

İŞLETİM SİSTEMLERİ

İşletim Sistemi Prosesler

ubuntu

Windows 11

macOS

macOS
Sonoma

Prof.Dr.Ahmet Zengin

Öğrenme Hedefleri

Bu konuyu çalıştıktan sonra:

- 📖 Proses Kavramı
- 📖 Proses Sıralama (Scheduling)
- 📖 Proses Üzerinde İşlemler
- 📖 Prosesler Arası İletişim
- 📖 Mesaj İletimi ve Paylaşımlı Bellek Sistemlerinde IPC
- 📖 IPC (Prosesler Arası İletişim) Örnekleri
- 📖 İstemci-Sunucu Sistemleri Arasındaki İletişim

Hedefler

- Bir prosesin ayrı bileşenlerini tanımlamak ve bunların bir işletim sisteminde nasıl temsil edildiğini ve sıralandığını görmek
- Uygun sistem çağrıları kullanarak programlar geliştirmek de dahil olmak üzere, bir işletim sisteminde proseslerin nasıl oluşturulduğunu ve sonlandırıldığını tanımlamak
- Paylaşılan bellek ve mesaj iletimini kullanarak prosesler arası iletişimi tanımlamak ve karşılaştırmak
- Prosesler arası iletişimi gerçekleştirmek için veri kanalları (pipes) ve POSIX paylaşılan belleği kullanan programlar tasarlamak
- Soketler ve uzak yordam çağrıları kullanarak istemci-sunucu iletişimini tanımlamak
- Linux işletim sistemiyle etkileşime giren çekirdek modülleri tasarlamak

Proses Kavramı

Bir işletim sistemi programları proses halinde yürütür:

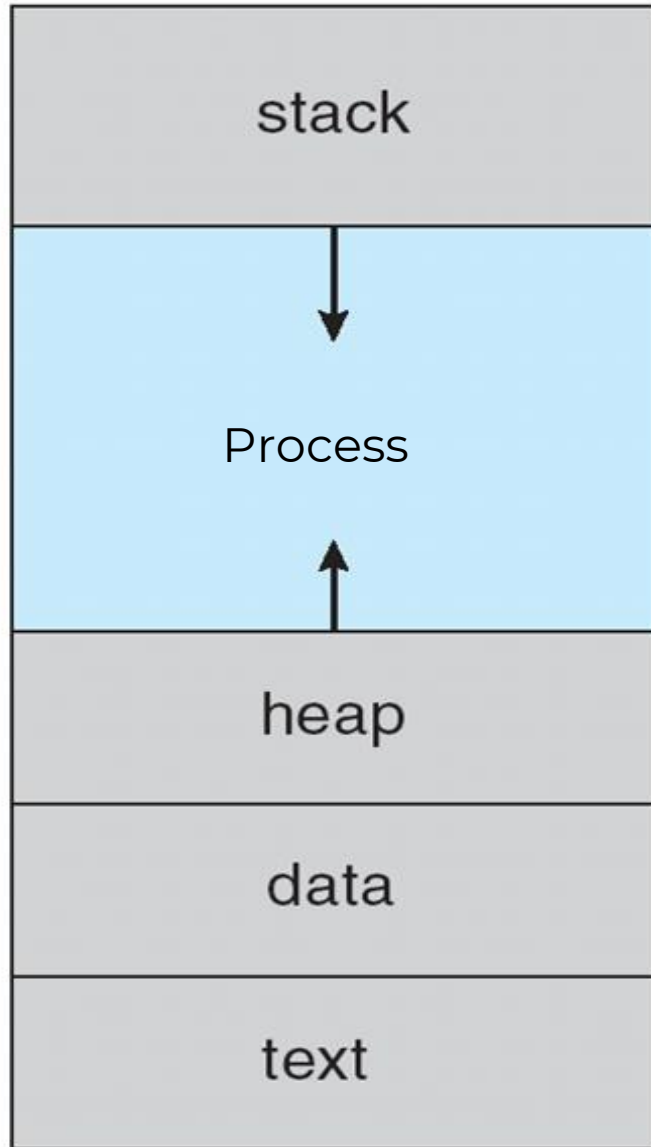
- Ders kitaplarında **görev / işlem / süreç / iş (job) / proses** terimleri birbirlerinin yerine kullanılır.
- **Proses**, yürütülen bir programdır ve prosesler sıralı bir biçimde yürütülmelidir.
- Bir proses aşağıdakileri alanları içerir:
 - Program kodu, metin alanı (**text section**)
 - Mevcut aktiviteyi içeren **program sayacı**, proses kaydedicileri
 - Geçici veriyi tutan yığın (**stack**)
 - Fonksiyon parametreleri, döndürülen adresler, yerel değişkenler
 - Veri bölümü (**data section**) global değişkenleri tutar
 - Çalışma anında dinamik olarak tahsis edilen bellek kısmı yığıt (**heap**)

Proses

- Program çalıştırılabilir bir dosya halinde harddiskte tutulan **pasif** bir varlıktır, proses ise **aktiftir**
 - Program çalıştırılabilir halde 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 birden fazla kullanıcının aynı programı çalıştırması.
 - Derleyici
 - Metin editörü

Bellekteki Bir Proses

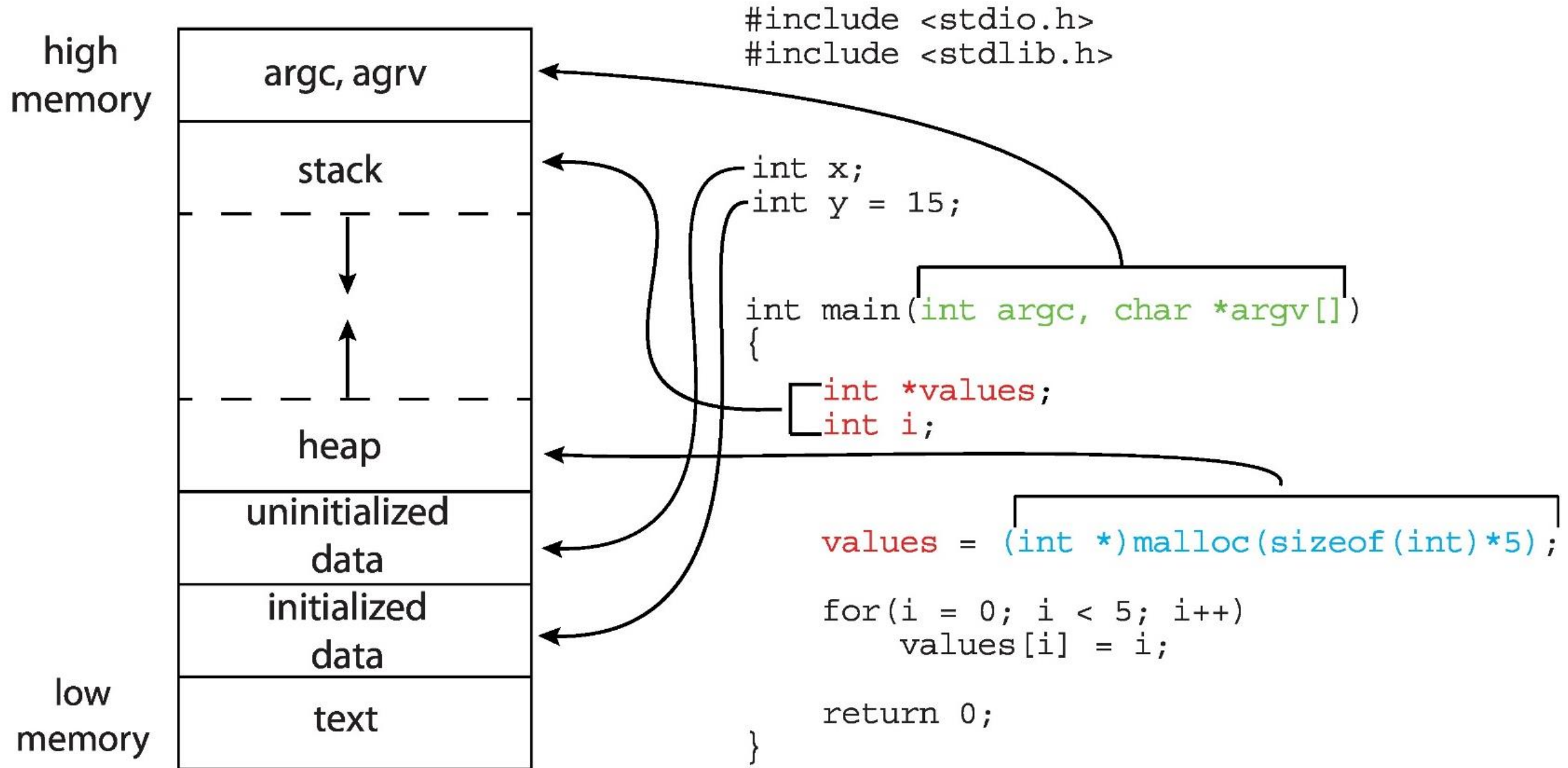
max



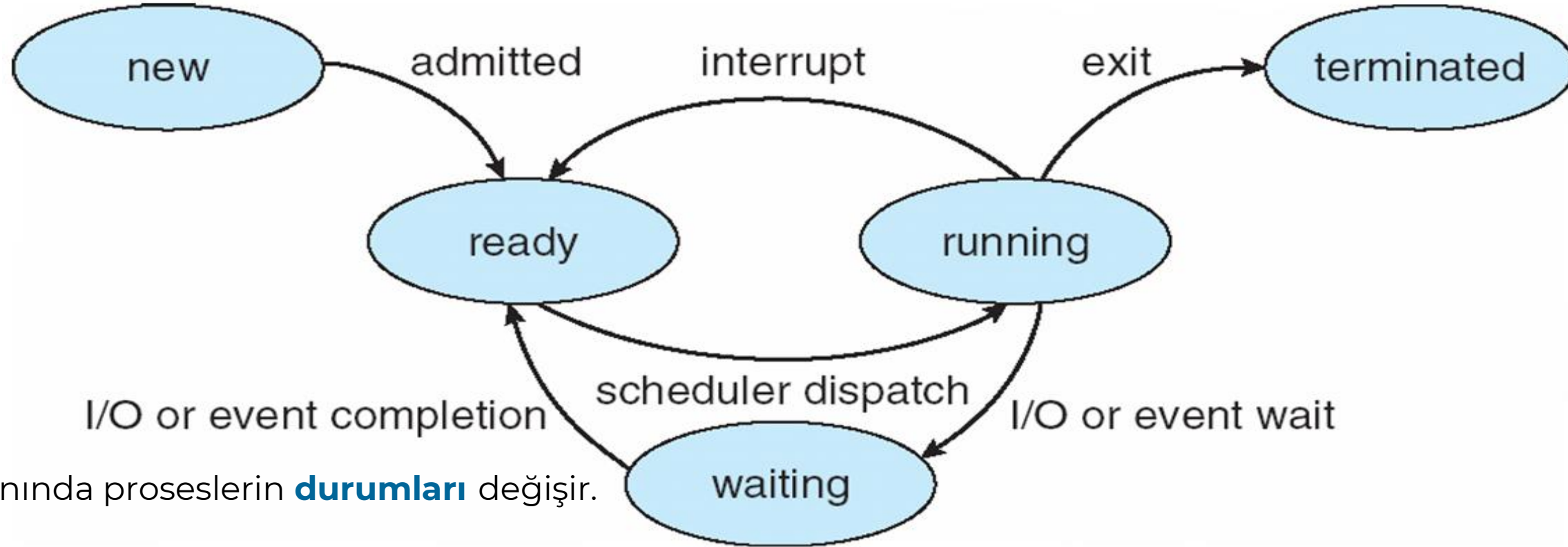
0

İşletim sistemi yeni oluşturulan bir prosesi, bellekte bulunan ilgili kısma yerleştirir.

Bir C Programının Bellek Yerleşimi



Proses Durumları



Çalışma anında proseslerin **durumları** değişir.

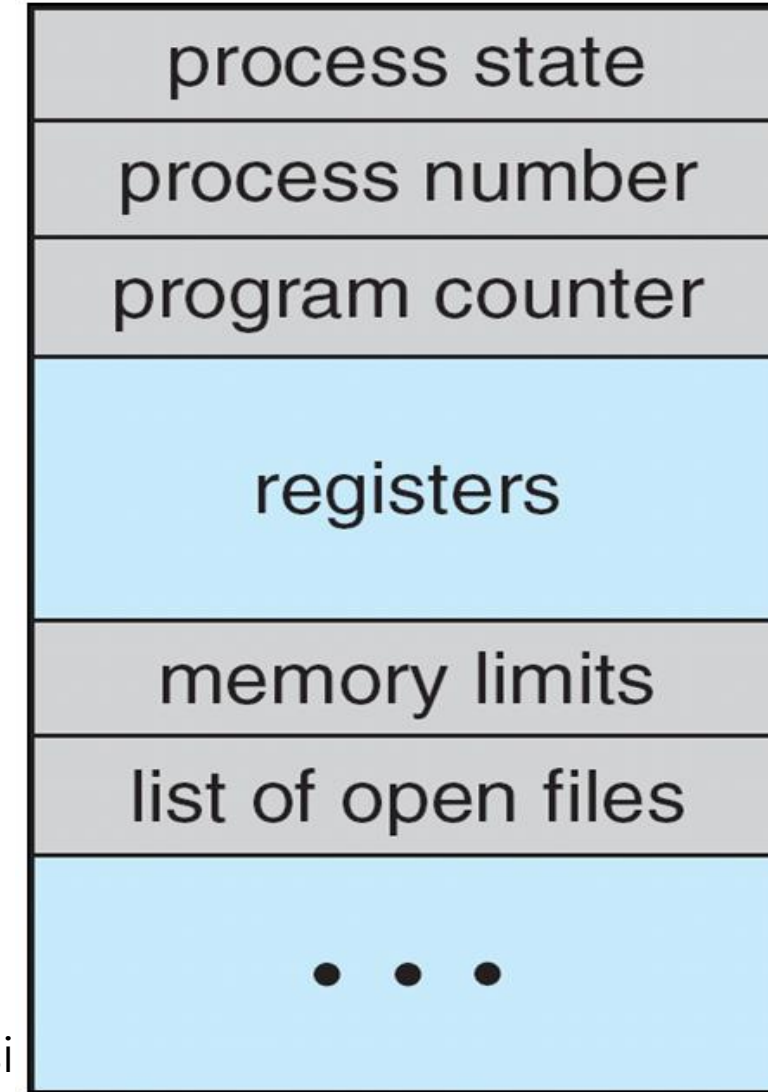
- **Yeni:** Yeni bir proses oluşturuluyor.
- **Çalışıyor:** İşlemler gerçekleştiriliyor.
- **Beklemede:** Proses bazı olayların gerçekleşmesini beklemektedir.
- **Hazır:** Proses, işlemciye aktarılmayı beklemektedir.
- **Sonlandırılmış:** Prosesin yürütülmesi tamamlandığını belirtir.

Proses Kontrol Bloğu (PCB)

Herbir proses ile ilişkili bilgi (**görev kontrol bloğu** olarak ta adlandırılır)

Şu bilgileri içerir :

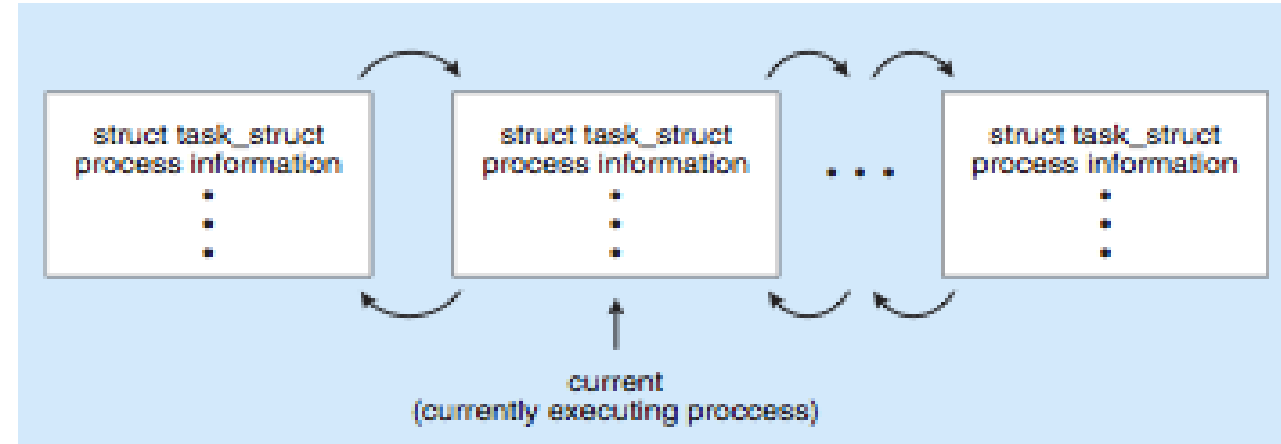
- **Proses durumu** – çalışıyor, beklemede
- **Program Sayacı** – bir sonraki çalışacak talimatın konumu
- **CPU kaydedicileri** – prosese ait tüm kaydedicilerin içerikleri
- **CPU sıralama bilgisi** – öncelikler, sıralama kuyruk pointerları
- **Bellek yönetim bilgisi** – prosese tahsis edilen bellek
- **Hesap bilgisi** – kullanılan CPU, başlangıçtan beri geçen zaman
- **I/O durum bilgisi** – prosese tahsis edilen I/O aygıtları, açık dosya listesi



Proseslerin Linux'te Temsili

- task_struct C yapısı içeriği:

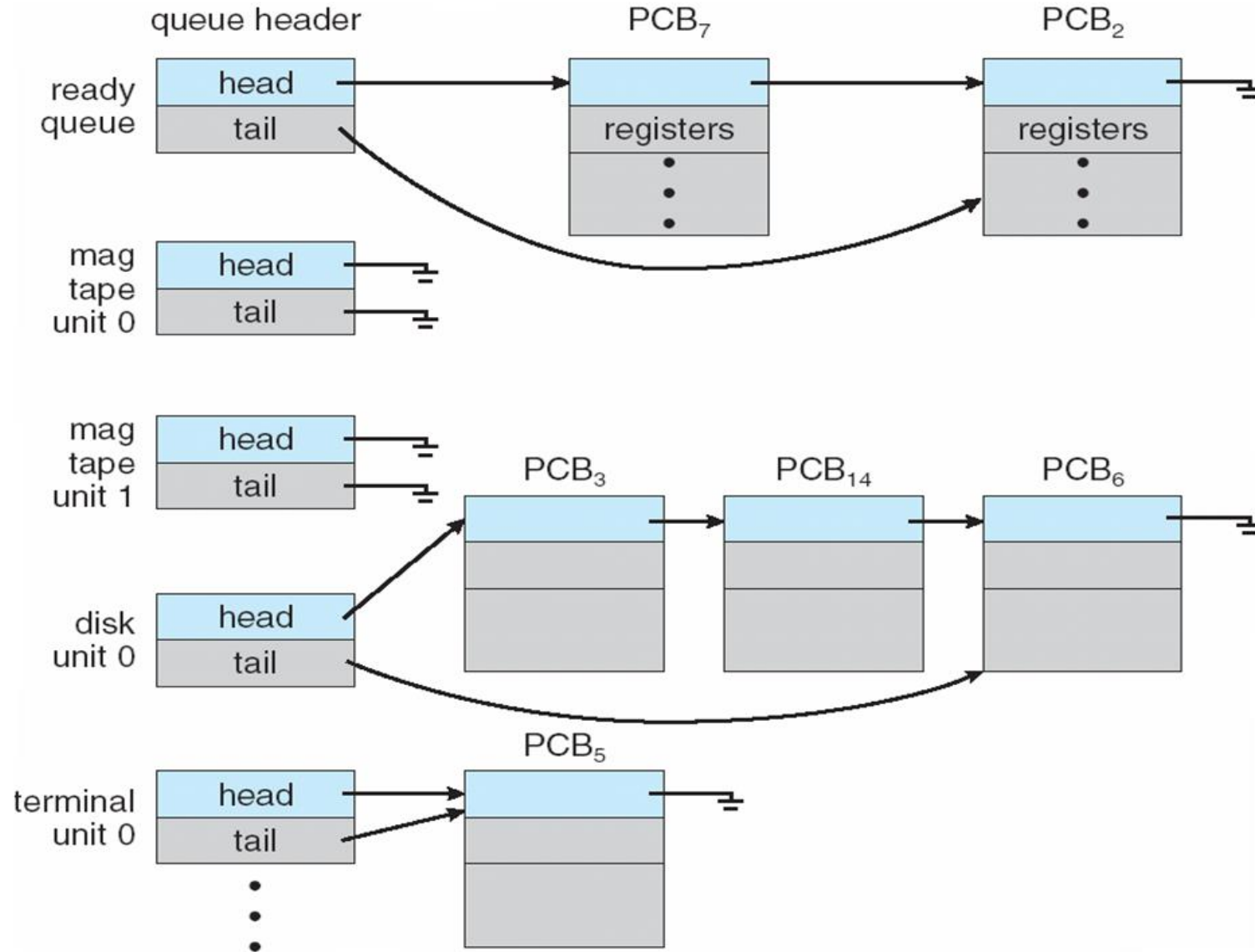
```
long state; /* state of the process */
struct sched_entity se; /* scheduling information */
struct task_struct *parent; /* this process's parent */
struct list_head children; /* this process's children */
struct files_struct *files; /* list of open files */
struct mm_struct *mm; /* address space */
```



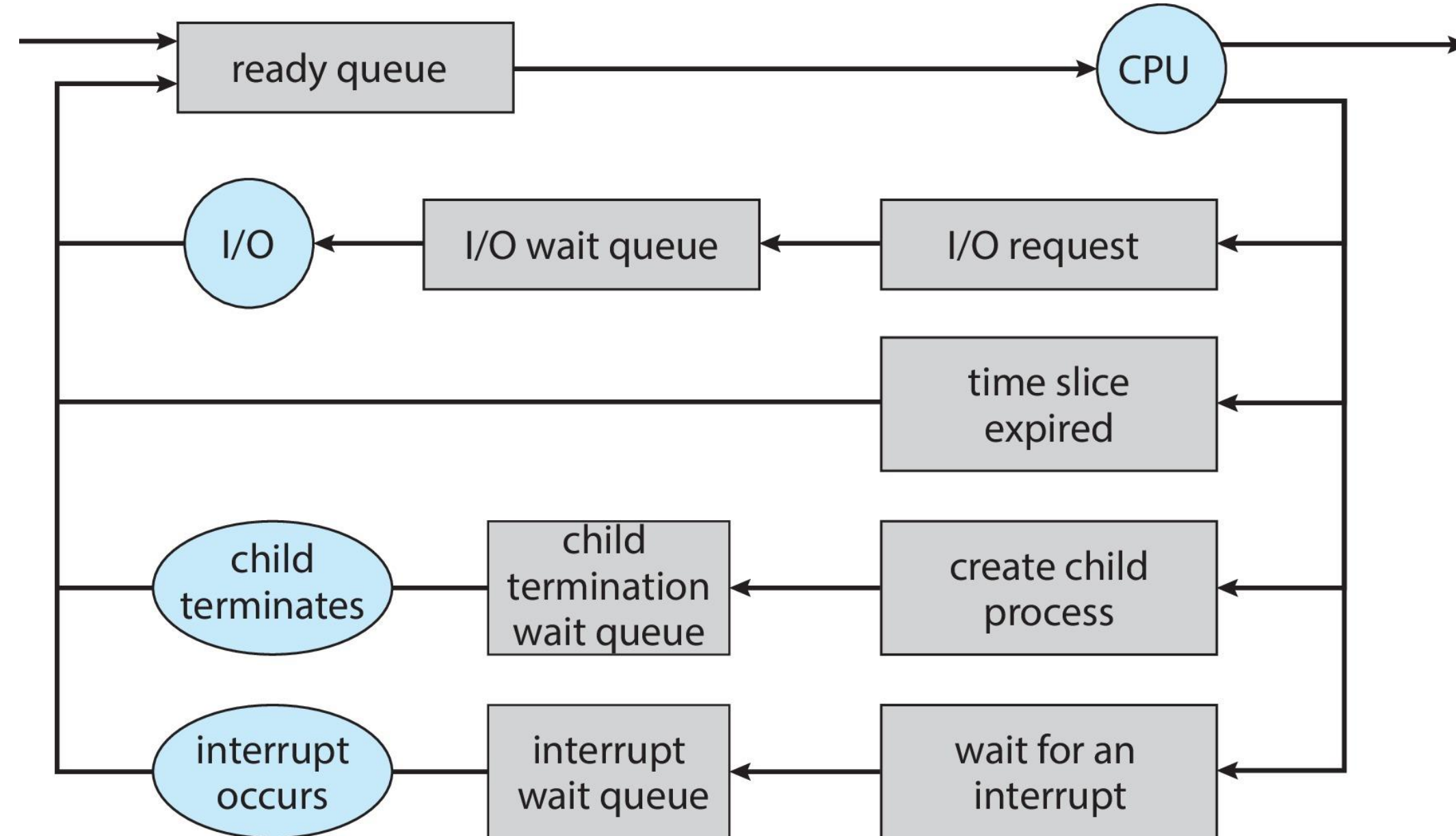
Proses Sıralama

- **Proses sıralayıcı (scheduler)**, işleme hazır prosesler arasından CPU çekirdeği üzerinde işlenecek prosesi seçer.
- Hedef - İşlemciyi azami kullan, prosesler arası geçiş ve zaman paylaşımını çok hızla gerçekleştir
- Prosesler aşağıdaki **sıralama kuyruklarında** tutulur.
 - **İş kuyruğu**– sistemdeki tüm prosesler kümesi
 - **Hazır kuyruğu**– ana bellekte bulunan, hazır ve işleme girmeyi bekleyen prosesler kümesi
 - **Bekleme(Aygıt) kuyrukları**– I/O aygıtlarından gelecek mesajı bekleyen prosesler kümesi
 - Prosesler kuyruklar arasında geçiş yapabilir.

Hazır Kuyruğu ve Çeşitli I/O Aygıt Kuyrukları

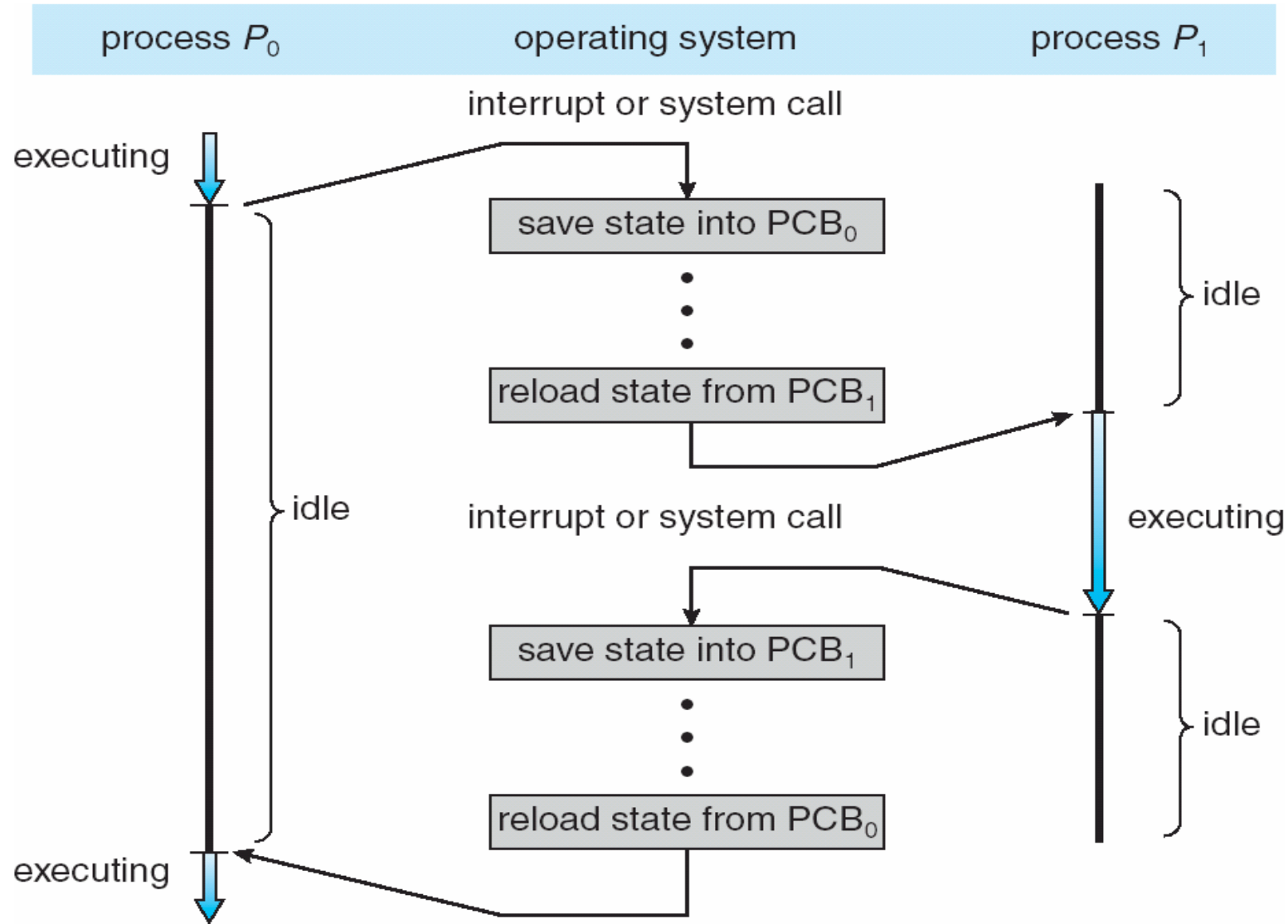


Proses Sıralama Diyagramı



CPU'da Proses Değişimi

CPU bir prosesden diğerine geçtiğinde bir bağlam değişimi (**context switch**) meydana gelir



Bağlam Değişimi (Context Switch)

- CPU diğer prosese geçtiği zaman, sistem mutlaka eski prosesin **durumunu kaydetmeli** ve **bağlam değişimi** yoluyla yeni prosesin daha önce **kaydedilmiş durumunu** yüklemeli
- Bir prosesini **bağlamı(context)** PCB ile temsil edilir
- Bağlam değiştirme bir maliyettir; sistem geçişler sırasında kullanım dışıdır
 - Daha karmaşık OS ve PCB -> daha uzun bağlam değişimi
- Donanım desteği zamana bağlıdır.
 - Bazı donanımlar CPU başına birden fazla kaydedici sağlar -> birden fazla bağlam bir kerede yüklenir

Mobil Sistemlerde Çoklugoörev

- Bazı mobil sistemler (örneğin, iOS'un erken sürümü) yalnızca bir işlemin çalışmasına izin verir, diğerleri askıya alınır.
- Ekran alanının darlığı nedeniyle, iOS'un sağladığı kullanıcı arabirimi sınırları
 - Kullanıcı arayüzü üzerinden kontrol edilen tek **ön plan** prosesi
 - Çoklu **arka plan** prosesleri– bellekte, çalışıyor ancak ekranda değil ve limitli
 - Sınırlar, tek, kısa görev, sadece olayların bildirimini alma, ses çalma gibi uzun süredir devam eden belirli görevleri içerir
- Android, daha az sınırla ön ve arka planda çalışır
 - Arka plan prosesi, görevleri gerçekleştirmek için bir **servis** kullanır
 - Arka plan prosesi askıya alınsa bile servis çalışmaya devam edebilir
 - Servisin kullanıcı arayüzü yok, küçük bellek kullanımı

Proses İşlemleri

Çoğu sistemde süreçler aynı anda yürütülebilir ve dinamik olarak oluşturulabilir ve silinebilirler.

Bu nedenle, bu sistemler

- Proses oluşturma ve
- Proses sonlandırma için bir mekanizma sağlamalıdır.

Bu bölümde, proses oluşturmaya inceleyerek UNIX ve Windows sistemlerinde proses oluşturmaya gösteren mekanizmaları ele alacağız.

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 türleri:

- Ebeveyn ve çocuk prosesler tüm kaynakları paylaşır.
- Çocuk prosesler ebeveyn prosesin kaynaklarını kullanır.
- Ebeveyn ve çocuk hiçbir kaynağı paylaşmaz.

Uygulama:

- Ebeveyn ve çocuk proses eşzamanlı çalışır
- Ebeveyn proses, çocuk proses sonlanana kadar bekler

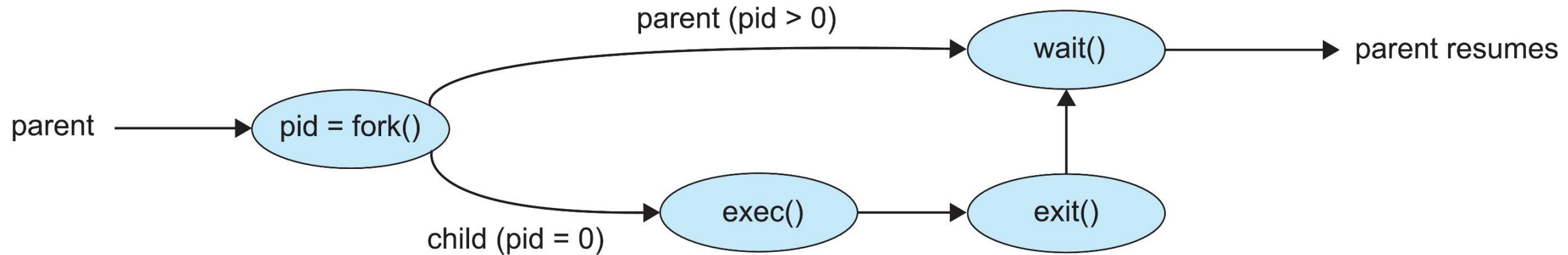
Proses Oluşturulması (Devam)

Adres alanı

- Çocuk proses, ebeveyn prosesin alanını kopyalar .
- Çocuk prosese bir program yüklenmiş olur.

UNIX örnekleri :

- **fork()** sistem çağrısı yeni bir proses oluşturur
- **exec()** sistem çağrısı prosesin bellek alanını yeni bir program ile değiştirmek için bir **fork()** sonrası çağrılır.
- Ebeveyn proses çocuğun sonlanmasını beklemek için **wait()** çağırır

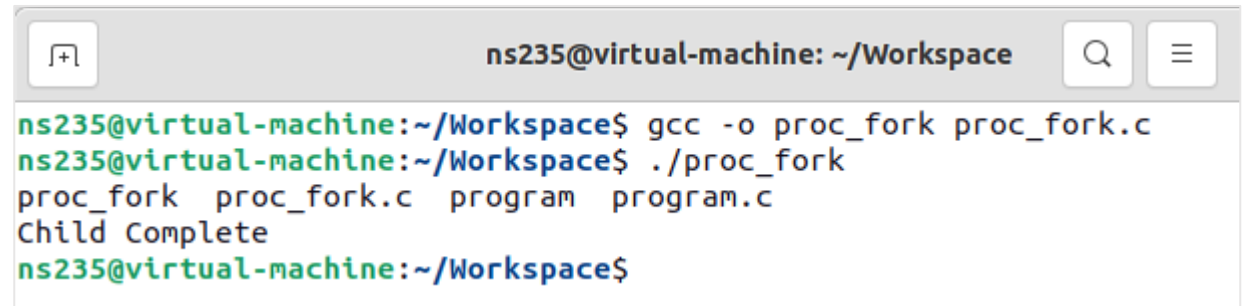


Fork İşlemi Yapan C Programı

```
#include <sys/types.h>
#include <sys/wait.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;
}
```

proc_fork.c isimli C kodunu linux üzerinde derleyerek çalıştıralım.

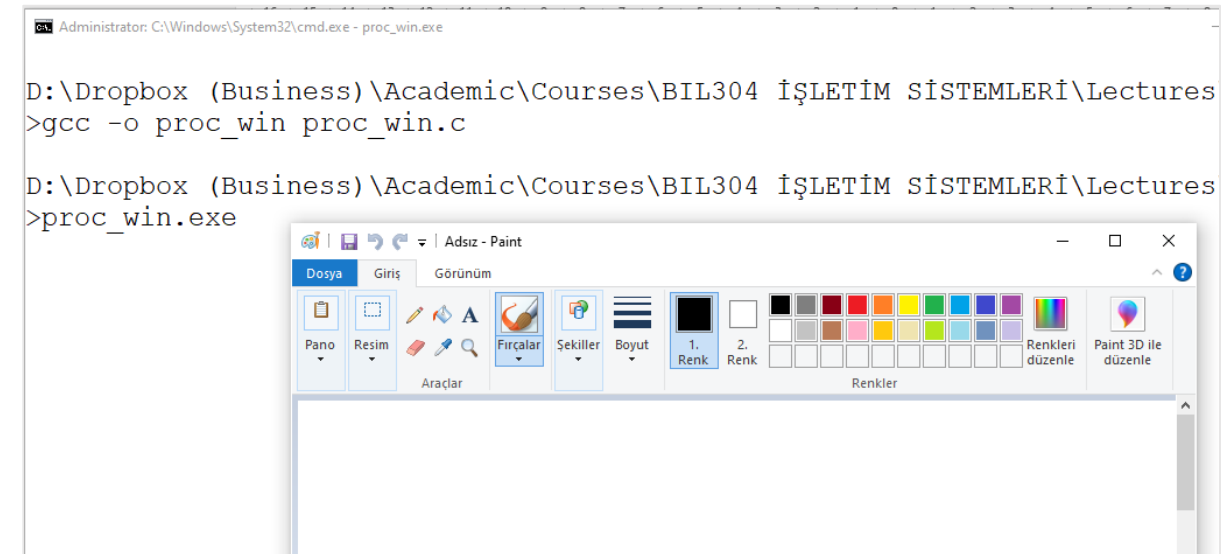


```
ns235@virtual-machine: ~/Workspace
ns235@virtual-machine:~/Workspace$ gcc -o proc_fork proc_fork.c
ns235@virtual-machine:~/Workspace$ ./proc_fork
proc_fork proc_fork.c program program.c
Child Complete
ns235@virtual-machine:~/Workspace$
```

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

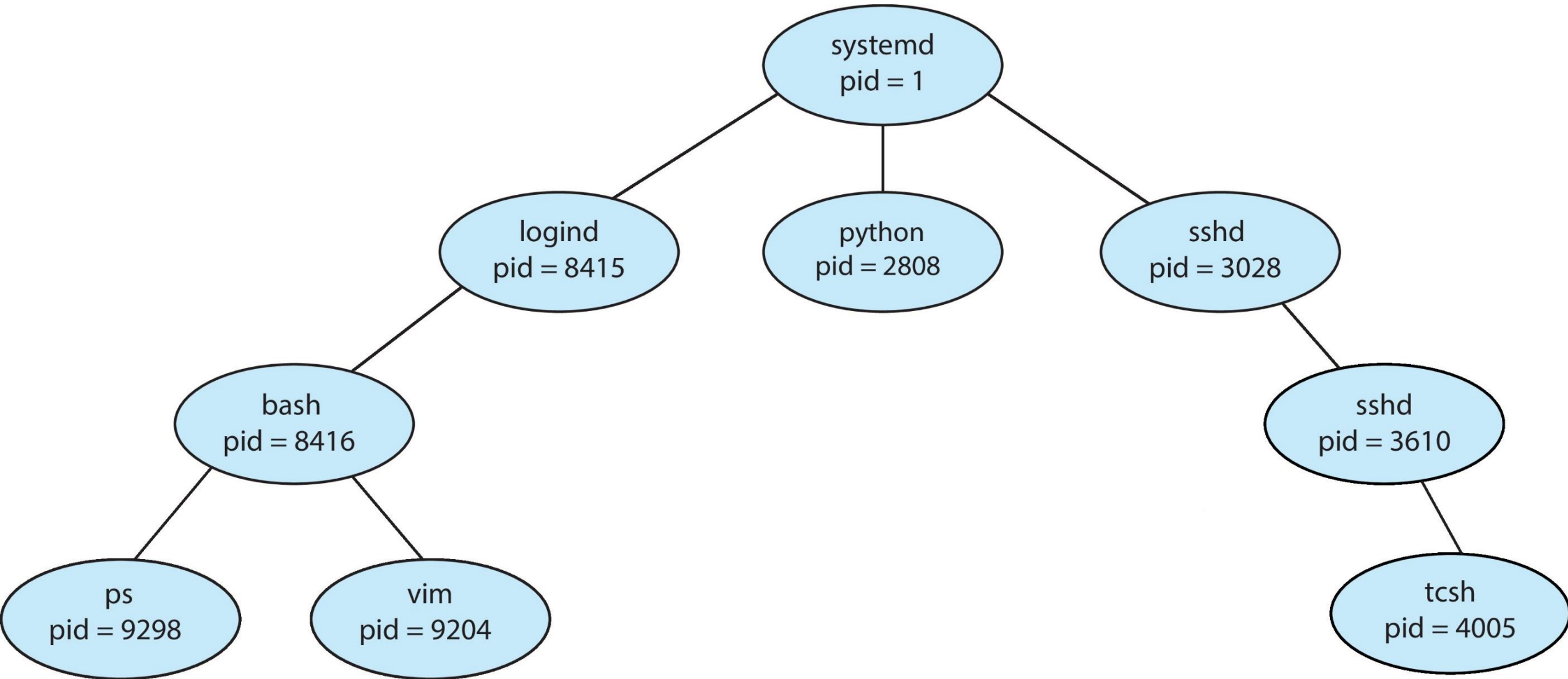
Windows API Aracılığıyla Ayır Proses Oluşturma

```
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);
}
```



proc_win.c isimli C kodunu windows üzerinde derleyerek çalıştırılın.

Linux Proses Ağacı



pstree -p

<https://linuxhandbook.com/show-process-tree/>

Proses Sonlanması

- Proses son kod ifadesinin çalıştırır ve işletim sistemine **exit ()** sistem çağrısını kullanarak silinmesini talep eder
 - Durum verisi çocuktan ebeveyn prosese döner (**wait () aracılığıyla**)
 - Prosese ayrılan alan işletim sistemi tarafından geri alınır
- Ebeveyn **abort ()** sistem çağrısını kullanarak çocuk proseslerin çalışmasını sonlandırabilir. Bunun sebepleri:
 - Çocuk tahsis edilen kaynakların dışına çıkmıştır.
 - Çocuk prosese tayin edilen iş artık gerekli değildir
 - Ebeveyn prosten çıkılır, ve işletim sistemi ebeveyn proses sonlandırıldıktan sonra çocuk prosesin çalışmasına izin vermez

Proses Sonlanması

- Bazı işletim sistemleri, ebeveyni sonlandırıldıysa çocuk prosesin var olmasına izin vermez. Bir proses sona ererse, tüm çocukları da sonlandırılmalıdır.
 - **basamaklı sonlandırma.** Tüm çocukların, torunların vs. çalışmasına son verilir.
 - Sonlandırma işletim sistemi tarafından başlatılır.
 - Ebeveyn proses **wait()** sistem çağrısını kullanarak bir çocuk prosesin sonlandırılmasını bekleyebilir. Çağrı durum bilgilerini ve sonlandırılan prosesin pid'sini döndürür

pid = wait(&status);

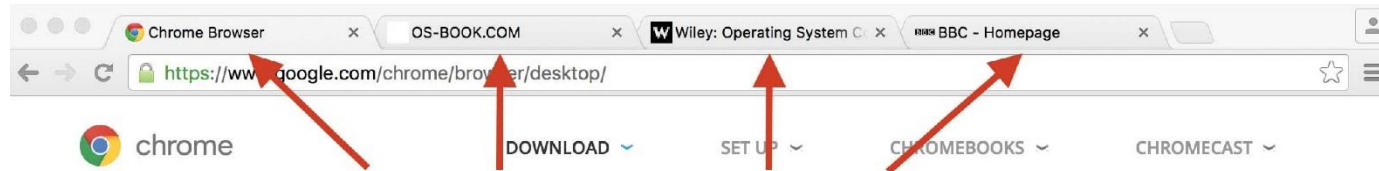
- Bekleyen hiçbir ebeveyn yoksa (wait() çağrılmadıysa) proses bir **zombidir**.
- Ebeveyn wait () çağrılmadan sonlandırıldıysa, proses bir **artıkdır(orphan)**

Android Proses Önem Hiyerarşisi

- Mobil işletim sistemleri genellikle bellek gibi sistem kaynaklarını geri kazanmak için prosesleri sonlandırmak zorundadır. En önemlisinden en az önemlisine:
 - Ön plan prosesi
 - Görünür proses
 - Servis prosesi
 - Arka plan prosesi
 - Boş proses
- Android, en az önemli olan prosesleri sonlandırmaya başlayacaktır.

Çoklu Proses Mimarisi – Chrome Tarayıcısı

- Birçok web tarayıcısı tek proses olarak çalıştırılır (bazıları hala öyle)
 - Bir web sitesi soruna neden olursa, tüm tarayıcı askıda kalabilir veya çökebilir
- Google Chrome Tarayıcı 3 farklı proses türüyle çoklu bir proses yapısındadır:
 - **Tarayıcı** prosesi kullanıcı arayüzünü yönetir, disk ve ağ G/Ç
 - **İşleyici (Renderer)** prosesi web sayfalarını işler, HTML, Javascript ile ilgilenir. Her web sitesi için oluşturulan yeni bir işleyici bulunur
 - Disk ve ağ G/Ç'yi kısıtlayan, güvenlik açıklarının etkisini en aza indiren korumalı (**sandbox**) bir yapıda çalışır
 - Her eklenti türü için **Plug-in** prosesi

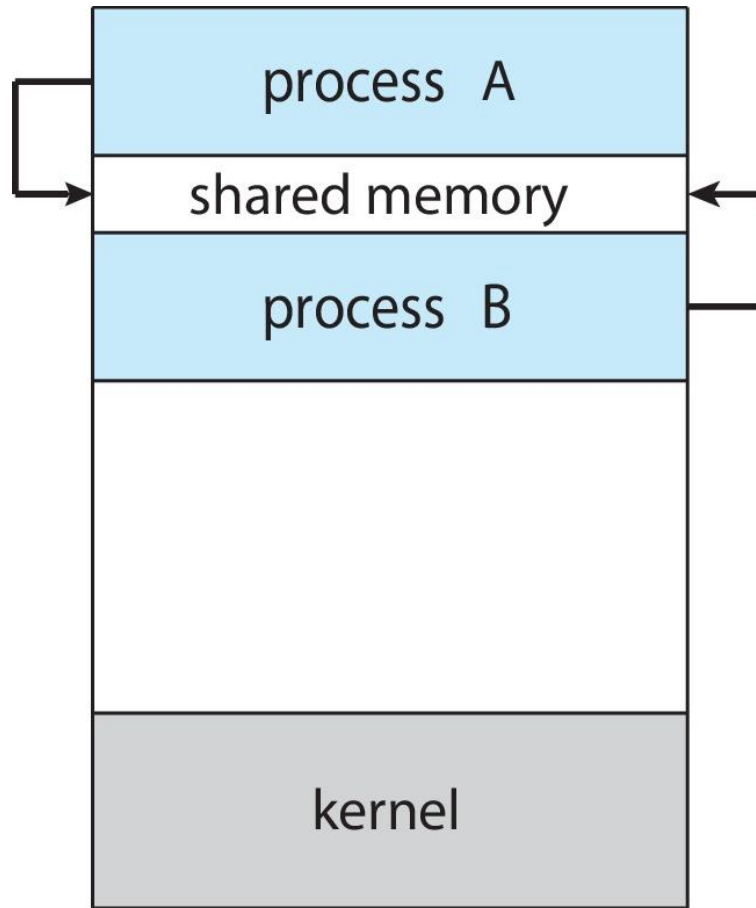


Prosesler Arası İletişim

- Prosesler işletim sistemi içerisinde **bağımsız** ya da **işbirliği halinde** 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
 - Rahatlık
- İşbirliği içindeki prosesler **prosesler arası haberleşmeye (Interproses communication - IPC)** ihtiyaç duyarlar.
- 2 temel IPC modeli mevcuttur:
 - **Paylaşımlı bellek** (kullanıcıların kontrolünde)
 - **Mesajlaş iletimi** (OS nin kontrolünde)

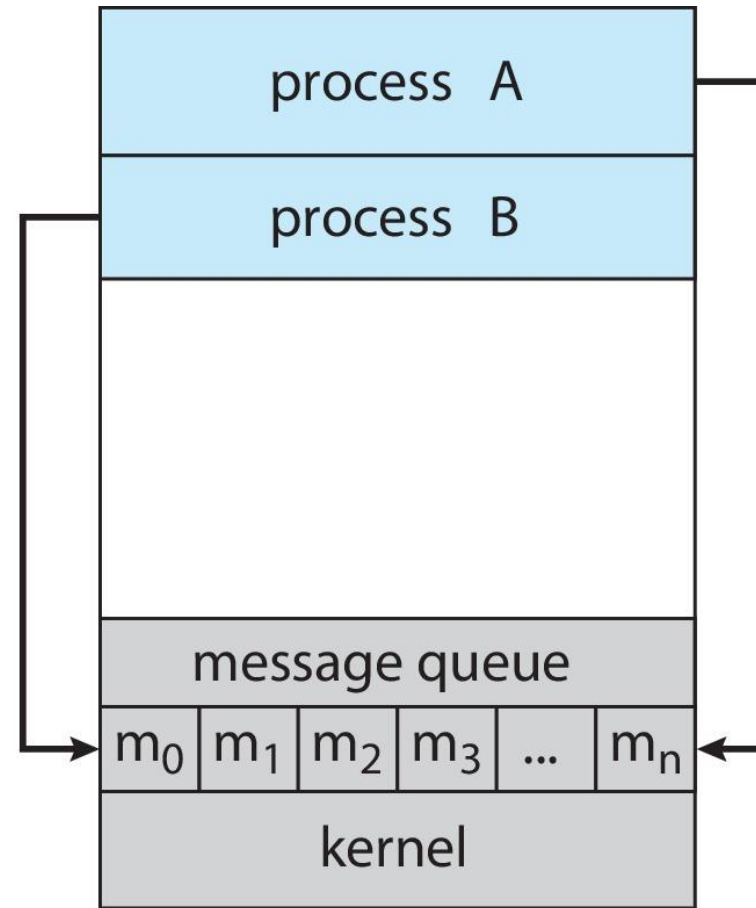
Haberleşme Modelleri

(a) Shared memory.



(a)

(b) Message passing.



(b)

Üretici-Tüketici Problemi

- İşbirliği içindeki proseslere ilişkin bir paradigma:
 - **üretici** proses **tüketici** proses tarafından kullanılmak üzere bilgi üretir.
- İki versiyonu vardır:
 - **Sınırlandırılmamış tampon**: tampon için limit konulmamıştır
 - Üretici asla beklemez
 - Tüketici tamponda ürün yoksa bekler
 - **Sınırlı tampon**: sabit bir tampon boyutu mevcuttur.
 - Üretici eğer tampon dolu ise beklemek zorunda
 - Tüketici tamponda ürün yoksa bekler

Paylaşımlı Bellek Çözümü

- İletişim kurmak isteyen prosesler arasında paylaşılan bir bellek alanı
- İletişim, işletim sisteminin değil, kullanıcı proseslerinin kontrolü altındadır.
- Önemli sorun, kullanıcı proseslerinin paylaşılan belleğe eriştiklerinde eylemlerini senkronize etmesine izin verecek bir mekanizma sağlamaktır.
- Senkronizasyon, Bölüm 6 ve 7'de ayrıntılı olarak ele alınacaktır

Sınırlı-Tamponlu- Paylaşımlı-Bellek Çözümü

- Paylaşılan veri

```
#define BUFFER_SIZE 10

typedef struct {

    . . .

} item;

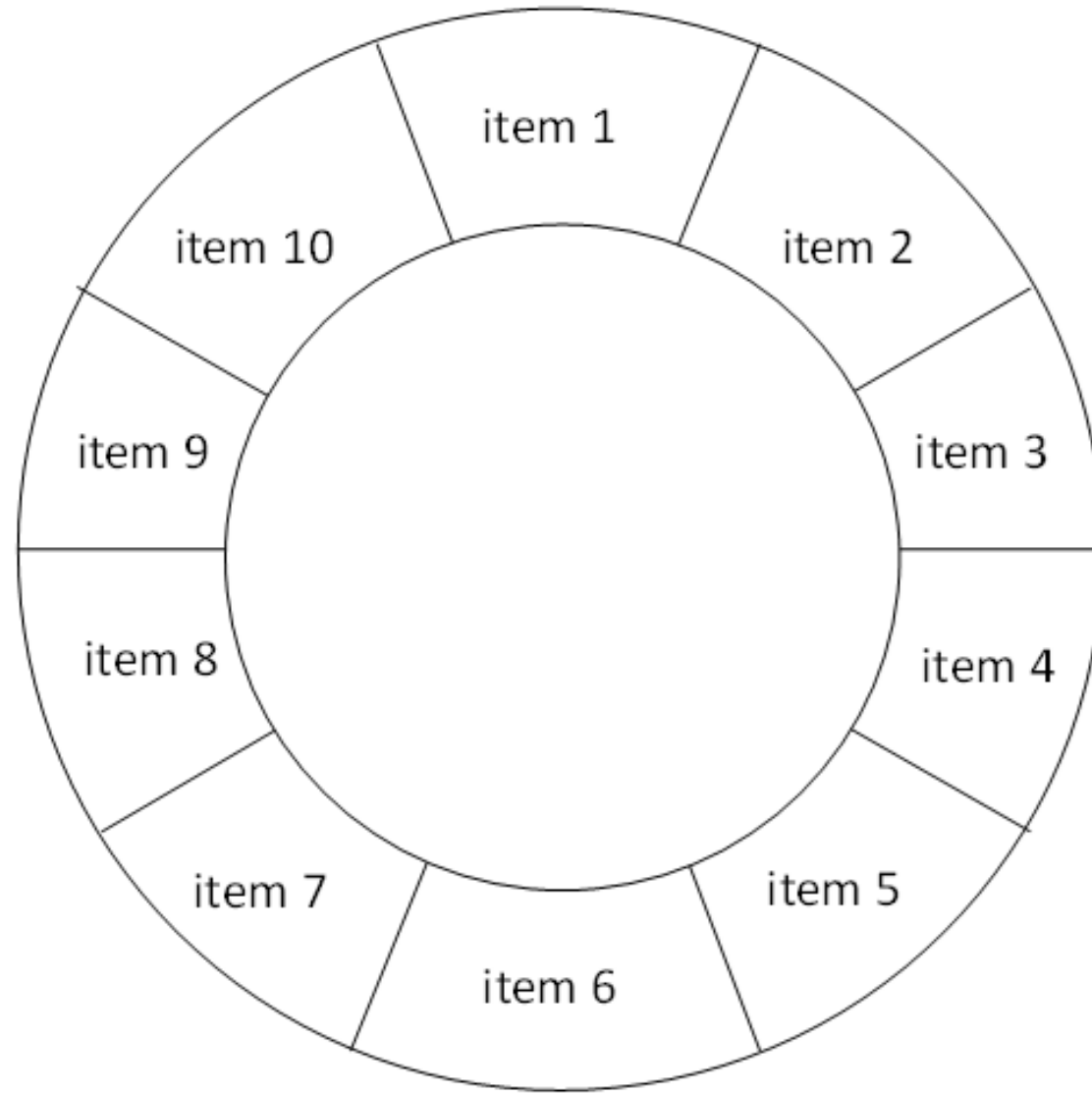
item buffer[BUFFER_SIZE];

int in = 0;

int out = 0;
```

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

Sınırlı-Tampon (devamı)



Sınırlı-Tampon – 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 Tampon – 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 */
}
```

Tamponlar dolduğunda ne olur?

- **Tüm** tamponları dolduran tüketici-üretici sorununa bir çözüm sunmak istediğimizi varsayalım.
- Bunu, tam tampon sayısını izleyen bir tamsayı **sayacı** ile yapabiliriz.
- Başlangıçta **sayacı** 0 olarak ayarlanır.
- Tamsayı **sayacı**, yeni bir tampon elemanı ürettikten sonra üretici tarafından artırılır.
- Tamsayı **sayacı**, tampon elemanını tükettikten sonra tüketici tarafından azaltılır ve azaltılır.

Üretici

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

Tüketici

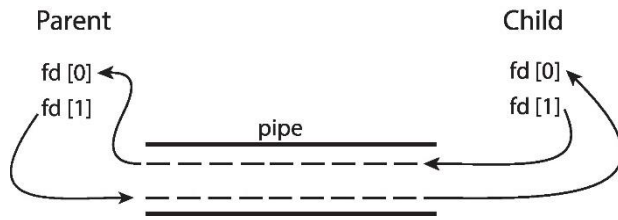
```
while (true) {  
    while (counter == 0)  
        ; /* do nothing */  
  
    next_consumed = buffer[out];  
  
    out = (out + 1) % BUFFER_SIZE;  
  
    counter--;  
  
    /* consume the item in next consumed */  
  
}
```

Veri Kanalları- Pipes

- İki proses arasında iletişime izin veren yapıdır.
- Sorunlar:
 - 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?
 - Kanallar ağ üzerinden kullanılabilir mi?
- **Sıradan kanallar** –oluşturan prosesin dışından erişilemez. Genellikle, bir ebeveyn proses bir kanal oluşturur ve çocuk prosesi ile iletişim kurmak için kullanır.
- **İsimli kanallar** – üst-alt ilişkisi olmadan erişilebilir.

Sıradan Kanallar

- **Sıradan kanallar**, standart üretici-tüketici tipi iletişime izin verir.
- Üretici bir uçtan yazar (kanalın yazma ucu)
- Tüketici diğer ucundan okur (kanalın okuma ucu)
- Sıradan kanallar bu nedenle tek yönlü iletişim sağlar.
- Haberleşen prosesler arasında ebeveyn-çocuk ilişkisi gerekir.



- Windows bunları **anonim kanallar** olarak adlandırır

İsimli Kanallar

- İsimli Kanallar, 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.

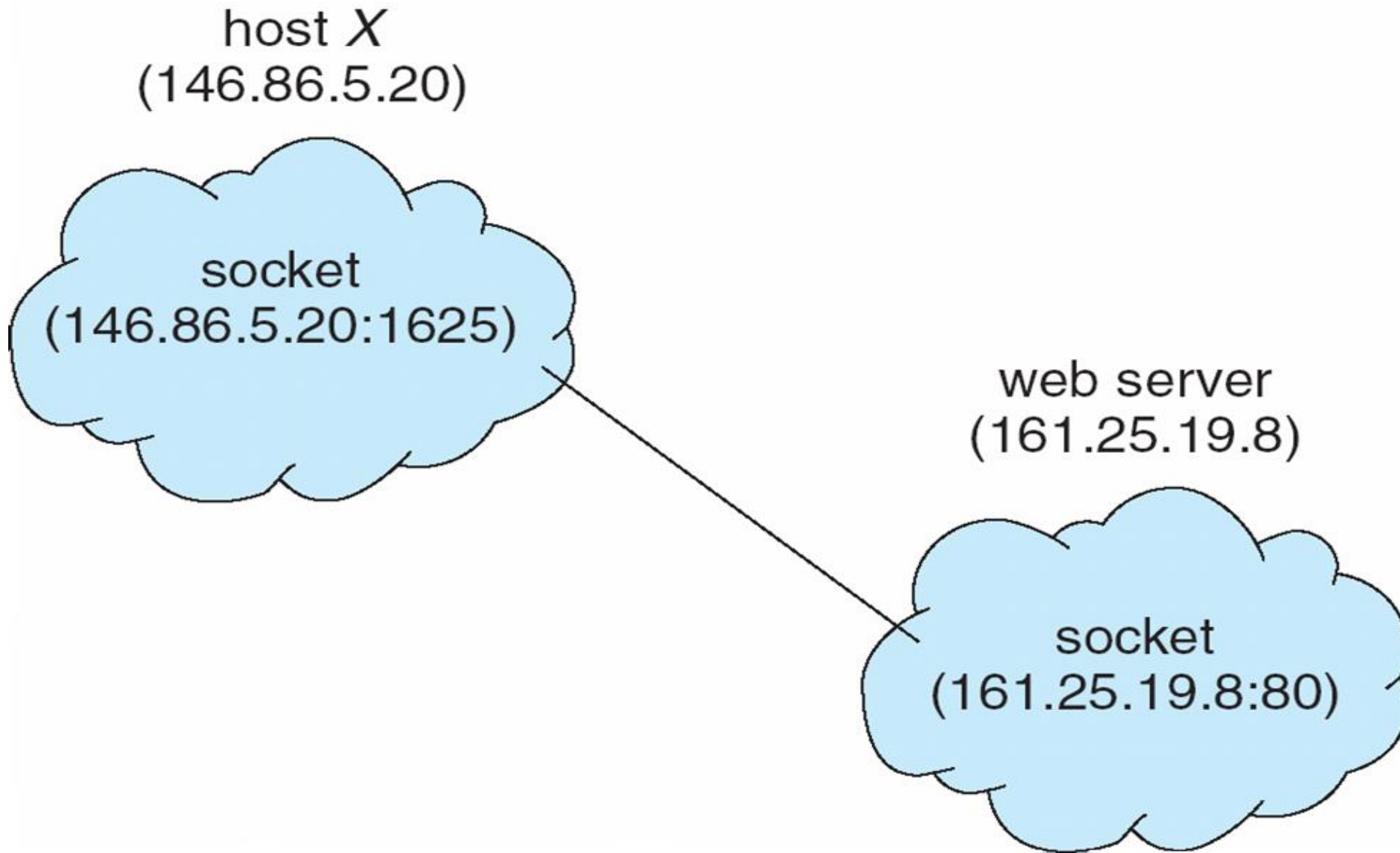
İstemci – Sunucu Sistemlerinde İletişim

- Soketler
- Uzaktan Prosedür Çağrılar
- Uzaktan Metot Çağrılar (RMI - Java)

Soketler

- Bir **soket**, bir iletişim uç noktası olarak tanımlanabilir.
- IP adresinin ve **portun** birleşimidir.
 - Port, bir ana bilgisayardaki ağ hizmetlerini ayırt etmek için mesaj paketinin başında bulunan bir sayıdır
- **161.25.19.8:1625** soketi, **1625** portu ve **161.25.19.8** sunucusu demektir.
- İletişim, bir çift soket arasında meydana gelir.
- 1024'ün altındaki tüm port numaraları iyi bilinmektedir, standart hizmetler için kullanılır
- Prosesin çalıştığı sisteme başvurmak için özel IP adresi 127.0.0.1 (geridöngü-**loopback**)

Soket İletişimi



Java Soketleri

- Üç tip soket
 - **Connection-oriented (TCP)**
 - **Connectionless (UDP)**
 - **MulticastSocket** class– data can be sent to multiple recipients
- Consider this “Date” server in Java:

Java Soketleri

DateServer.java

```
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);
        }
    }
}
```

DateClient.java

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

public class DateClient
{
    public static void main(String[] args) {
        try {
            /* make connection to server socket */
            Socket sock = new Socket("127.0.0.1", 6013);

            InputStream in = sock.getInputStream();
            BufferedReader bin = new
                BufferedReader(new InputStreamReader(in));

            /* read the date from the socket */
            String line;
            while ( (line = bin.readLine()) != null)
                System.out.println(line);

            /* close the socket connection*/
            sock.close();
        } catch (IOException ioe) {
            System.err.println(ioe);
        }
    }
}
```

Java Soket Uygulaması

Server.java

```
import java.io.IOException;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;
import java.text.SimpleDateFormat;
import java.util.Date;

public class Server {
    public static void main(String[] args) {
        try {
            ServerSocket serverSocket = new ServerSocket(12345);
            System.out.println("Sunucu başlatıldı. Bağlantı bekleniyor...");

            while (true) {
                Socket clientSocket = serverSocket.accept();
                System.out.println("Bir istemci bağlandı.");

                PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);

                // Tarih ve saat bilgisini al
                SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
                String date = dateFormat.format(new Date());

                // İstemciye tarih bilgisini gönder
                out.println(date);

                // Bağlantıyı kapat
                clientSocket.close();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Java Soket Uygulaması

Client.java

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.Socket;

public class Client {
    public static void main(String[] args) {
        try {
            Socket socket = new Socket("localhost", 12345);
            BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));

            // Sunucudan tarih bilgisini oku
            String date = in.readLine();
            System.out.println("Sunucu tarih bilgisi: " + date);

            // Bağlantıyı kapat
            socket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Java Soket Uygulaması

Data (D:) > Workspace > Java

Ad

Değiştirme tarihi

Tür

Boyut

Client.class

Client.java

Server.class

Server.java

Administrator: C:\Windows\System32\cmd.exe - Java Server

```
D:\Workspace\Java>Java Server
Sunucu başlatıldı. Bağlantı bekleniyor...
Bir istemci bağlandı.
```

Administrator: C:\Windows\System32\cmd.exe

```
D:\Workspace\Java>Java Client
Sunucu tarih bilgisi: 2024-03-11 02:59:31

D:\Workspace\Java>
```

```
import java.io.IOException;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;
import java.text.SimpleDateFormat;
import java.util.Date;

public class Server {
    public static void main(String[] args) {
        try {
            ServerSocket serverSocket = new ServerSocket(12345);
            System.out.println("Sunucu başlatıldı. Bağlantı bekleniyor...");

            while (true) {
                Socket clientSocket = serverSocket.accept();
                System.out.println("Bir istemci bağlandı.");

                PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);

                // Tarih ve saat bilgisini al
                SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
                String date = dateFormat.format(new Date());

                // İstemciye tarih bilgisini gönder
                out.println(date);

                // Bağlantıyı kapat
                clientSocket.close();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```


Uzak Yordam Çağrıları

- Uzak prosedür çağrısı (Remote procedure call - RPC), yordam çağrılarını bağlı sistemler üzerindeki işlemlere ayırır.
 - Servisleri ayırt etmek için portları kullanır
- **Stub** – sunucu üzerindeki gerçek prosedür için istemci tarafındaki aracı
- İstemci tarafındaki stub, sunucunun yerini belirler ve parametreleri yönlendirir.
- Sunucu tarafındaki stub, mesajı alır, yönlendirilmiş parametreleri açar ve yordamı sunucu üzerinde uygular.
- Windows' ta stub kodu **Microsoft Interface Definition Language (MIDL)** dili kullanılarak yazılır

Uzak Yordam Çağrıları (devam)

- Farklı mimariler için veri temsili **External Data Representation (XDL)** formatı ile yönetilir
 - **Big-endian** and **little-endian**
- Uzaktan iletişimde yerelden daha fazla hata senaryosu var
 - İletiler en fazla bir kez değil, tam olarak bir kez ve tam teslim edilir
- İşletim sistemi genellikle istemci ve sunucuyu bağlamak için bir randevu (veya **çöpçatan**) hizmeti sağlar

RPC'nin Çalışma Prensipleri

