

# İşletim Sistemleri

---

Hazırlayan: M. Ali Akcayol  
Gazi Üniversitesi  
Bilgisayar Mühendisliği Bölümü

Bu dersin sunumları, "Abraham Silberschatz, Greg Gagne, Peter B. Galvin, Operating System Concepts 9/e, Wiley, 2013." kitabı kullanılarak hazırlanmıştır.

## Konular

---

- **Process kavramı**
- Process planlama
- Process işlemleri
- Process'ler arası iletişim
- İstemci-sunucu sistemlerde iletişim

## Process kavramı

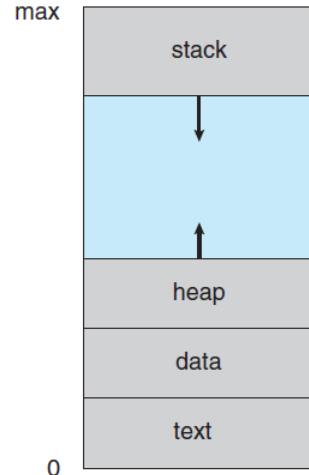
- Günümüz işletim sistemleri birden çok programın hafızaya yüklenmesine ve eşzamanlı çalıştırılmasına izin verir.
- Çalışmakta olan programa **process** denilmektedir.
- **Modern işletim sistemlerinde, process işin bir parçasıdır.**
- **CPU, aralarında geçiş yaparak tüm process'leri eş zamanlı çalıştırılabilir.**
- Bir sistem, tek kullanıcı bile olsa, birden fazla uygulamayı (Word, Excel, Web Browser, ...) birlikte çalıştırabilir.
- İşletim sistemi multitasking desteklemese bile, işletim sisteminin kendi fonksiyonlarını çağırarak çalıştırır.

3

## Process kavramı

### Process

- **Bir process, yürütmekte olduğu işi, program counter değerini, CPU register'larının değerlerini içermektedir.**
- Bir process aşağıdaki bileşenleri içermektedir:
  - **stack**, fonksiyon parametreleri, return adresleri ve lokal değişkenleri saklar.
  - **data section**, global değişkenleri saklar.
  - **heap**, process'e runtime'da dinamik olarak atanan bellektir.



4

## Process kavramı

### Process

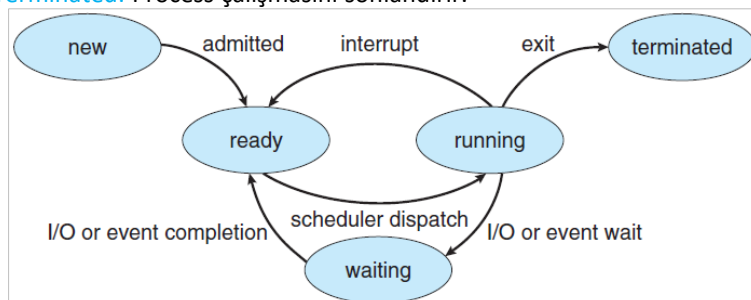
- **Bir program pasif varlıktır (executable file)**, disk üzerinde saklanan komut kümesidir.
- **Bir process aktif varlıktır**, sonraki çalışacak komut adresini program counter saklar.
- **Aynı program birden fazla process ile ilişkili olabilir** (birden fazla eşzamanlı çalışan Web browser).
- **Her process'in stack, data section ve heap kısımları farklıdır.**

5

## Process kavramı

### Process state

- **Bir process çalıştığı sürece durum değiştirir.**
  - **New:** Process oluşturulmaktadır.
  - **Running:** Komutlar çalıştırılmaktadır.
  - **Waiting:** Process bir olayın gerçekleşmesini beklemektedir (I/O, bir cihazdan geribildirim).
  - **Ready:** Process çalışmak için CPU'ya atanmak üzere bekliyor.
  - **Terminated:** Process çalışmasını sonlandırır.



6

## Process kavramı

### Process control block

- Her process, işletim sisteminde **process control block (PCB)** (veya **task control block**) tarafından temsil edilir.

|                    |
|--------------------|
| process state      |
| process number     |
| program counter    |
| registers          |
| memory limits      |
| list of open files |
| ...                |

7

## Process kavramı

### Process control block

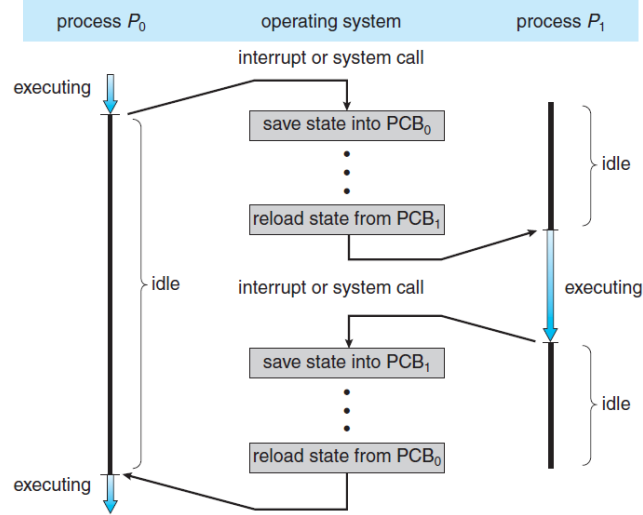
- Bir PCB aşağıdaki bilgilerle ilişkilendirilmiştir:
  - Process state:** Durum, new, ready, running, waiting, halted olabilir.
  - Program counter:** Bu process için sonraki komutun adresini gösterir.
  - CPU register'ları:** CPU'ya göre değişen tür ve boyutta register'lar vardır. (accumulators, index registers, stack pointers, general-purpose registers, condition codes, ...)
  - CPU-scheduling information:** Process önceliğini içerir.
  - Memory-management information:** Base ve limit register'ları, sayfa ve segment tabloları, ... içerir.
  - Accounting information:** CPU kullanım oranları, account bilgileri ve process numaralarını içerir.
  - I/O status information:** Process'lere tahsis edilmiş I/O cihazları ile açık durumdaki dosyaları içerir.

8

## Process kavramı

### Process control block

- CPU'nun process'ler arasında geçişi şekilde görülmektedir.



## Process kavramı

### Threads

- **Tek thread ile bir process kontrol edilir ve birden fazla görev aynı anda yapılamaz** (karakter girişi ile spell check aynı anda yapılamaz.).
- **Modern işletim sistemlerinde bir process ile birden fazla thread çalıştırılmasına izin verilir.**
- **Bu özellik multicore işlemcilerde çok faydalıdır** ve çok thread eşzamanlı çalıştırılır.
- Çok thread ile çalışan sistemlerde, PCB ile her bir thread'e ait bilgiler saklanır.

## Konular

- Process kavramı
- **Process planlama**
- Process işlemleri
- Process'ler arası iletişim
- İstemci-sunucu sistemlerde iletişim

11

## Process planlama

- Çok programlı sistemlerin temel amacı, **CPU kullanımını maksimuma çıkaracak şekilde process'leri çalıştırmaktır.**
- Zaman paylaşımlı sistemlerde CPU çok kısa aralıklarla process'ler arasında geçiş yapar.
- Bir process'i CPU'da çalışması için **process scheduler** seçer.

12

## Process planlama

### Scheduling queues

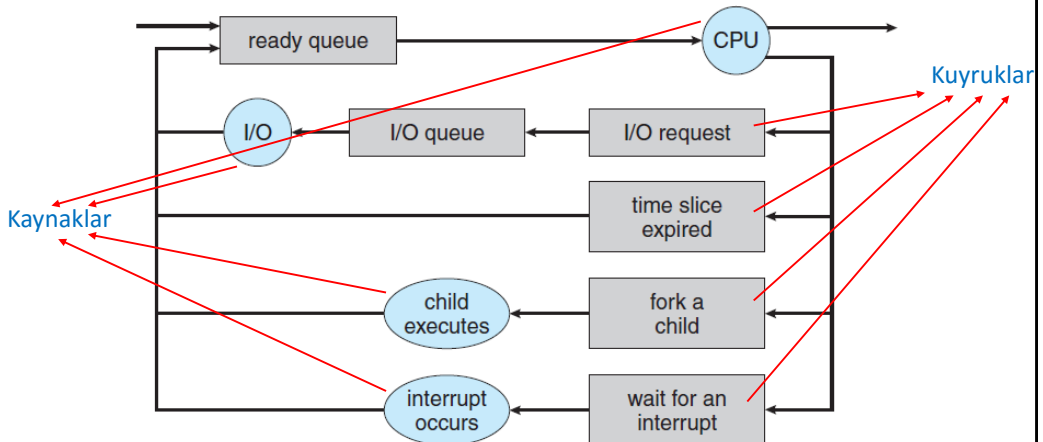
- Bir process sisteme girdiğinde, tüm işlerin bulunduğu iş kuyruğuna (**job queue**) alınır.
- Hafızaya alınmış ve çalışmayı bekleyen process'ler hazır kuyruğuna (**ready queue**) alınır.
- Kuyruk yapıları genellikle **linked list** veri yapısı ile gerçekleştirilir.
- Ready queue, listedeki PCB'lerin ilk ve son elemanlarını işaret eder.
- Bir sistemde hazır kuyruğu dışında I/O cihazları için de kuyruk (**device queue**) bulunur.

13

## Process planlama

### Scheduling queues

- Process planlama için aşağıdaki şekilde verilen **queueing diagram** yaygın kullanılan gösterimdir.



14

## Process planlama

### *Scheduling queues*

- Bir process **I/O isteğinde bulunursa, I/O kuyruğına aktarılır.**
- Bir process başka bir process'i çalıştırırsa onun bitmesini bekler.
- Bir process çalışması için **ayrılan süre tamamlanırsa** CPU tarafından **tekrar hazır kuyruğunun sonuna alınır.**
- Bir process interrupt beklemeye başlarsa **interrupt kuyruğına alınır.**

15

## Process planlama

### *Schedulers*

- Bir process, çalışma süresi boyunca farklı kuyruklara alınabilir.
- Kuyruktaki process'lerin seçilmesi **scheduler** tarafından gerçekleştirilir.
- **Genellikle batch sistemlerde çok sayıda process çalıştırılmak üzere sisteme gönderilir.**
- Bu process'ler disk üzerinde biriktirilir ve daha sonra çalıştırılır.
- **Long-term scheduler** (veya **job scheduler**) bu işleri seçerek çalıştırılmak üzere hafızaya yükler.
- **Short-term scheduler** (veya **CPU scheduler**) bu işlerden çalıştırılmak üzere hazır olanları seçerek CPU'yu onlara tahsis eder.
- Short-term scheduler çok kısa aralıklarla (<100ms) ve sıklıkla çalıştırılır. Long-term scheduler ise dakika seviyesindeki aralıklarla çalıştırılır.

16



## Process planlama

### Schedulers

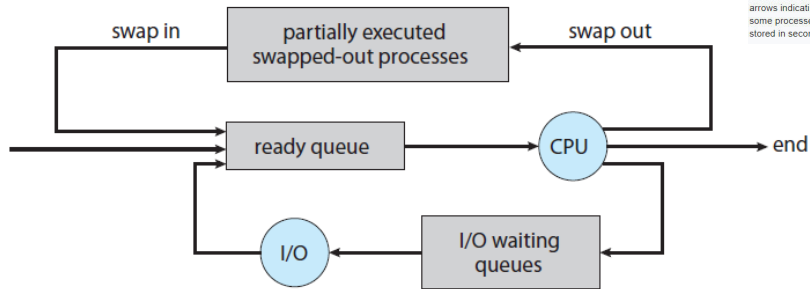
- Process'ler **I/O-bound** ve **CPU-bound** olarak iki gruba ayrılır.
- I/O-bound process'ler I/O işlemleri için daha fazla süre ayırırlar.
- CPU-bound process'ler CPU ile hesaplama işlemleri için daha fazla süre ayırırlar.
- Eğer tüm process'ler I/O-bound olursa, hafızadaki ready queue hemen hemen her zaman boş kalır ve short-term scheduler çok az çalışır.
- Eğer tüm process'ler CPU-bound olursa, I/O waiting queue hemen hemen her zaman boş kalır ve cihazlar kullanılmadan boş kalır.
- Her iki durumda da sistem dengesiz iş dağılımına sahip olur.
- Windows ve Unix işletim sistemlerinde long-term scheduler bulunmaz ve sadece short-term scheduler kullanılır.

17

## Process planlama

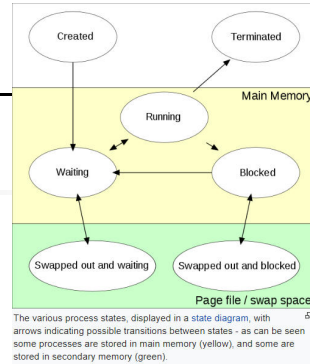
### Schedulers

- Bazı işletim sistemleri **medium-term scheduler** kullanır.



- Hafıza gereksiniminin değişmesi gibi bazı durumlarda, process'ler hafızadan atılır ve daha sonra tekrar hafızaya alınır (swapping).

18



A running process may become suspended if it makes an I/O request. A suspended processes cannot make any progress towards completion. In this condition, to remove the process from memory and make space for other processes, the suspended process is moved to the secondary storage. This process is called swapping, and the process is said to be swapped out or rolled out. Swapping may be necessary to improve the process mix.

## Process planlama

### Context switch

- **Interrupt'lar**, CPU'nun yürütmekte olduğu **bir görevden** işletim sisteminin **kernel fonksiyonuna geçmesine neden olurlar**.
- Bir interrupt gerçekleştiğinde, mevcut konfigürasyon (**context**) saklanır ve geri döndüğünde yeniden aynı içeriğe dönülerek devam edilir.
- **Bir process için context, program control block (PCB) içerisinde saklanır.**
- Context, **CPU register'larının değerleri, hafıza yönetim bilgileri, process state bilgisini içerir.**
- **CPU'nun bir process'ten başka bir process'e geçmesine context switch denilmektedir.**
- **Context switch süresi**, bir iş üretilmediği için **overhead** olarak adlandırılır ve genellikle birkaç milisaniyedir.

19

## Konular

- Process kavramı
- Process planlama
- **Process işlemleri**
- Process'ler arası iletişim
- İstemci-sunucu sistemlerde iletişim

20

## Process işlemleri

### *Process creation*

- Process'ler dinamik olarak oluşturulurlar ve silinirler.
- **Bir process çalışması sırasında birkaç tane başka process'i çalıştırabilir.**
- Çağırın process **parent**, yeni oluşturulan process **child** olarak adlandırılır.
- Unix, Linux ve Windows gibi işletim sistemleri, her process için **process identifier (pid)** değeri atarlar.
- Her process için atanan değer tekildir (**unique**) ve process'e erişim için kullanılır.

21


## Process işlemleri

### *Process creation*

- Bir parent process, yeni bir child process oluşturduğunda, **yeni child process CPU time, hafıza, dosyalar ve I/O cihazları gibi kaynaklara ihtiyaç duyar.**
- **Parent process, kendi kaynaklarını child process'lere paylaştırabilir veya işletim sistemi tarafından child process'e yeni kaynak tahsis edilebilir.**
- Bir parent process, child process başlattığında sonlana kadar bekleyebilir veya eşzamanlı çalışmasını sürdürebilir.

**A child process is a process created by a parent process in operating system using a fork() system call. A child process may also be called a subprocess or a subtask. A child process is created as its parent process's copy and inherits most of its attributes**

22



## Process işlemleri

### Process creation - Unix

Yeni process için sistem çağrısı

Yeni process için hata oluştu.

Yeni child process başlatıldı.

Dizin listesini ekrana yazan process.

Parent process (pid > 0)

Parent process, child process'i bekliyor.

```

#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;
}

```

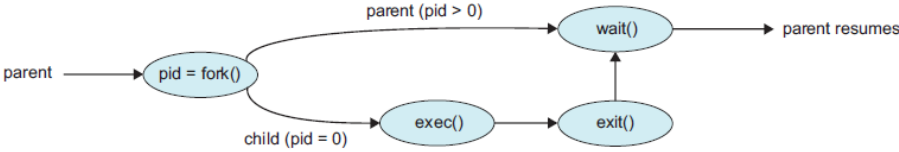
The `execlp()` function replaces the current process image with a new process image specified by file. The new image is constructed from a regular, executable file called the new process image file. No return is made because the calling process image is replaced by the new process image.



## Process işlemleri

### Process creation - Unix

- Child process `exit()` ile sonlanıncaya kadar parent process bekler.
- Şekilde parent process'in child process'i beklemesi görülmektedir.



```

graph LR
    parent --> fork(pid = fork())
    fork -- "parent (pid > 0)" --> wait1(wait())
    fork -- "child (pid = 0)" --> exec(exec())
    exec --> exit(exit())
    exit --> wait1
    wait1 --> resumes(parent resumes)

```

24

## Process işlemleri

### Process creation - Windows

Yeni process için sistem çağrısı

Yeni process "mspaint.exe"

Yeni child process özellikleri  
(Window size, giriş/çıkış dosyaları, ...)

Process identifier

Process oluşturma hatası

Parent process, child process'i bekliyor.

```

#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);
}

```

**LPSTARTUPINFOA**  
**lpStartupInfo,**  
  
**LPPROCESS\_INFORMATION**  
**lpProcessInformation**

## Process işlemleri

### Process termination

- Bir process son deyimini çalıştırıp tamamlandığında, exit() sistem çağrısını çalıştırır ve hafızadan silinmesini ister.
- **Sonlanan process, parent process'e durum bilgisini gösteren değer (integer) döndürebilir.**
- Bir parent process, child process'in sonlanmasına da neden olabilir (Windows'ta TerminateProcess() sistem çağrısı).
  - Child process kaynak kullanım sınırını aştığında, parent process tarafından sonlandırılabilir.
  - Child process'in yaptığı işe gerek kalmayabilir.
  - İşletim sistemi parent process'i sonlandırdığında child process'lerin de sonlandırılmasını isteyebilir.
- **exit(); doğrudan, return int; dolaylı sonlandırma yapar.**

## Konular

- Process kavramı
- Process planlama
- Process işlemleri
- **Process'ler arası iletişim**
- İstemci-sunucu sistemlerde iletişim

27

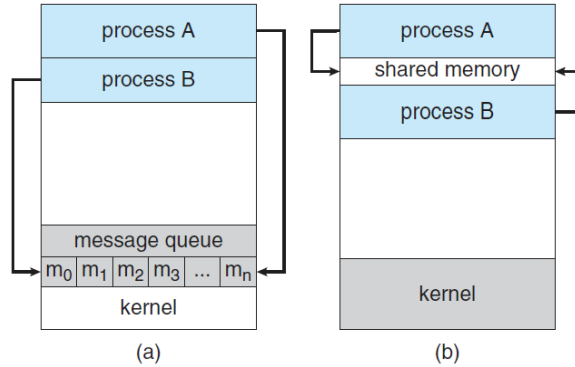
## Process'ler arası iletişim

- Process'ler işletim sisteminde **independent process** veya **cooperating process** olarak çalışırlar.
- **Independent process'ler** diğer process'leri etkilemezler ve onlardan etkilenmezler.
- **Cooperating process'ler** diğer process'leri etkilerler ve onlardan etkilenirler (diğer process'lerle veri paylaşımı yaparlar).
  - **Information sharing:** Paylaşılmamış dosyalar üzerinde işlem yapmak gerekebilir.
  - **Computation speedup:** Birden fazla core'a sahip işlemcili bilgisayarlarda, görevler parçalar halinde eşzamanlı yürütülürler.
  - **Modularity:** Sistem parçalar (process'ler, thread'ler) halinde oluşturulabilir ve bu parçalar arasında iletişim yapılabilir.
  - **Convenience:** Bir kullanıcı farklı işleri (müzik dinleme, metin yazma, compile, ...) aynı anda gerçekleştirebilir.

28

## Process'ler arası iletişim

- Cooperating process'ler **shared memory** ve **message passing** modelleri ile veri aktarımı yaparlar. **Shared memory modeli daha hızlıdır!!!**
  - Shared memory modelinde, **hafızada bir bölge process'ler arasında paylaşılır.**
  - Message passing modelinde, **process'ler arasında mesaj ile veri gönderilir.**



Communications models. (a) Message passing. (b) Shared memory.

29

## Process'ler arası iletişim

### Shared memory

- Shared memory modelinde, **producer** veriyi oluşturur ve paylaşılmış hafıza alanına yazar, **consumer** ise veriyi okuyarak kullanır.
- Compiler**, bir programı derler ve assembly kod üretir, **assembler** bu kodu alır ve object kod üretir, **loader** ise bu kodu giriş olarak alır.
- Shared buffer aşağıdaki kod ile tanımlanır:

```
#define BUFFER_SIZE 10

typedef struct {
    . . .
} item;

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

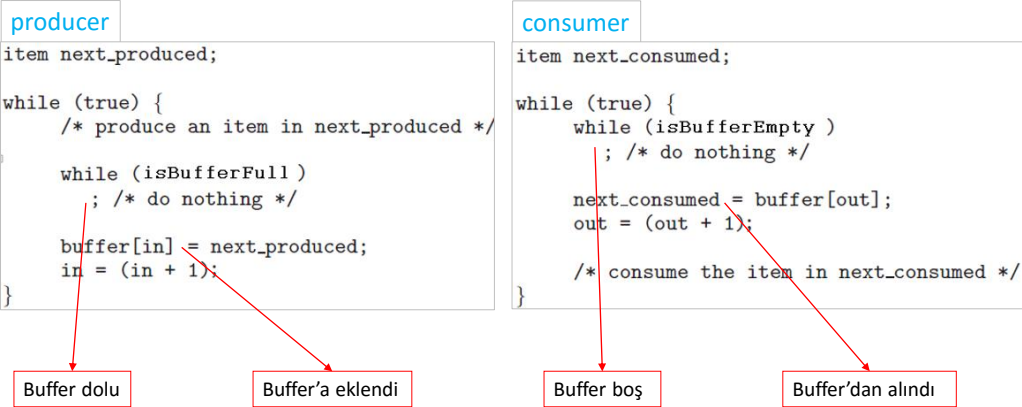
- Buffer dizi şeklinde oluşturulur (**dairesel bağlı liste kullanılabilir**) ve **in** değişkeni sonraki boş yeri, **out** ise ilk dolu yeri gösterir.

30

## Process'ler arası iletişim

### Shared memory

- Shared memory modelinde, **producer** veriyi oluşturur ve paylaşılmış hafıza alanına yazar, **consumer** ise veriyi okuyarak kullanır.



31

## Process'ler arası iletişim

### Message passing

- Message passing modeli, **dağıtık ortamlardaki process'lerin** (örn. chat programı) **iletişiminde faydalıdır**.
- Message passing modelinde en az iki işlem tanımlanır:
  - send(message)**
  - receive(message)**
- Mesaj boyutları sabit uzunlukta veya değişken uzunlukta olabilir.
- Process'ler birbirlerini doğrudan isimleriyle adresleyerek mesaj gönderirler:**
  - send(P, message)** // P process'ine mesaj gönderilir.
  - receive(Q, message)** // Q process'inden mesaj alınır.

32



## Konular

- Process kavramı
- Process planlama
- Process işlemleri
- Process'ler arası iletişim
- İstemci-sunucu sistemlerde iletişim

33

## İstemci-sunucu sistemde iletişim

### Soketler

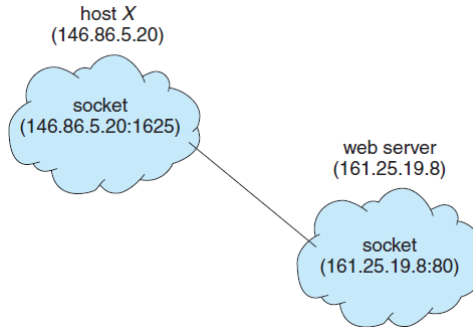
- Bir soket iletişim için uç noktayı (process) tanımlar.
- Bir ağ üzerinde haberleşen iki process'in her biri bir sokete sahiptir.
- Bir soket, IP adresi ve port numarasıyla tanımlanır.
- Sunucu, bir portu dinleyerek gelen istekleri bekler.
- Sunucuya bir istek geldiğinde alır ve gerekli işlemleri başlatır.
- FTP, HTTP gibi protokoller ayrılmış port numaralarına sahiptir (HTTP için 80, FTP için 21).
- Bir host üzerindeki tüm process'ler için bağlantıların tekil olması zorunludur.
- Tüm process'ler için işletim sisteminin atadığı port numaraları farklı olmak zorundadır.

34

## İstemci-sunucu sistemde iletişim

### Soketler

- IP adresi **146.86.5.20** olan istemci host üzerindeki process port numarası olarak **1625**'e sahiptir.
- Web sunucu **161.25.19.8** IP adresine ve sunucu process **80** port numarasına sahiptir.
- **soket çiftleri (146.86.5.20:1625) ile (161.25.19.8:80) olacaktır.**



35

## İstemci-sunucu sistemde iletişim

### Soketler

- Soketler arasında iki tür bağlantı yapılmaktadır:
  - **Connection-oriented (reliable)**
  - **Connectionless (unreliable)**
- Reliable iletişim **TCP (Transmission Control Protocol)** ile, unreliable iletişim ise **UDP (User Datagram Protocol)** ile gerçekleştirilir.
- Java programlama dilinde TCP bağlantısı **Socket** sınıfı ile, UDP bağlantısı **DatagramSocket** sınıfı ile gerçekleştirilir.


36

## İstemci-sunucu sistemde iletişim

### Soketler – Örnek

- Sunucu 6013 portunu dinler.
- İstemcilerden gelen isteklere tarih ve saat bilgisini cevap olarak gönderir.
- Sunucu **ServerSocket** nesnesi oluşturarak **accept() metodu ile 6013 portunu dinlemektedir.**
- **PrintWriter** nesnesi bir sokete **print()** veya **println()** metotları ile yazma işlemi yapar.
- İstemci process, sunucu process ile belirlenmiş port üzerinden bağlantı yapar.
- İstemci Socket nesnesi oluşturur ve **127.0.0.1** IP adresinden **6013** portu ile bağlantı yapar.
- 127.0.0.1 IP adresi **loopback** olarak adlandırılır ve kendisini gösterir.

37



**Sunucu process**

Sunucu **ServerSocket** nesnesi oluşturarak **accept()** metodu ile **6013 portunu dinlemektedir.**

**PrintWriter** nesnesi bir sokete **print()** veya **println()** metotları ile yazma işlemi yapar.

İstemci ile bağlantı kapatılır.

```
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);
        }
    }
}
```

## İstemci process

İstemci **Socket** nesnesi oluşturarak **127.0.0.1** IP adresinde **6013** portuyla bağlantı yapar.

**BufferedReader** nesnesi ile nesnesi soketten okuma bağlantısı tanımlanır.

İstemci gelen veriyi okur ve ekrana yazar.

Bağlantı kapatılır.

```
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);

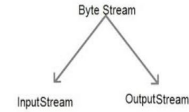
            /* close the socket connection*/
            sock.close();
        }
        catch (IOException ioe) {
            System.err.println(ioe);
        }
    }
}
```

first thing you need to understand is what is a **STREAM**

A stream can be defined as a sequence of data. The **InputStream** is used to read data from a source and the **OutputStream** is used for writing data to a destination.

\*\*\*\*Next is type of streams\*\*\*\*

we have byte stream and character stream.



classes we have in **Input Stream** and **output stream**

