

YZM 3102

İşletim Sistemleri

Yrd. Doç. Dr. Deniz KILINÇ

Celal Bayar Üniversitesi

Hasan Ferdi Turgutlu Teknoloji Fakültesi

Yazılım Mühendisliği

BÖLÜM - 2

Bu bölümde,

- İşletim Sistemi Servisleri
- CLI ve GUI
- Sistem Çağrılar
- Sistem Çağrısı Implementasyonu
- Sistem Çağrısı Türleri
- Sistem Programları
- İşletim Sistemi Implementasyonu

konularına değinilecektir.

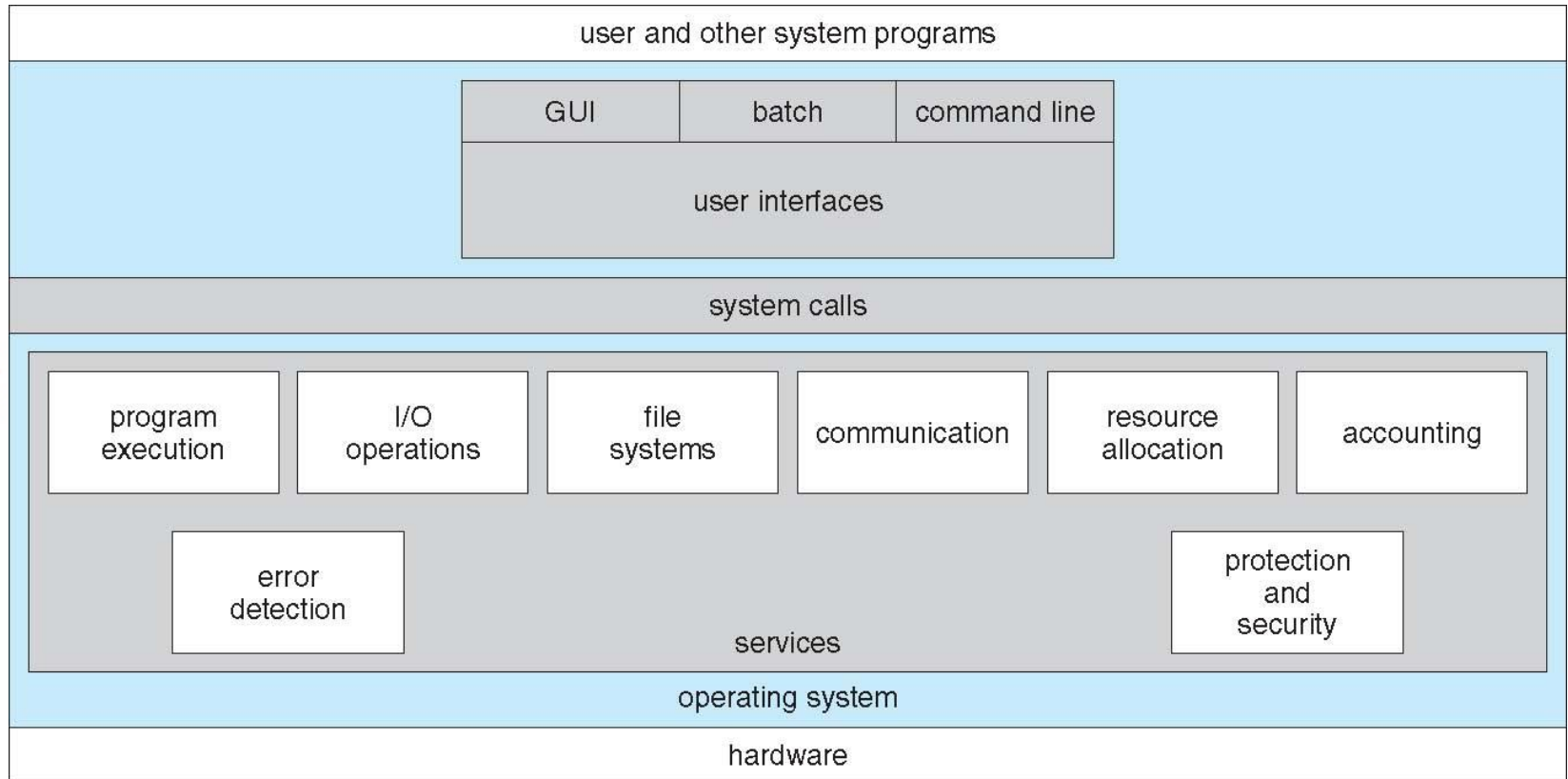
İşletim Sistemi Servisleri

- İşletim sistemi, programların **çalıştırılabilmesi** için **gerekli ortamı** (donanım ve yazılım kaynaklarını, bunların paylaşımlarını) sağlar.
- Örneğin, programlara ve bu programların kullanıcılarına, çeşitli **sistem servisleri** ile hizmet eder.
- Bazı spesifik servisler, ***işletim sistemi özelinde*** farklılık gösterse de her işletim sisteminde **ortak** olan bazı servisler *bulunmaktadır*. İşletim sistemi servislerini kullanım **amaçlarına göre** **ikiye** ayırabiliriz:
 1. **Kullanıcılara** yardımcı olanlar
 2. İşletim sisteminin **etkin çalışmasını** sağlayanlar

İşletim Sistemi Servisleri (devam...)

- Kullanıcılara yardımcı olmak için fonksiyonlar sağlayan *işletim sistemi servisleri*
 - Kullanıcı Arayüzü (UI)
 - Program Çalışması / Yürütmesi (Program Execution)
 - Giriş/Çıkış İşlemleri
 - Dosya Sistemi İşlemleri
 - Haberleşme (Communication)
 - Hata Tespiti
- İşletim sisteminin etkin çalışmasını sağlayan *işletim sistemi servisleri*
 - Kaynak Tahsisi (Resource Allocation)
 - Hesaplama / İstatistik
 - Koruma ve Güvenlik (Protection and Security)

İşletim Sistemi Servisleri (devam...)



İşletim Sistemi Servisleri (devam...)

- **Kullanıcı Arayüzü (User Interface - UI):** Tüm işletim sistemleri mutlaka bir kullanıcı ara yüzüne sahiptirler. UI çeşitli formlara sahiptir:
 - **Command-line Interface (CLI):** *Komut satırından* belirli komutlar ve parametreleri girilerek belirli işlerin yapıldığı interface'dir. Örneğin: Windows → **dir**, Unix → **ls** komutları klasörleri listeler.
 - **Batch Interface:** Birden fazla komutun bir dosyaya kaydedilerek, dosyanın tek başına tüm komutları çalıştıracak şekilde çalıştırılmasıdır (Örneğin: *Unix makefile*).
 - **Graphical User Interface (GUI):** En çok kullanılan ara yüzdür. Pencereleler, menüler ve klavye kullanarak, giriş çıkış ve diğer işlemler gerçekleştirilebilir.

İşletim Sistemi Servisleri (devam...)

- **Program Çalışması / Yürütmesi (Program Execution):** İşletim sistemi bir programı **diskten belleğe yükleyerek**, çalıştırabilmelidir. Program normal veya anormal olarak (hata alarak) bu çalışmasını **sonlandırabilmelidir**.
- **Giriş/Çıkış İşlemleri:** İşletim sistemi bir kullanıcı programının çalışma anında ihtiyacı **tüm I/O işlemleri** (disk, fare, CD, DVD) için çeşitli fonksiyonlar sunar.
- **Dosya Sistemi (File System) İşlemleri:** Dosya sistemlerinin yönetimi, işletim sisteminde önemli ve özel bir yere sahiptir. İşletim sistemi bünyesinde yürütülen programlar; **klasörler ve dosyalar** üzerinde **yaratma-silme-yazma-okuma-arama-listeleme** işlemi gerçekleştirirler (**FAT, FAT32, NTFS, EXT2**). İşletim sistemi ayrıca, yine klasör ve dosya bazlı yetkilendirme (erişime izin verir, ya da engeller – access or deny) yönetimini gerçekleştirir

İşletim Sistemi Servisleri (devam...)

- **Haberleşme (Communication):** Bir prosesin diğer prosesle bilgi alışverişi yapmasını **gerektirecek** birçok **durum** vardır. Bu prosesler aynı fiziksel bilgisayarda veya aynı ağdaki farklı bilgisayarlarda olabilirler.
 - İletişim yöntemlerinden birisi **belleğin paylaşımlı bir bölümünde yer alan “paylaşımlı bellek – shared memory”** üzerine **iki veya daha fazla prosesin yazması** ve **okuması** şeklinde gerçekleşebilir.
 - Veya bilgi paketlerinin tanımlı formatlarda, mesaj olarak OS üzerinden gönderilmesi (**message passing**) ile gerçekleşebilir.

İşletim Sistemi Servisleri (devam...)

- **Hata Tespiti (Error Detection):** İşletim sistemi sürekli olarak hataları tespit etmek ve çözmek **zorundadır**. Hatalar, **CPU'da fiziksel bellekte**, Giriş/Çıkış aygıtlarında (disk üzerindeki parity hatası, ağ bağlantı hatası, yazıcıdaki kağıt eksikliği) veya kullanıcı programlarında (aritmetik overflow, izinsiz bellek bölgesine erişmeye çalışma, aşırı CPU kullanma vb.) meydana gelebilir.
- **Kaynak Tahsisi (Resource Allocation):** Aynı anda çalışan çoklu kullanıcı ortamlarda, **tüm kaynaklar her kullanıcı için tahsis edilmelidir**. OS kaynakların yönetimini (CPU cycle, bellek, G/C aygıtları) gerçekleştirir. Örneğin, işletim sistemi CPU'yu en iyi şekilde tahsis edebilmek adına; **belleğe yüklenen birden fazla iş (job)**, **aynı anda çalışmak için hazır olduğunda**, **CPU scheduling** ile **hangisinin execute edileceğinin** belirlenmesi işlemini gerçekleştirilir.

İşletim Sistemi Servisleri (devam...)

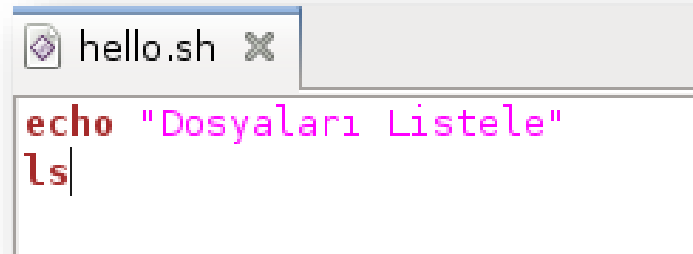
- **Hesaplama / İstatistik:** Maliyet çıkartabilmek, sistem kiralamak, sistem iyileştirmek ve istatistik alabilmek için OS kullanıcılarının hangi kaynağı ne ölçüde kullandığının mutlaka saklanması gerekmektedir.
- **Koruma ve Güvenlik (Protection and Security):** Çok kullanıcılı veya ağ üzerinde çalışan bilgisayarlarda, bilginin sahibi olan kullanıcılar bilginin kullanımının kontrol haklarını belirlemek isterler (Örneğin: kullanıcı ve/veya grup bazında erişme, görme, değiştirme, silme vb.).
 - **Protection**, tüm sistem kaynaklarına erişimin *kontrol edildiğinden* emin olunmasını sağlar.
 - **Security**, dışarıdan sisteme erişen kullanıcılar için **kimlik doğrulaması (authentication)** yapılması ve harici **G/Ç aygıtlarına yetkisiz erişimden korunma** sağlanması gibi konuları kapsar.

CLI (Command Line Interface) Arayüzü

- Metin tabanlı komutlar girilerek **çalıştırılır** ve OS servislerine ulaşılır. Bazı CLI'lar *direk kernel'da çalışırken* bazıları da **sistem programları şeklinde** hizmet verirler. OS'larda birden fazla komutun desteklenmesini sağlayan **shell (kabuk)** adı verilen CLI türleri vardır. UNIX'de sıkça kullanılan kabuklar aşağıdaki gibidir:
 - **sh (Shell, Bourne Shell)**: İlk UNIX kabuğudur ve çoğu UNIX dağıtımı ile birlikte dağıtılır.
 - **ksh (Korn Shell)**: sh uyumlu, birçok ek programlama özelliği de içeren bir kabuktur.
 - **bash (Bourne Again Shell)**: Kullanım kolaylığı bakımından en çok rağbet gören bash, sh ve ksh uyumluluğunu korurken, özellikle etkileşimli kullanıma yönelik (komut tamamlama) yenilikler de içerir.
 - **csh (C shell)**: Berkeley Üniversitesi'nde geliştirilen csh'in C diline benzer bir programlama yapısı vardır.
 - **tcsh**: csh'in biraz geliştirilmiş halidir.

CLI (Command Line Int.) Arayüzü (devam...)

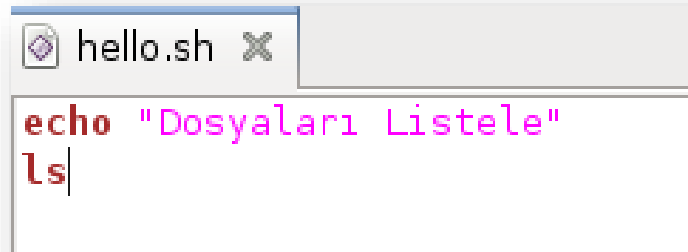
- **Shell programlama**, Linux dünyasında [popüler](#) programlama tekniklerindendir. “**sh**” uzantılı dosya direk çalıştırılır.
- **Örnek:** “hello.sh” isimli bir dosya oluşturalım ve içerisine aşağıdaki komutları yazarak herhangi bir klasöre kaydedelim.



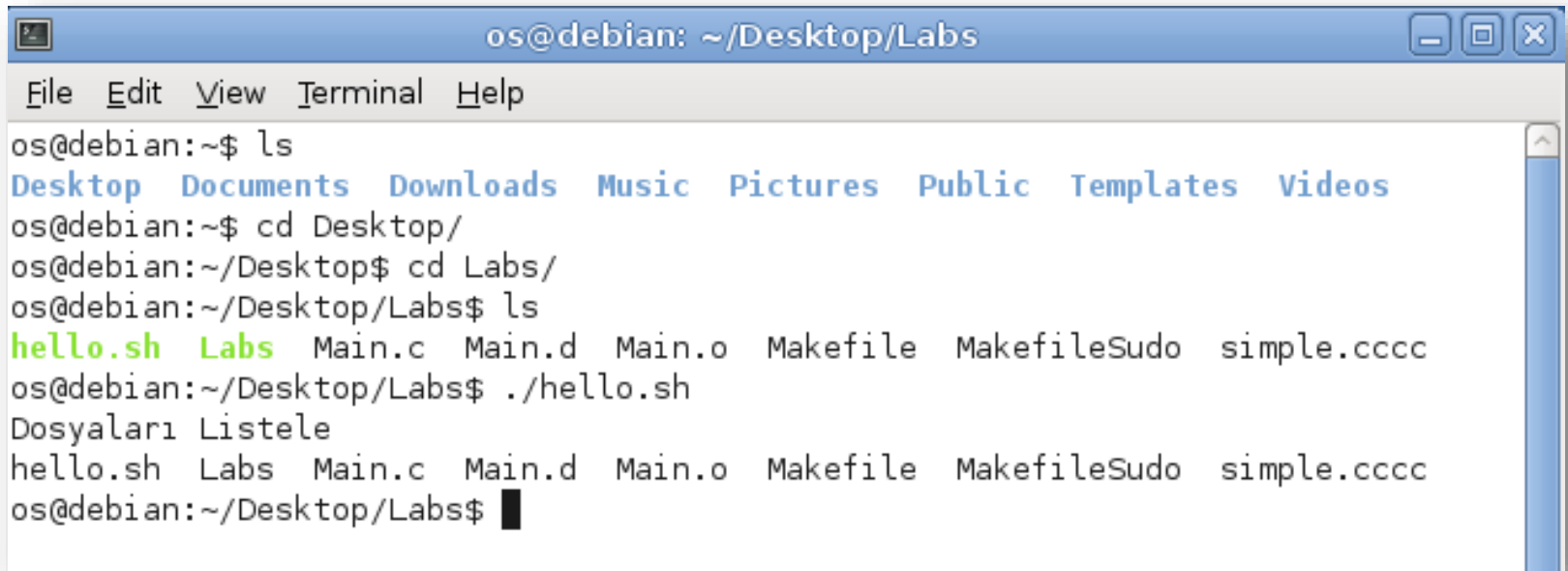
```
hello.sh x
echo "Dosyaları Listele"
ls|
```

- Daha sonra terminal aracılığıyla **bash** ekran çalıştıralım ve dosyanın olduğu klasörlere konumlanarak **shell** dosyasını çalıştıralım.

CLI (Command Line Int.) Arayüzü (devam...)



```
hello.sh x
echo "Dosyaları Listele"
ls|
```



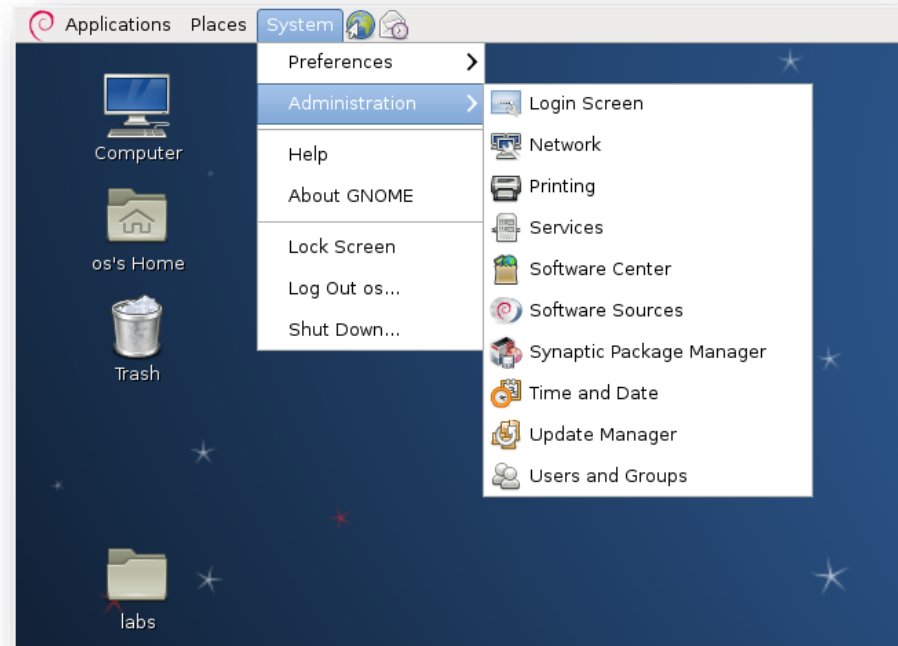
```
os@debian: ~/Desktop/Labs
File Edit View Terminal Help
os@debian:~$ ls
Desktop Documents Downloads Music Pictures Public Templates Videos
os@debian:~$ cd Desktop/
os@debian:~/Desktop$ cd Labs/
os@debian:~/Desktop/Labs$ ls
hello.sh Labs Main.c Main.d Main.o Makefile MakefileSudo simple.cccc
os@debian:~/Desktop/Labs$ ./hello.sh
Dosyaları Listele
hello.sh Labs Main.c Main.d Main.o Makefile MakefileSudo simple.cccc
os@debian:~/Desktop/Labs$
```

GUI Arayüzü

- **En çok kullanıma** sahip, **desktop metaforu** destekli OS arayüzüdür. İlk olarak Xerox PARC tarafından 1970 yıllarında başında geliştirilmiştir.
- Ancak **Apple Macintosh** bilgisayarlarla birlikte **1980'lerin başında popüler** olmuştur. Bu arayüzlerin özellikleri aşağıdaki gibidir:
 - Kullanıcı dostudur.
 - Genelde bir fare, klavye ve monitör bulunur.
 - **İkonlar**; dosyaları, programları ve çeşitli aksiyonları ifade eder.
 - Arayüzdeki nesneler üzerine fare ile gelinerek fare butonları sayesinde çeşitli aksiyonlar tetiklenir.

GUI Arayüzü (devam...)

- Birçok OS hem CLI hem de GUI ara yüzlerinin ikisini de içerir:
 - **Microsoft Windows** GUI ve CLI “command” shell
 - **Apple Mac OS X**, “Aqua” GUI ara yüzü ve UNIX kernel üzerinde shell
 - **Unix ve Linux** orijinal olarak CLI tabanlı, opsiyonel GUI ara yüzlerine sahiptirler.
 - CDE (Common Desktop Environment), KDE (K Desktop Environment), GNOME



GUI Arayüzü (devam...)

- **Dokunmatik** ekrana sahip mobil cihazlarda ise **durum daha farklıdır**:
 - Fare kullanılmaz, kullanılması uygun değildir.
 - **Aksiyonlar ve etkileşim gesture'lar** (parmakla dokunma, parmakları hareket ettirme) üzerinden gerçekleşir.
 - Metin girişi için **sanal klavye** mevcuttur.



Sistem Çağrıları (System Calls)

- Sistem çağrıları (SC), proseslerin (**çalışan programların**) OS servislerine erişmelerini sağlayan programlama ara yüzleridir.
- SC'ler, genelde C ve C++ programlama dilleri ile yazılmış **rutinlerdir**.
- **HW** tarafından *gerçekleştirilmesi gereken* SC'ler ise **assembly dili** komutları ile yazılmak **zorundadır**.
- Sistem çağrılarının nasıl kullanıldığını anlamak için **basit bir program** yazdığımızı hayal edelim ve gerçekleşen sistem çağrılarını inceleyelim.

Sistem Çağrılarını (System Calls) (devam...)

Oluşan Sistem Çağrısı Serisi

- Okunacak kaynak dosya ismi gir.
 - Okunacak dosya ismini sor (printf)
 - Okunacak dosya ismini gir (scanf)
- **Ornek Program:** Bir dosyadaki içeriği başka bir dosyaya kopyalayan programı yazınız. **Nasıl?**
- Yazılacak hedef dosya ismi gir
 - Yazılacak dosya ismini sor (printf)
 - Yazılacak dosya ismini gir (scanf)
- Kaynak dosyayı oku
 - Dosya yoksa hata mesajı göster ve çık (abort)
 - Dosya var ama erişim hakkı yoksa hata göster ve çık (abort)
 - Not: 1 SC hata mesajı göstermek, 1 SC abnormal sistemi sonlandırmak için...
- Hedef dosya oluştur
 - Dosya oluşturma hakkın yoksa hata mesajı göster ve çık (abort)
 - Aynı dosyadan varsa sil veya hata mesajı göster ve çık (abort)
 - Veya sor "Sileyim mi"?
- Döngü kur
 - Kaynak dosyadan oku.
 - EOF gelip hata verebilir
 - HW tabanlı hata olabilir (Parity Error)
 - Hedef dosyaya yaz.
 - Diskte yer kalmamış olabilir
- Kaynak ve hedef dosyaları kapat.
- İşlemin bittiğine dair mesaj yaz.
- Normal şekilde program sonlandır (Terminate normally – Final System Call)

Sistem Çağrıları (System Calls) (devam...)

- Örnekten de anlaşılacağı üzere en basit uygulamalarda bile **çok yoğun sistem çağrısı** gerçekleştirilmektedir.
- Çoğu yazılımcı *bu detayların farkında değildir*.
- Yazılımcılar genelde **API (Application Programming Interface)**'ler kullanarak programlama yaparlar ve bu *API'ler aracılığı ile* **sistem çağrıları** gerçekleştirirler.
- En çok kullanılan 3 tip API vardır.
 - **Win32 API:** Windows işletim sistemi üzerindeki uygulamalarda kullanılır.
 - **POSIX API:** POSIX tabanlı (UNIX, Linux, Mac OS X) işletim sistemlerindeki uygulamalarda kullanılır.
 - **Java API:** Java Virtual Machine (JVM) üzerinde çalışan uygulamalarda kullanılır.

Sistem Çağrıları (System Calls) (devam...)

- Yazılımcıların, kernel içerisindeki **sistem çağrılarını** **direk kullanmak yerine** API'ler aracılığı ile **sistem çağrısı** gerçekleştirmelerinin belli başlı nedenleri vardır:
 - **Taşınabilirlik:** Aynı API'yi destekleyen her işletim sistemi versiyonunda uygulamalar çalışır (**Teoride** ☺).
 - **Karmaşıklık:** Genelde API'lerin içerdikleri fonksiyonlar ve **kernel içerisindeki sistem çağrıları** birbirine çok yakın olsalar da yazılımcılar **daha az karmaşık** olduğu için *API'leri tercih ederler.*

Sistem Çağrıları (System Calls) (devam...)

Örnek: Dosyadan okuma işlemini gerçekleştirmek için kullanılan **“read”** fonksiyonu standart API dokümanı.

EXAMPLE OF STANDARD API

As an example of a standard API, consider the `read()` function that is available in UNIX and Linux systems. The API for this function is obtained from the `man` page by invoking the command

```
man read
```

on the command line. A description of this API appears below:

#include <unistd.h>		
ssize_t	read(int fd, void *buf, size_t count)	
return value	function name	parameters

A program that uses the `read()` function must include the `unistd.h` header file, as this file defines the `ssize_t` and `size_t` data types (among other things). The parameters passed to `read()` are as follows:

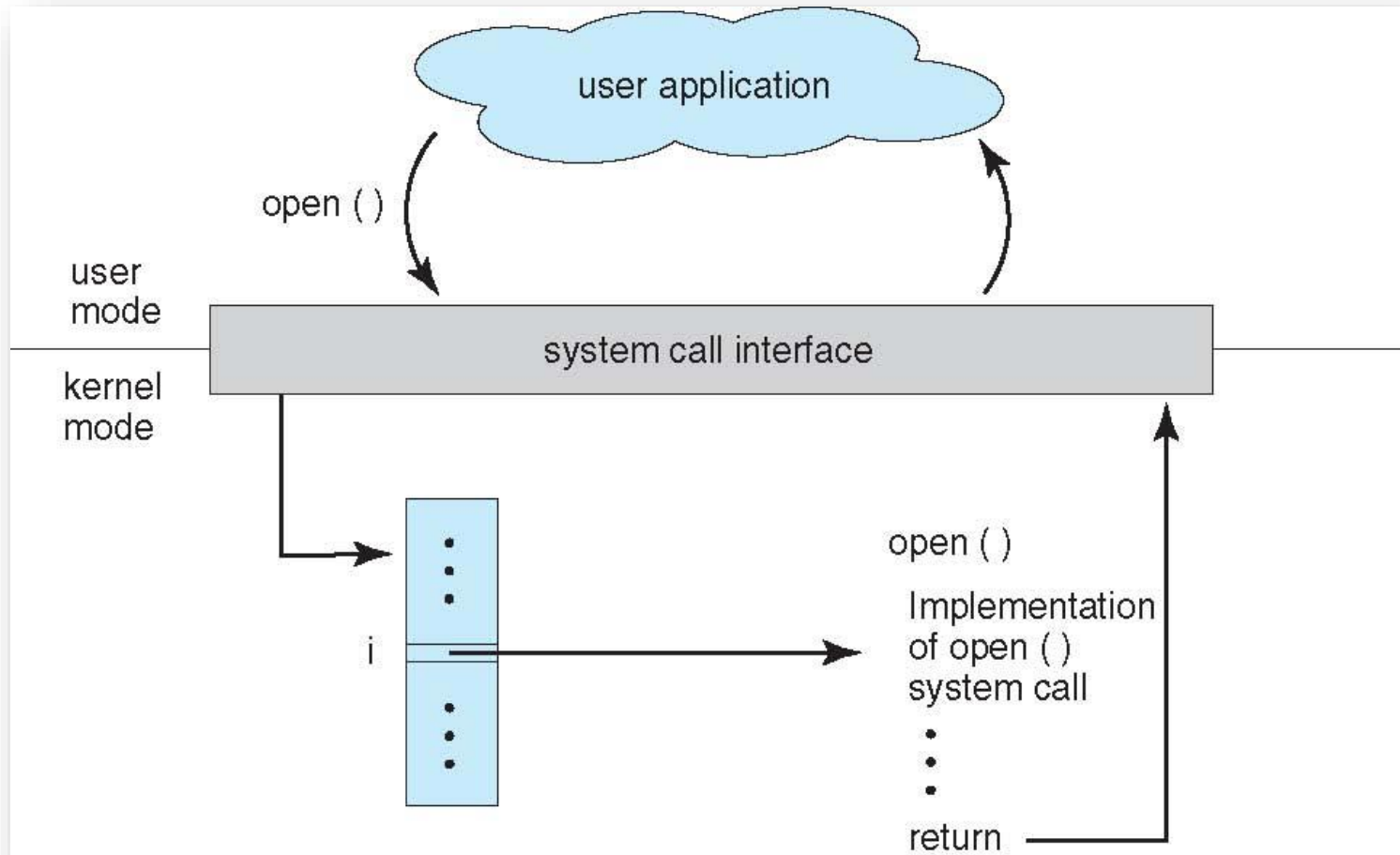
- `int fd`—the file descriptor to be read
- `void *buf`—a buffer where the data will be read into
- `size_t count`—the maximum number of bytes to be read into the buffer

On a successful read, the number of bytes read is returned. A return value of 0 indicates end of file. If an error occurs, `read()` returns `-1`.

Sistem Çağrısı Implementasyonu

- Genelde tüm OS'larda her sistem çağrısına ait bir **numara** vardır.
- **System call interface (SCI)** – Sistem çağrısı arayüzü, bu numaraları içeren **indexlenmiş bir tabloyu** *sürekli günceller*.
- SCI, API tarafından talep edilen OS kernel'daki ilgili sistem çağrısını **tetikleyerek**, sistem çağrısının **statüsünü** ve geri dönüş değerlerini **sonuç** olarak döner.
- API'yi **tetikleyen** program / yazılımcı sistem çağrısının **nasıl implemente edildiği** ile ilgili detayları bilmek zorunda değildir. Sadece API'yi ve **geri dönüş değerlerini** bilmesi yeterlidir.

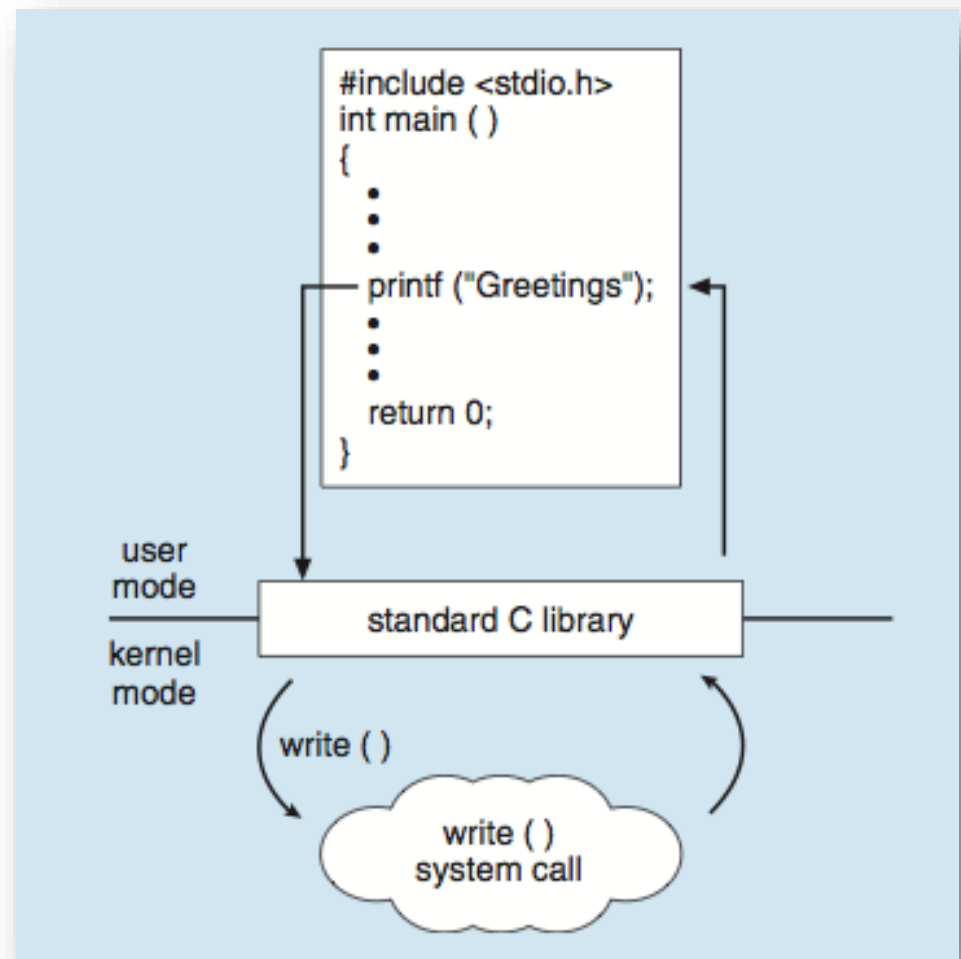
Sistem Çağrısı Implementasyonu (devam...)



Sistem Çağrısı Implementasyonu (devam...)

Örnek: `printf()`

fonksiyonunu çağıran
bir C programının
write() sistem
çağrısını tetikleme
süreci



Sistem Çağrısı Implementasyonu (devam...)

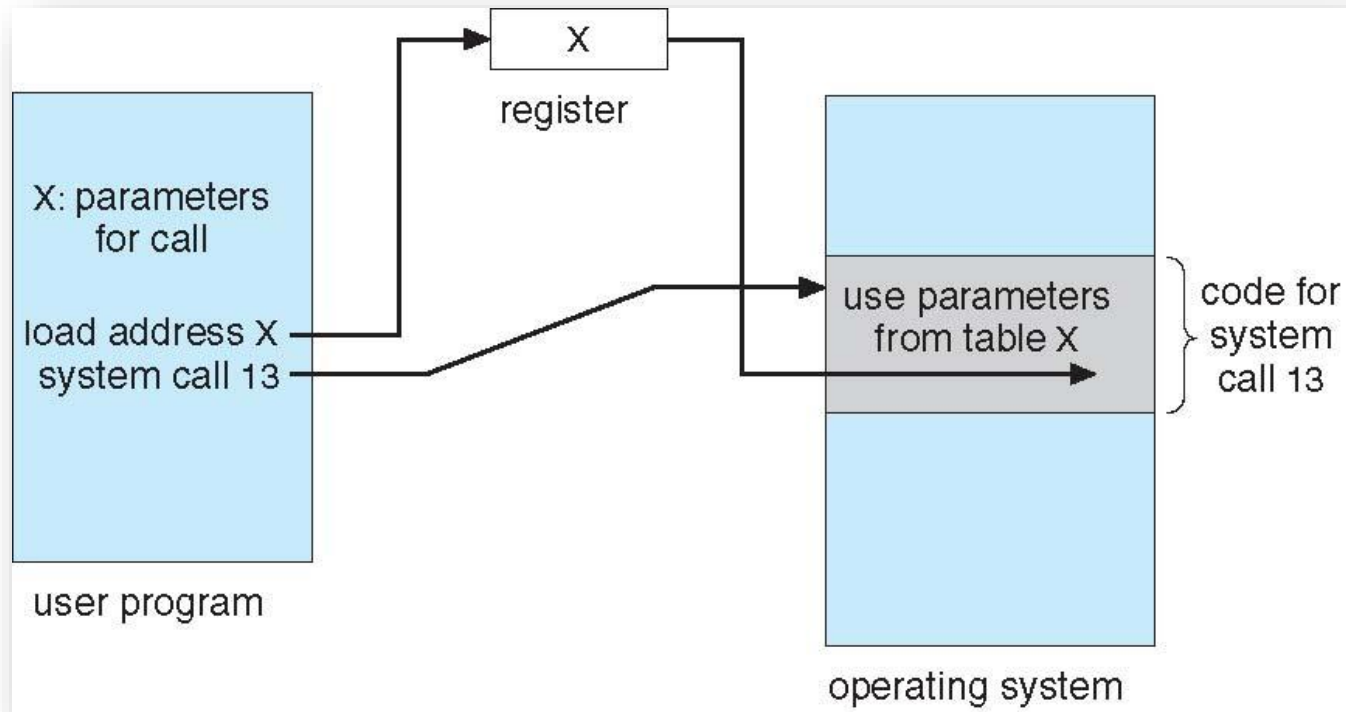
- Sistem çağrılarını **tetiklenirken** **kullanılan bilgilerin boyutu** **değişkenlik gösterebilir.**
- Örneğin bir **input** alınmak istediğinde, tetiklenen sistem çağrısına ait aşağıdaki bilgiler önemli olur:
 - Inputun nereden alınacağı bilgisi (kaynağı)
 - ❖ Dosya, klavye veya başka cihaz...
 - Alınacak inputun **boyutu** ve **yazılacağı buffer**'ın **başlangıç** hafıza adresi

Sistem Çağrısı Implementasyonu (devam...)

- Bu parametreleri OS kernal'a geçirmek için **3 yöntem** bulunmaktadır:
 - **Register kullanmak:** Parametreleri geçirmek için register kullanmak en basit yöntemdir. Ancak parametre sayısı register sayısından daha fazla olabilir.
 - **Bellekteki blok veya tabloları kullanmak:** Parametreler bellekteki bloklara veya tablolara **yazılır**. **Blok adresleri** registerlar aracılığı ile **parametre olarak** OS kernel'a geçirilir (Linux ve Solaris işletim sistemlerinde kullanılan yöntem).
 - **Stack kullanmak:** Parametreler, program tarafından **stack'a** **push** edilir ve OS tarafından **stack'den** **pop** edilir.

Sistem Çağrısı Implementasyonu (devam...)

- **Örnek:** Bellekteki tabloyu kullanarak OS kernel'a parametre geçirilmesi.



Sistem Çağrısı Türleri

- İşletim sistem tarafından desteklenmesi gereken **6 kategoride** yer alan sistem çağrısı bulunmaktadır.

1. Process control (Proses kontrol)

- end, abort
- load, execute
- create process, terminate process
- get process attributes, set process attributes
- wait for time
- wait event, signal event
- allocate and free memory
- Dump memory if error
- **Locks** for managing access to shared data between processes

Sistem Çağrısı Türleri (devam...)

2. File management (Dosya yönetimi)

- create file, delete file
- open, close file
- read, write, reposition
- get and set file attributes

3. Device management (Cihaz yönetimi)

- request device, release device
- read, write, reposition
- get device attributes, set device attributes
- logically attach or detach devices

Sistem Çağrısı Türleri (devam...)

4. Information maintenance (Sistem bilgi güncelleme)

- get time or date, set time or date
- get system data, set system data
- get and set process, file, or device attributes

5. Communications (Prosesler arası iletişim)

- create, delete communication connection
- send, receive messages (**message passing model**) to host name or process name
 - From client to server
- create and gain access to memory regions (**shared-memory model**)
- transfer status information
- attach and detach remote devices

Sistem Çağrısı Türleri (devam...)

6. Protection (Koruma)

- Control access to resources
- Get and set permissions
- Allow and deny user access

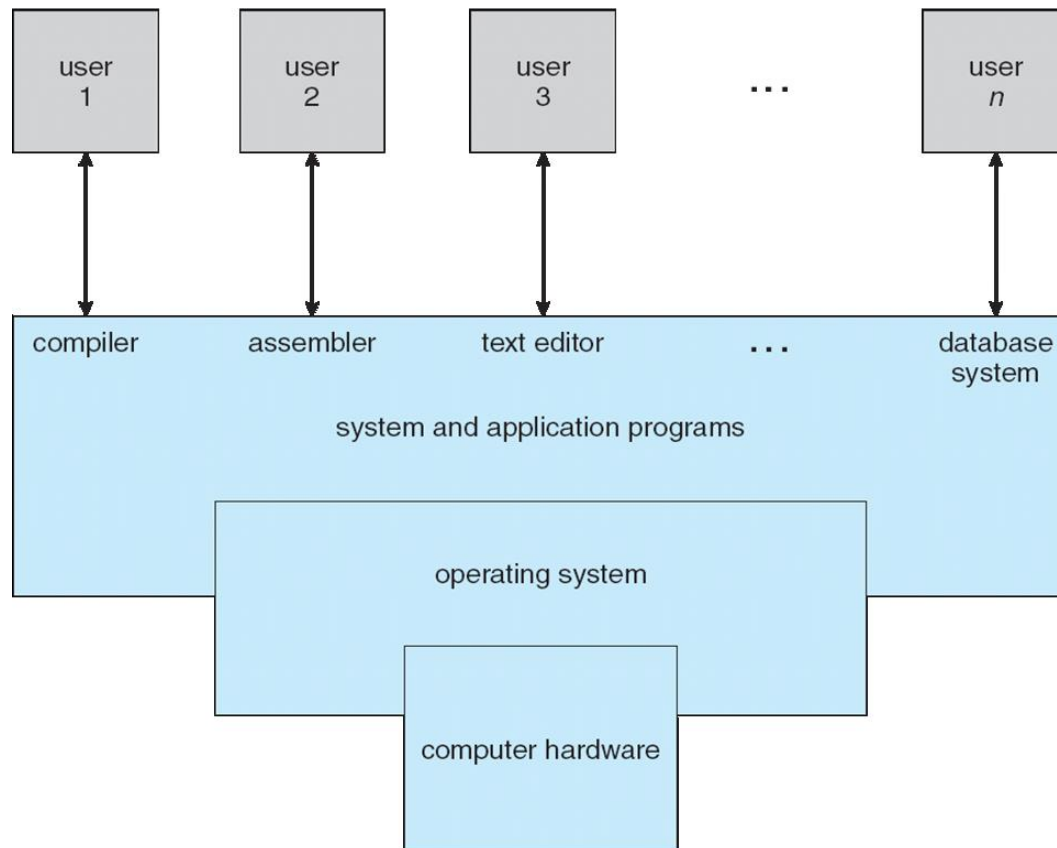


Sistem Çağrısı Türleri (devam...)

	Windows	Unix
Process Control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
File Manipulation	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
Device Manipulation	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
Information Maintenance	GetCurrentProcessID() SetTimer() Sleep()	getpid() alarm() sleep()
Communication	CreatePipe() CreateFileMapping() MapViewOfFile()	pipe() shmget() mmap()
Protection	SetFileSecurity() InitializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()

Sistem Programları

- Bölüm 1’de kısaca anlattığımız aşağıdaki şekli hatırlayalım. En altta bilgisayar **donanımı**, onun üstünde **işletim sistemi** ve OS–kullanıcılar arasında **sistem ve uygulama programları** yer almaktadır.



Sistem Programları (devam...)

- Sistem yardımcı araçları (**system utilities**) olarak da bilinen sistem programları, program geliştirilmesi ve çalıştırılması için **uygun ortamı sağlarlar**.
- Bazıları, sistem çağrılarına erişmek için kullanılan **basit *birer ara yüzken***, bazıları ***daha karmaşık*** yapıdadır.
- Sistem programları sayesinde kullanıcılar, **ortam yazılımı geliştirmek zorunda kalmazlar** (editör, derleyici).

Sistem Programları (devam...)

- **Dosya yönetimi:** Dosyaları *yaratan, silen, kopyalayan, ismini değiştiren, yazdıran* sistem programlardır.
- **Durum bilgisi (Status information):** Bazıları; Tarih, saat, bellek - disk kullanım durumu, kullanıcı sayısı, CPU kullanım oranı vb. **bilgilerin elde edilmesini** sağlayan **basit** programlardır. Bazıları ise; *Loglama, debug bilgisi, ve performans sorgulama* gibi **detaylı** bilgi veren daha karmaşık programlardır.
- **Dosya değiştirme (File modification):** Diskte veya harici depolama cihazlarındaki dosyaları oluşturmak veya bu dosyalarda değişiklik yapmak için birçok farklı metin editörü bulunmaktadır.

Sistem Programları (devam...)

- **Programlama dili desteği:** C, C++ ve Java gibi diller için sistemde **derleyiciler**, **yorumlayıcılar**, **debuggerlar**, **assemblerlar** mevcuttur.
- **Program yükleme ve çalıştırma (loading and execution):** Programlar assemble edildiğinde veya derlendiğinde, *çalıştırılmak için belleğe yüklenmelidirler.*
- **Haberleşme (Communications):** Prosesler, kullanıcılar ve bilgisayar sistemleri arasında **sanal bağlantı** yapılmasını sağlayan programlardır. Web sayfalarına erişmeyi, e-posta göndermeyi, uzak bağlantı yapmayı (**remote desktop connection**), kullanıcı bilgisayarları arası mesaj göndermeyi ve dosyaları bir bilgisayardan diğerine iletmeyi sağlarlar.

Sistem Programları (devam...)

- **Arka-plan servisleri (Background services):** Arka planda çalışan *sistem prosesleridir*.
 - Bilgisayarın **boot** edilmesi ile birlikte çalışmaya başlayan bu proseslerin kimisi **boot sonrası sonlanırlar**.
 - Kimisi de bilgisayar sistemi **kapanana kadar** çalışmaya devam ederler.
 - Windows OS’larda “**Servisler**” ve “*nix” OS’larda “**Daemons**” olarak geçerler. Daemon proseslerinin parent’ı **init()**’dir. Terminal’den erişilemezler.
 - Örneğin, her OS’da olması gereken “**network daemon (service)**”, ağ bağlantılarını dinler, bağlantı taleplerini yönetir, OS üzerindeki gerekli proseslerin çağrılmasını sağlar.
 - Diğer örnekleri: **proses planlayıcı (scheduler)**, **sistem hata izleme servisleri** ve **yazıcı sunucularıdır** (print server).

İşletim Sistemi Implementasyonu

- İşletim sistemi birçok bileşenden oluştuğu ve birçok geliştirici tarafından yazıldığı için OS **implementasyonu** ile ilgili **genellemeler** yapmak **yanlış olur**.
- İlk işletim sistemleri **assembly** dili kullanılarak yazılmıştır.
- Günümüzde bazı OS'lar *assembly ile yazılsa da*; **çoğu OS** C ve C++ gibi **üst seviye programlama dili** ile yazılmaktadır.
- Aslında, OS **birden fazla dil** ile yazılabilir:
 - Kernel'in **alt seviyeleri** assembly ile,
 - Daha **üst seviye rutinleri** C ile,
 - **Sistem programları** da C / C++ ile veya yorumlanan PERL / Python dilleri ile veya shell scriptleri ile yazılabilir.

İşletim Sistemi Implementasyonu (devam...)

- İşletim sistemlerini **assembly yerine** daha üst seviye bir dil olan **C ile yazmak** daha **avantajlıdır**, çünkü:
- Üst seviye programlama dillerinin *yazılması, anlaşılması, debug edilmesi* ve *bakımı* daha **kolaydır**.
- Üst seviye programlama dili ile yazılmış bir OS'u **farklı donanımlar** üzerinde çalıştırmak **mümkündür**.
 - Örneğin: **MD-DOS Intel 8088 assembly** dili ile yazıldığı için **sadece** *Intel X86 CPU* ailesindeki işlemciler ile uyumlu çalışmaktaydı.
 - Diğer taraftan **Linux çoğunlukla C programlama** dili ile yazıldığı için birçok CPU ile uyumlu şekilde çalışır (*Intel X86, Oracle SPARC, IBM PowerPC, ARM, Texas Instruments* vb.)

İşletim Sistemi Implementasyonu (devam...)

- Üst seviye bir dil ile yazılmış bir OS'un **olası dezavantajı** *daha yavaş çalışması* yani *performans düşüklüğü* ve daha fazla ***bilgi depolama gereksinimi*** olabilir. Ancak günümüzdeki sistemlerde bu durum majör bir problem olmaktan çıkmıştır.
 - **Örneğin:** Günümüz *Veri Yapıları ve Algoritmalarını* üst seviye dil ile kullanan OS'lar, mükemmel yazılmış assembly programlarına sahip OS'lardan dahi çok **daha performanslıdır**.
 - Ayrıca tüm işletim sistemi üst seviye bir dil ile yazıldıktan sonra, **kritik kısımlarındaki** (interrupt handler, I/O ve bellek yöneticisi) darboğazlar tespit edilerek, *assembly dili ile yeniden yazılabilir*.

İYİ ÇALIŞMALAR...

Yararlanılan Kaynaklar

- **Ders Kitabı:**
 - **Operating System Concepts**, Ninth Edition, Abraham Silberschatz, Peter Bear Galvin, Greg Gagne
- **Yardımcı Okumalar:**
 - İşletim Sistemleri, Ali Saatçi
 - Şirin Karadeniz, Ders Notları
 - İbrahim Türkoğlu, Ders Notları