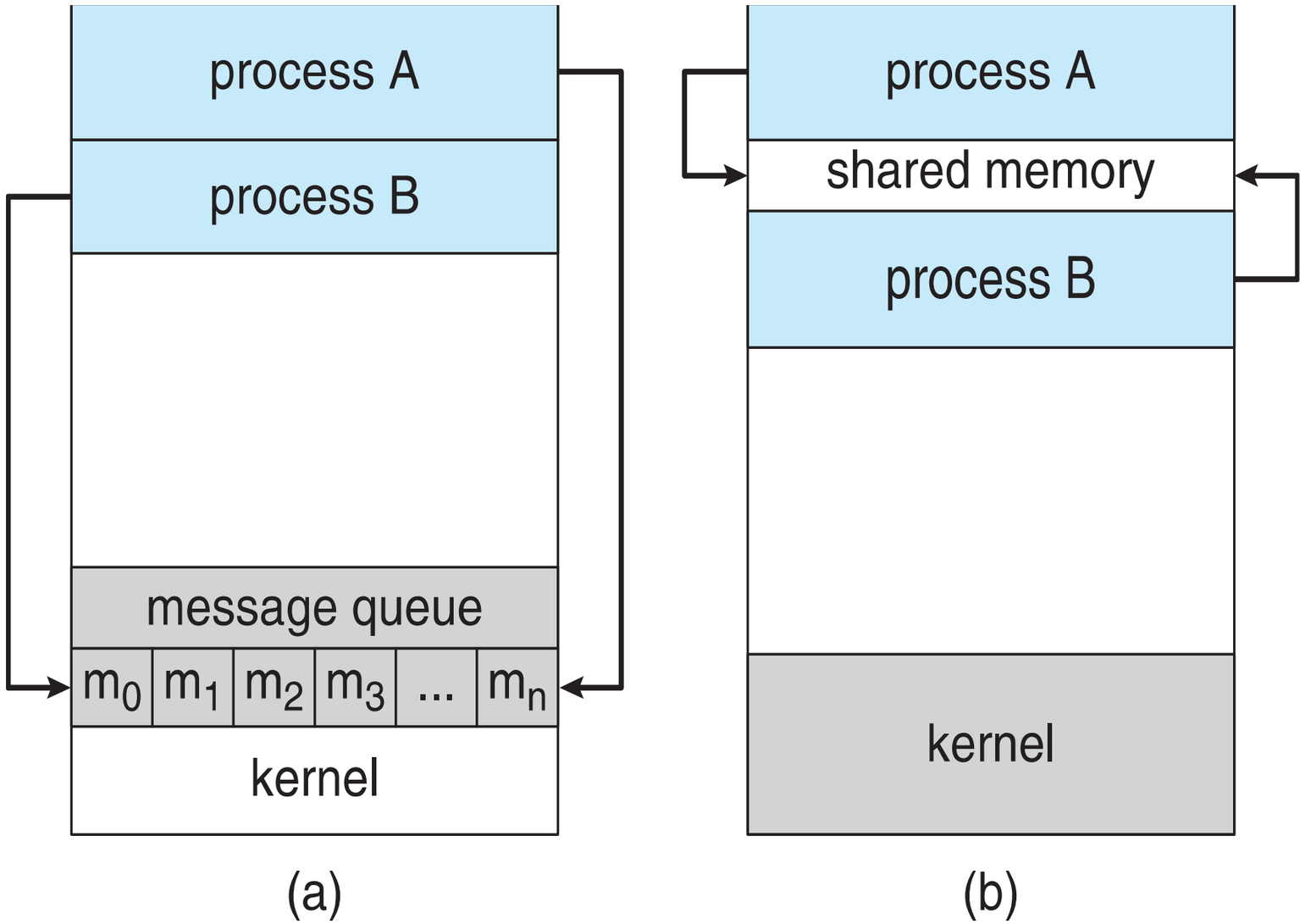


(a) (b)

(a) Mesajlaşma

(b) Paylaşılan Hafıza

Haberleşme Modelleri (Dolaylı)



(a) Message passing.

(b) shared memory.

İşbirliği içerisindeki Prosesler

- **Bağımsız** prosesler, diğer proseslerin çalışmasından etkilenemez ve diğer prosesleri etkileyemezler.
- **İşbirliği yapan** prosesler, diğer proseslerin çalışmasından etkilenebilir ve diğer prosesleri etkileyebilirler.

- Prosesler arası işbirliğinin avantajları :
  - Bilgi paylaşımı
  - Daha hızlı hesaplama
  - Modülerlik
  - Rahatlık

Üretici-Tüketici Problemi

- İşbirliği içindeki proseslere ilişkin bir paradigma: üretici proses tüketici proses tarafından kullanılmak üzere bilgi üretir. Bir derleyicini assembly kodu üretmesi ve assembler'ın bu kodu işlemesi. (İstemci-Sunucu mantığının temeli). Çözüm : paylaşılmış hafıza.
  - *Sınırlandırılmamış tampon*: tampon için limit konulmamıştır
  - *Sınırlandırılmış tampon*: sabit bir tampon boyutu mevcuttur.

Sınırlı-Tamponlu–Paylaşımlı-Bellek Çözümü

- Paylaşılmış veri

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

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

- Çözüm doğru, ancak sadece BUFFER_SIZE-1 eleman kullanılabilir.

Sınırlı-Tampon – Üretici

```
while (true) {
   /* data üretilir */

  while (((in = (in + 1) % BUFFER SIZE count)  == out)

     ;   /* çalışmaz – serbest tampon yok */

     buffer[in] = item;

     in = (in + 1) % BUFFER SIZE;

}
```

Sınırlı Tampon – Tüketici

```
while (true) {
    while (in == out)
          ; // çalışmaz -- nothing to consume

     // buffer tarafından veri silinir
     item = buffer[out];
     out = (out + 1) % BUFFER SIZE;
 return item;
}
```

Prosesler Arası İletişim - (Paylaşımlı Bellek)

- İletişim kurmak isteyen prosesler arasında paylaşılan bir bellek alanı vardır
- İletişimin kontrolü prosesler arasındadır (İşletim sisteminde değil)
- Asıl sorun, kullanıcı proseslerinin paylaşılan belleğe eriştiklerinde işlemlerini senkronize etmesine olanak tanıyacak mekanizma sağlamaktır.
- Senkronizasyon konusu önümüzdeki bölümlerde detaylı bir şekilde anlatılacaktır

Prosesler Arası İletişim - (Mesajlaşma)

- Proseslerin arasında iletişimin ve senkronizasyonun sağlanması için mekanizma.
- Mesaj sistemi – prosesler birbiri ile, paylaşılan değişkenleri kullanmadan iletişim kurar.
- IPC iki işlemi destekler :
  - **send**(*message*) – gönderilecek mesaj boyutu, sabit ya da değişken olabilir.
  - **receive**(*message*)
- Eğer *P* ve *Q* prosesleri iletişim kurmak istiyorsa, şu işlemleri yapmaları gerekir :
  - Aralarında iletişim bağlantısı var olmalıdır.
  - send /receive yardımı ile mesaj alışverişi gerçekleştirmelidirler.
- İletişim bağlantısı oluşturulması
  - fiziksel (ör., paylaşılmış bellek, donanım veri yolu, ağ)
  - mantıksal (ör., doğrudan veya dolaylı, senkron veya asenkron, otomatik veya belirgin tamponlama)

Doğrudan İletişim

- Proseslerin her biri gönderici ve alıcı olarak isimlendirilmelidir :
  - **send** (*P, message*) – P prosesine mesaj gönder
  - **receive**(*Q, message*) – Q prosesinden mesaj al

- İletişim bağlantısının özellikleri :
  - Bağlantılar otomatik olarak kurulur.
  - Her bir proses çifti arasında tam olarak bir bağlantı vardır.
  - Bir link 2 proses çifti ile ilişkilendirilebilir.
  - Bağlantı tek yönlü olabilir, ancak genellikle iki yönlüdür.

Dolaylı İletişim

- Mesajlar posta kutularından (portlar) alınır veya buralara gönderilir.
  - Her posta kutusu tek bir tanımlayıcıya (ID) sahiptir
  - Prosesler paylaşılmış bir posta kutusuna sahipse iletişim kurabilirler.

- İletişim bağlantısı özellikleri şunlardır :
  - Bağlantı, prosesler arası paylaşılmış bir posta kutusu var ise kurulur.
  - Bir bağlantı ikiden fazla proses ile ilişkilendirilebilir.
  - Her bir proses çifti birden fazla bağlantıya sahip olabilir.
  - Bağlantı tek yönlü ya da çift yönlü olabilir.

Doğrudan Olmayan İletişim

- İşlemler
  - Yeni bir posta kutusu oluştur,
  - Posta kutusu aracılığıyla mesaj gönder ve al.
  - posta kutusunu yok et.

- İletişim basitçe şu şekilde gerçekleşir:

  send(*A, message*) – A posta kutusuna bir mesaj gönder

  receive(*A, message*) – A posta kutusundan bir mesaj al.

Doğrudan Olmayan İletişim

- Posta kutusu paylaşımı
  - P_1, P_2, ve P_3 Prosesleri A posta kutusunu paylaşıyor.
  - P_1 mesaj gönderiyor; P_2 ve P_3 mesajı alıyor.
  - Mesajı hangisi almıştır?
- Çözüm:
  - Bir bağlantının en fazla iki proses ile ilişkilendirilmesine izin verir.
  - Bir seferde yalnızca bir proses alma işlemini yürütmesine izin verir.
  - Sistemin rastgele bir alıcı seçimine izin verir. Gönderici, alıcının kim olduğunu bildirir.

Senkronizasyon

- Mesaj iletimi engelli ya da engelsiz olabilir.
- **Engelli**, senkron iletim olarak düşünülebilir.
  - **Engelli** gönderim**,** mesaj alınana kadar gönderici engellenir.
  - **Engelli** alım**,** mesaj hazır olana kadar alıcı engellenir.
- **Engelsiz,** asenkron iletim olarak düşünülebilir.
  - **Engelsiz** gönderim, mesaj yollanır ve devam edilir.
  - **Engelsiz** alım, hazır mesaj varsa alır yoksa boş-null değer alır.
- Farklı kombinasyonlarda mümkün
  - Eğer alım ve gönderim bloke edilmiş ise, alıcı ve gönderici arasında ***randevü*** vardır

Senkronizasyon

- Üretici-tüketici prob. önemsiz hale gelir. Üretici sadece send() çağrısında bulunur ve mesaj alıcıya veya mesaj kutusuna iletilene kadar bekler. Aynı şekilde tüketicide receive() çağrısında bulunur ve mesaj hazır oluncaya kadar bekler.

```
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 */
}
```

Tamponlama

- Haberleşme mesajlaşma ile gerçekleştirildiğinde mesajlar geçici bir kuyrukta tutulur ve bu mesaj kuyruğu, şu 3 yolla düzenlenir:
  1. Sıfır kapasite – Kuyrukta 0 mesaj
     Gönderici, alıcıya ulaşana kadar beklemelidir (Randevülü).
  2. Sınırlı kapasite –n adet mesaj kapasiteli, sınırlı uzunluk
     Gönderici, bağlantı dolu ise beklemelidir.
  3. Sınırsız kapasite – sonsuz uzunluk
     Gönderici hiçbir zaman beklemez.

IPC Sistem Örnekleri - POSIX

- **The Mode Bits for Access Permission**
- (octal 0666) when creating a new file, the permissions on the resulting file will be: S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH
- S_IRUSR Read permission owner of file, S_IWUSR Write permission owner of file, S_IRGRP Read permission bit for the group owner of the file, S_IROTH Read permission bit for other users.
- O_CREAT | O_RDWR oluştur, oku-yaz

IPC Sistem Örnekleri - POSIX

- /void *mmap(void *addr, size_t length, int prot, int flags, int fd, off_t offset);
- Eğer adres NULL ise, kernel haritalamanın oluşturulacağı (sayfa hizalı) adresi seçer;

```
void *addr = mmap(NULL,  object_size, PROT_READ |
PROT_WRITE, //Okuma ve yazma hakkı
MAP_SHARED, //IPC yapmak istiyoruz
shmem_fd, 0 );
```

- PROT_EXEC Pages may be executed.
- PROT_READ Pages may be read.
- PROT_WRITE Pages may be written.

- mmap işletim sisteminin sayfaları üzerinde çalıştığı için istenilen hafıza miktarı otomatik olarak sayfa büyüklüğüne göre ayarlanır.

IPC Sistem Örnekleri - POSIX

- POSIX(Portable Operating System Interface for Unix) Paylaşılmış bellek
- POSIX standardını kullanan programlar bir sistemden diğerine kolayca taşınabilir. Bir işletim sistemi hizmeti kümesi tanımlayan (IEEE) standardı.
  - Proses öncelikle paylaşılmış bellek alanı oluşturur.

```
segment id = shmget(IPC PRIVATE, size, S IRUSR | S
   IWUSR);
```

  - Proses paylaşılmış belleğe erişmek istemektedir.

```
shared memory = (char *) shmat(id, NULL, 0);
```

  - Nesnenin boyutu ayarlanmaktadır.

```
ftruncate(shm fd, 4096);
```

  - Şimdi, proses paylaşılan belleğe yazabilir.

```
sprintf(shared memory, "Writing to shared memory");
```

  - İşlem tamamlandığında önceden ayırılan bellek alanı geri alınabilir.

```
shmdt(shared memory);
```

IPC POSIX Producer

```
#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 Consumer

```
#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 Örnekleri - Mach

- Mach iletişimi mesaj tabanlıdır.
  - Hatta sistem çağrıları dahi birer mesajdır.
  - Her görev, oluşturma sırasında iki posta kutusu alır– Kernel ve Notify. Çekirdek görev ile haberleşirken kernel mesaj kutusunu kullanır ve olayın gerçeleşme durumunu notify kutusuna bilgilendirir.
  - Mesaj transferi için Sadece 3 sistem çağrısına ihtiyaç duyulur.

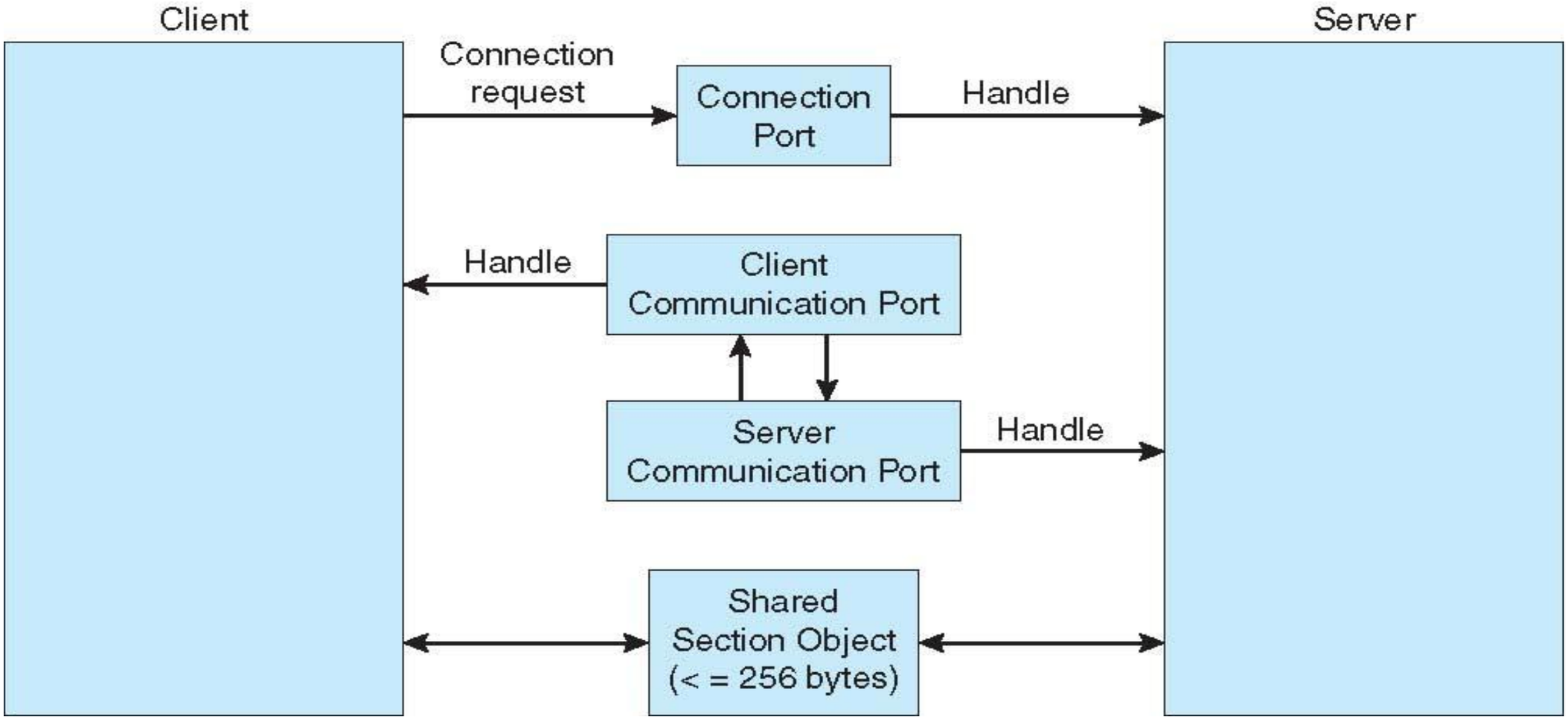
`msg_send(), msg_receive(), msg_rpc()[RemoteProcedureCall için]`

  - İletişimde mesaj kutularına ihtiyaç duyulur ve mesaj kutuları `port_allocate()` aracılığıyla oluşturulur. Portlar üzerinden haberleşirler.
  - Alım-gönderim esnektir, eğer posta-kutusu dolu ise 4 seçenek vardır;
    - Belirsiz süre bekle, mesaj kutuları boşalana kadar
    - N milisaniye bekle
    - Hemen geri dön
    - Geçici olarak mesajı önbelleğe al

IPC Sistem Örnekleri – Windows XP

- Mesaj iletimi yerel prosedür çağrı **(local procedure call - LPC)** birimi aracılıyla yönetilir
  - Yalnızca Aynı sistemdeki prosesler arasında çalışır
  - İletişim kanalları kurmak ve sürdürmek için portlarını (posta kutuları gibi) kullanır.
  - Haberleşme aşağıdaki gibi çalışır:
    - İstemci altsistemi bir bağlantı portu nesnesi oluşturur.
    - İstemci bağlantı isteği gönderir.
    - Sunucu iki özel iletişim portu oluşturur ve bunlardan birini istemciye gönderir.
    - İstemci ve sunucu mesajları göndermek, almak ve cevapları dinlemek amacıyla karşılıklı portlar kullanır.

Windows XP'de Yerel Prosedür Çağrıları



İstemci – Sunucu Sistemlerinde İletişim

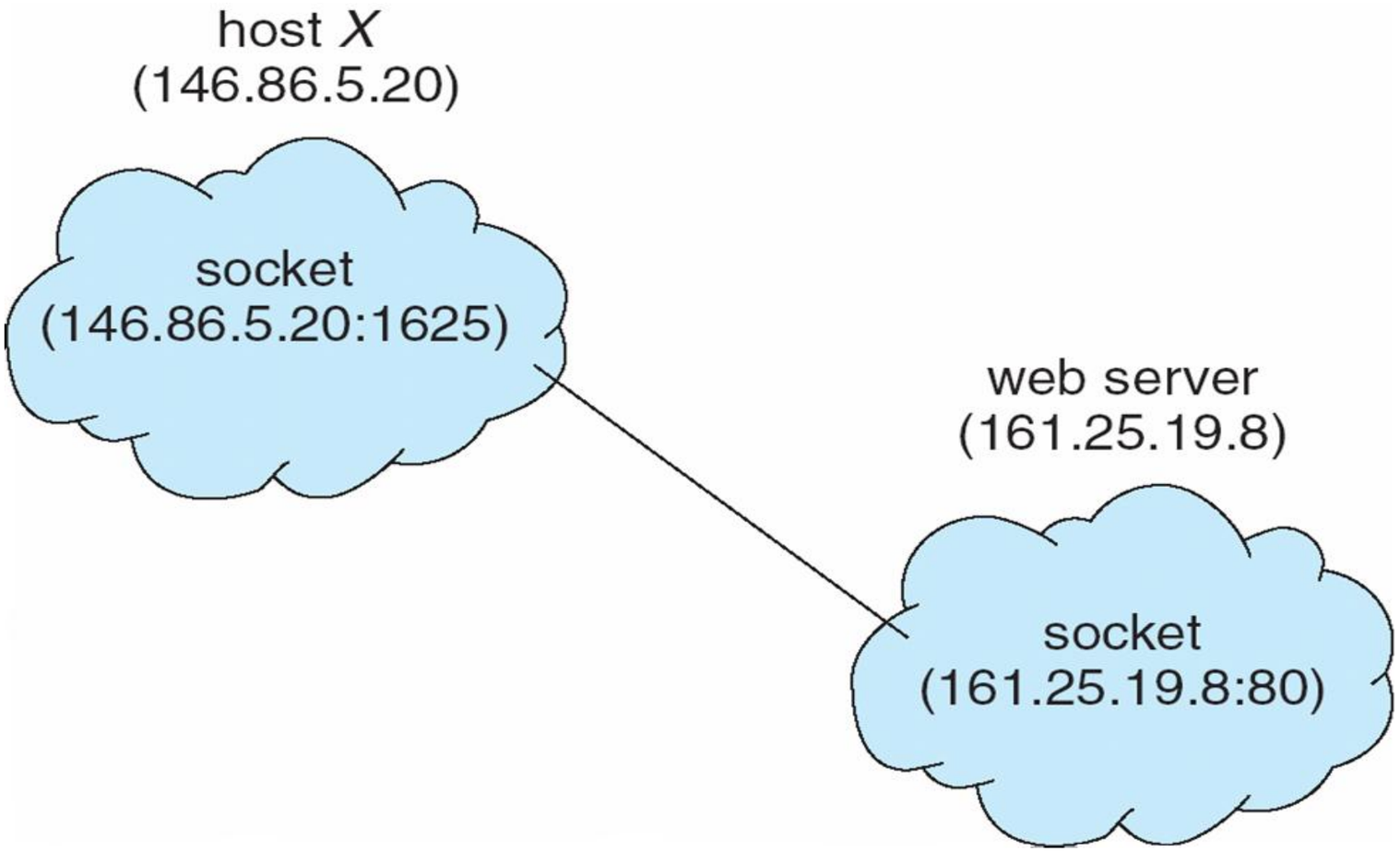
- Soketler
- Uzak Prosedür Çağrıları
- Tüneller- pipes
- Uzak Metot İsteği (RMI - Java)

Soketler

- Soket, bir iletişimin bitiş noktası olarak tanımlanabilir.
- İletişim, bir çift soket arasında meydana gelir.
- IP adresinin ve portun birleşimidir.

- **161.25.19.8:1625** soketi, **1625** portu ve **161.25.19.8** host (ana makinesi) demektir.
- 1024'ün altındaki portlar standart servisler için kullanılır (telnet sunucusu 23. portu dinler; FTP sunucu 21. portu dinler; web veya HTTP, sunucu 80. portu dinler).
- Özel IP adresi 127.0.0.1, prosesin üzerinde çalıştığı sistemi ifade eder.

Soket İletişimi



- Eğer yeni bir proses (host X'deki) yeni bir bağlantı oluşturmak isterse 1024 değerinin üstünde ve 1625 olmayan bir değerde port oluşturur ve yeni bir bağlantı oluşturur

Sockets in Java

- 3 çeşit soket vardır;
  - **Connection-oriented** (**TCP**)
    - Bağlantıya dayalı
  - **Connectionless** (**UDP**)
    - Bağlantısız
    - **`MulticastSocket`** sınıfı– veriyi çoklu alıcıya gönderilebilir

- Bu "Tarih" sunucusuna bakalım

```
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);
      }
   }
}
```

Uzak Prosedür Çağrısı

- Uzak prosedür çağrısı (Remote procedure call - RPC), Ağ ile birbirine bağlı sistemler üzerindeki prosesler arasında

- **Stub ()** – sunucu üzerindeki gerçek prosedür için istemci tarafındaki vekili. **Stub**, uzak nesneyi(remote object) temsil eden **Remote** referansları olarak adlandırılır.

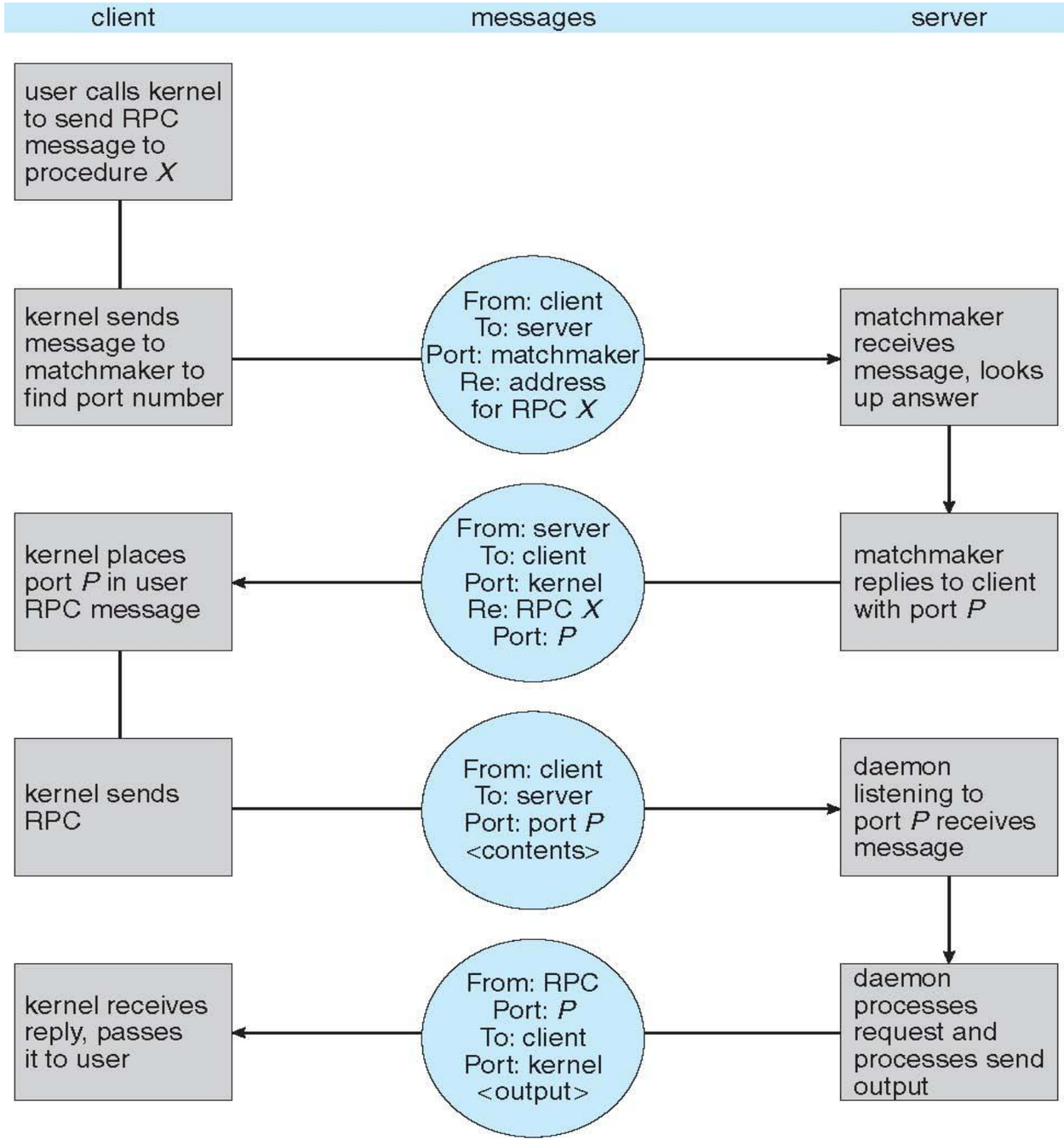
- İstemci tarafındaki stub, sunucunun yerini belirler ve parametreleri yönlendirir.

- Sunucu tarafındaki stub, mesajı alır, yönlendirilmiş parametreleri açar ve prosedürü sunucu üzerinde uygular.

Stub

- Bir (Stub) kalıntı/saplama, daha uzun bir programın yerini alacak, daha sonra yüklenecek veya uzaktan yerleştirilmiş küçük bir program yordamıdır.
- Örneğin, Uzaktan Prosedür Çağrıları (RPC) kullanan bir program, istenen prosedürü sağlayan programın yerine geçen taslaklar ile derlenir.
- (Stub) kalıntı/Saplama isteği kabul eder ve ardından (başka bir program aracılığıyla) uzak işleme yönlendirir.
- Bu prosedür hizmetini tamamladığında, sonuçları veya diğer durumu, talebi yapan programa geri döndüren (Stub) kalıntı/saplamaya geri döndürür.

RPC'nin Çalışma Prensibi



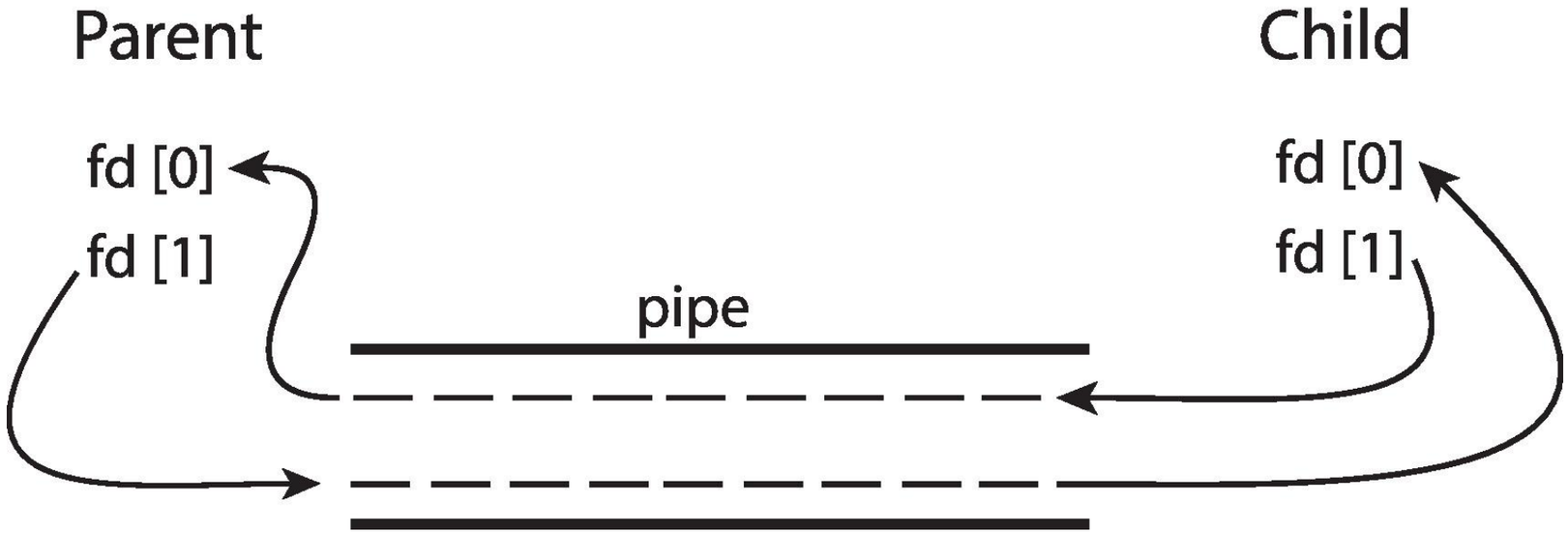
Boru/Tünel- Pipe

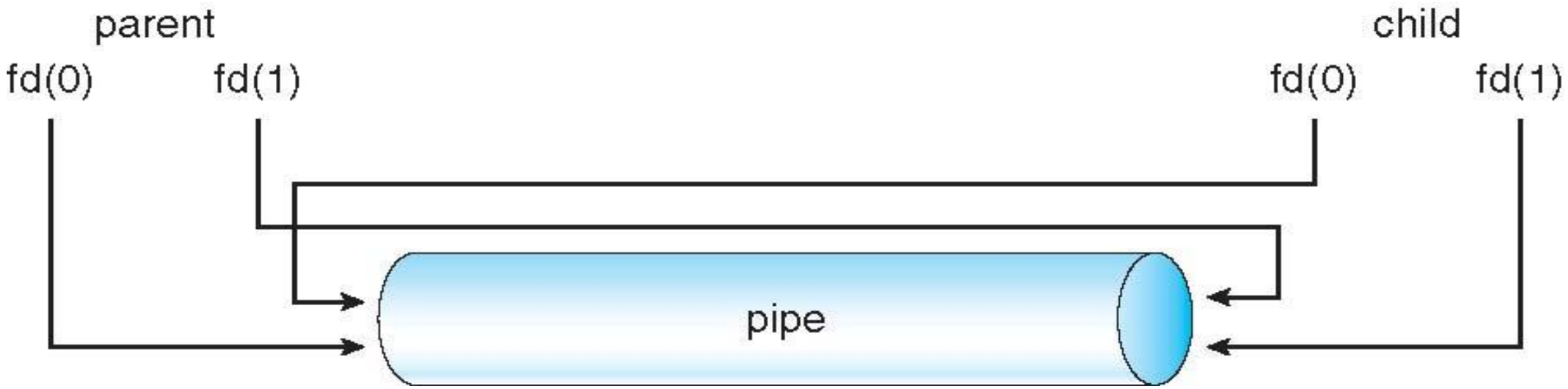
- İki proses arasında iletişime izin veren yapıdır. En temel haberleşme mekanizmalarındandır.
- **Sorunlar: (Uygularken dikkate alınması gerekenler)**
  - İletişim tek yönü mü, çift yönlü müdür?
  - İletişim iki yönlü ise yarı dubleks mi çalışır, yoksa tam dubleks mi çalışır?
  - İletişim halindeki prosesler arasında bir ilişki (ebeveyn-çocuk) olmalı mıdır?
  - Tüneller ağ üzerinden kullanılabilir mi? Yoksa aynı makine üzerinde mi

Sıradan Tüneller

- **Sıradan tüneller,** standart üretici-tüketici tipi iletişime izin verir. Dışarıdan erişime izin vermez.
- Üretici bir uçtan yazar (tünelin yazma ucu)
- Tüketici diğer ucundan okur (tünelin okuma ucu)
- Sıradan tüneller bu nedenle tek yönlü iletişim sağlar.
- Haberleşen prosesler arasında ebeveyn-çocuk ilişkisi gerekir.
- Karşılıklı iletişim gerekiyorsa 2 adet tünel gerekir

Sıradan Tüneller





Derleme/Çalıştırma

```
abdullah@abdullah-VirtualBox: ~/İndirilenler/24.10.18/24.10.18
Dosya  Düzenle  Görünüm  Ara  Uçbirim  Yardım
abdullah@abdullah-VirtualBox:~$ cd İndirilenler
abdullah@abdullah-VirtualBox:~/İndirilenler$ cd 24.10.18
abdullah@abdullah-VirtualBox:~/İndirilenler/24.10.18$ ls
24.10.18
abdullah@abdullah-VirtualBox:~/İndirilenler/24.10.18$ cd 24.10.18
abdullah@abdullah-VirtualBox:~/İndirilenler/24.10.18/24.10.18$ ls
2.ogr  p1.c
abdullah@abdullah-VirtualBox:~/İndirilenler/24.10.18/24.10.18$ gcc p1.c -lpthread -o p1
abdullah@abdullah-VirtualBox:~/İndirilenler/24.10.18/24.10.18$ ./p1
-----Ebeveyn yazdı-----
-----Yavru okudu-----
okunan = Merhaba
abdullah@abdullah-VirtualBox:~/İndirilenler/24.10.18/24.10.18$
```

Pd[0] = Alım ucu
Pd[1] = Gönderim ucu

```
Aç   *p1.c  ~/İndirilenler/24.10.18/24.10.18   Kaydet
// Pipe örneği
#include<stdio.h>
#include<stdlib.h>
#include<fcntl.h>
#include<unistd.h>

int main()
{
  int pid, ret;
  int pd[2];

  ret = pipe(pd);
  if(ret < 0){
    printf("Pipe error\n");
    exit(1);
  }


  pid = fork();
  if(pid > 0){ // Ebeveyn - yazan
    close(pd[0]);
    write(pd[1], "Merhaba", 8);
    printf("-----Ebeveyn yazdı-----\n");
  } else if(pid == 0){ // Yavru okuyan

    char buf[10];
    close(pd[1]);
    read(pd[0], buf, 8);
    printf("-----Yavru okudu-----\n");
    printf("okunan = %s\n", buf);
  }

C  Etiket Genişliği: 8  Sat 1, Süt 15  ARY
```

Sıradan Tüneller(Windows)

parent proses

```
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
#define BUFFER SIZE 25
int main(VOID)
{
HANDLE ReadHandle, WriteHandle;
STARTUPINFO si;
PROCESS INFORMATION pi;
char message[BUFFER SIZE] = "Greetings";
DWORD written;
/* set up security attributes allowing pipes to be inherited */
SECURITY ATTRIBUTES sa = {sizeof(SECURITY
ATTRIBUTES),NULL,TRUE};
/* allocate memory */
ZeroMemory(&pi, sizeof(pi));
/* create the pipe */
if (!CreatePipe(&ReadHandle, &WriteHandle, &sa, 0)) {
fprintf(stderr, "Create Pipe Failed");
return 1;
}
/* establish the START INFO structure for the child process */
GetStartupInfo(&si);
si.hStdOutput = GetStdHandle(STD OUTPUT HANDLE);
/* redirect standard input to the read end of the pipe */
si.hStdInput = ReadHandle;
si.dwFlags = STARTF USESTDHANDLES;
/* don't allow the child to inherit the write end of pipe */
SetHandleInformation(WriteHandle, HANDLE FLAG INHERIT, 0);
/* create the child process */
CreateProcess(NULL, "child.exe", NULL, NULL,
TRUE, /* inherit handles */
0, NULL, NULL, &si, &pi);
/* close the unused end of the pipe */
CloseHandle(ReadHandle);
/* the parent writes to the pipe */
if (!WriteFile(WriteHandle, message,BUFFER
SIZE,&written,NULL))
fprintf(stderr, "Error writing to pipe.");
/* close the write end of the pipe */
CloseHandle(WriteHandle);
/* wait for the child to exit */
WaitForSingleObject(pi.hProcess, INFINITE);
CloseHandle(pi.hProcess);
CloseHandle(pi.hThread);
return 0;
}
```

Sıradan Tüneller(Windows)

Çocuk proses

```
#include <stdio.h>
#include <windows.h>
#define BUFFER SIZE 25
int main(VOID)
{
HANDLE Readhandle;
CHAR buffer[BUFFER SIZE];
DWORD read;
/* get the read handle of the pipe */
ReadHandle = GetStdHandle(STD INPUT HANDLE);
/* the child reads from the pipe */
if (ReadFile(ReadHandle, buffer, BUFFER SIZE, &read, NULL))
printf("child read %s",buffer);
else
fprintf(stderr, "Error reading from pipe");
return 0;
}
```

Adlandırılmış Tüneller

- Adlandırılmış tüneller, sıradan olanlardan daha güçlüdür.
- İletişim çift yönlüdür.
- Haberleşen prosesler arasında ebeveyn-çocuk ilişkisi gerekli değildir.
- Birden fazla proses, kullanabilir.
- UNIX ve Windows işletim sistemlerince desteklenir.

Bölüm 3 - Son