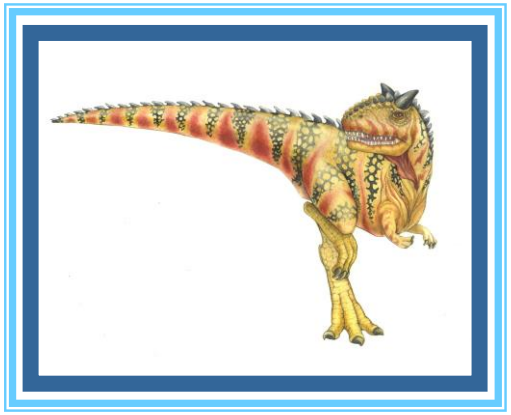
2. Bölüm: İşletim Sistemi Servisleri



Konular

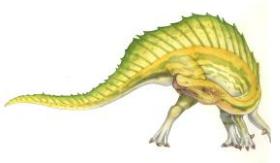
- Operating System Services
- User and Operating System-Interface
- System Calls
- System Services
- Linkers and Loaders
- Why Applications are Operating System Specific
- Design and Implementation
- Operating System Structure
- Building and Booting an Operating System
- Operating System Debugging

Amaçlar

- Bir işletim sistemi tarafından sağlanan servislerin tanımlanması
- İşletim sistemi servislerini sağlamak için sistem çağrılarının nasıl kullanıldığının gösterimi
- İşletim sistemleri tasarlamak için monolithic, layered, microkernel, modular, ve hybrid stratejilerin karşılaştırılması
- Bir işletim sisteminin booting işleminin gösterimi
- İşletim sistemi performansını izlemek için araçların kullanımı
- Bir Linux çekirdeği ile etkileşim için çekirdek modüllerinin tasarlanması ve uygulanması

İşletim Sistemi Servisleri

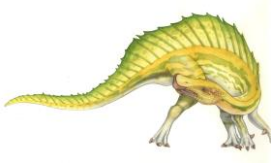
- İşletim sistemi, uygulama programlarının çalışması için ortam sağlar.
- Bir dizi **işletim sistemi servisi**, kullanıcıya yardımcı olan işlevler sağlar:
  - **User interface** - Hemen hemen tüm işletim sistemlerinin bir kullanıcı arayüzü vardır (**UI**).
    - **Command-Line Interface** (**CLI**) kullanıcı, işletim sistemini komutlarla bir arayüz üzerinden kullanır
    - **Graphics User Interface** (**GUI**) en yaygın olanıdır, I/O yu yönlendirmek, menülerden seçim yapmak için mouse kullanıldığı bir window arayüzü vardır.
    - **touch-screen, Batch**
  - **Program execution** - Sistem, bir programı belleğe yükleyebilmeli ve bu programı çalıştırabilmeli, normal veya anormal şekilde yürütmeyi sonlandırabilmelidir
  - **I/O operations** - Çalışan bir program bir dosyadan veya I/O aygıtından I/O içerebilir
  - **File-system manipulation** - Programlar dosya ve klasörler için okuma, yazma, oluşturma, silme, arama, dosya bilgilerini listeleme, izin yönetimi işlemlerine ihtiyaç duyarlar.



İşletim Sistemi Servisleri

(işletim sistemi servisleri - devam):

- **Communications** – Processler aynı bilgisayarda veya bir ağ üzerinden bilgisayarlar arasında bilgi alışverişinde bulunabilir.
  - İletişim paylaşılan hafıza veya mesaj geçişi yoluyla olabilir (işletim sistemi tarafından taşınan paketler)
- **Error detection** – İşletim sisteminin olası hatalardan sürekli olarak haberdar olması gerekir.
  - Hatalar; CPU ve bellek donanımında, I/O cihazlarında, kullanıcı programında meydana gelebilir.
  - Her bir hata türü için, işletim sistemi doğru ve tutarlı hesaplama sağlamak için uygun işlemi yapmalıdır.
  - Hata ayıklama imkanları, kullanıcının ve programcının sistemi verimli bir şekilde kullanma yeteneklerini büyük ölçüde geliştirebilir.



İşletim Sistemi Servisleri

(işletim sistemi servisleri - devam):

Kaynak paylaşımı yoluyla sistemin verimli çalışmasını sağlamak için işletim sistemi servisleri de vardır:

- **Resource allocation (**Kaynak ayırımı**) -** Birden çok kullanıcı veya birden çok iş, aynı anda çalıştığında her biri için kaynak ayrılmalıdır.
  - Birçok kaynak türü vardır- CPU cycles, main memory, file storage, I/O devices.
  - Örneğin CPU'dan maksimum düzeyde nasıl faydalanılacağı belirlenirken işletim sistemleri; CPU'nun hızını, yürütülmesi gereken işlemi, CPU'daki çekirdeği sayısını ve diğer faktörleri hesaba katan CPU-scheduling rutinine sahiptir.

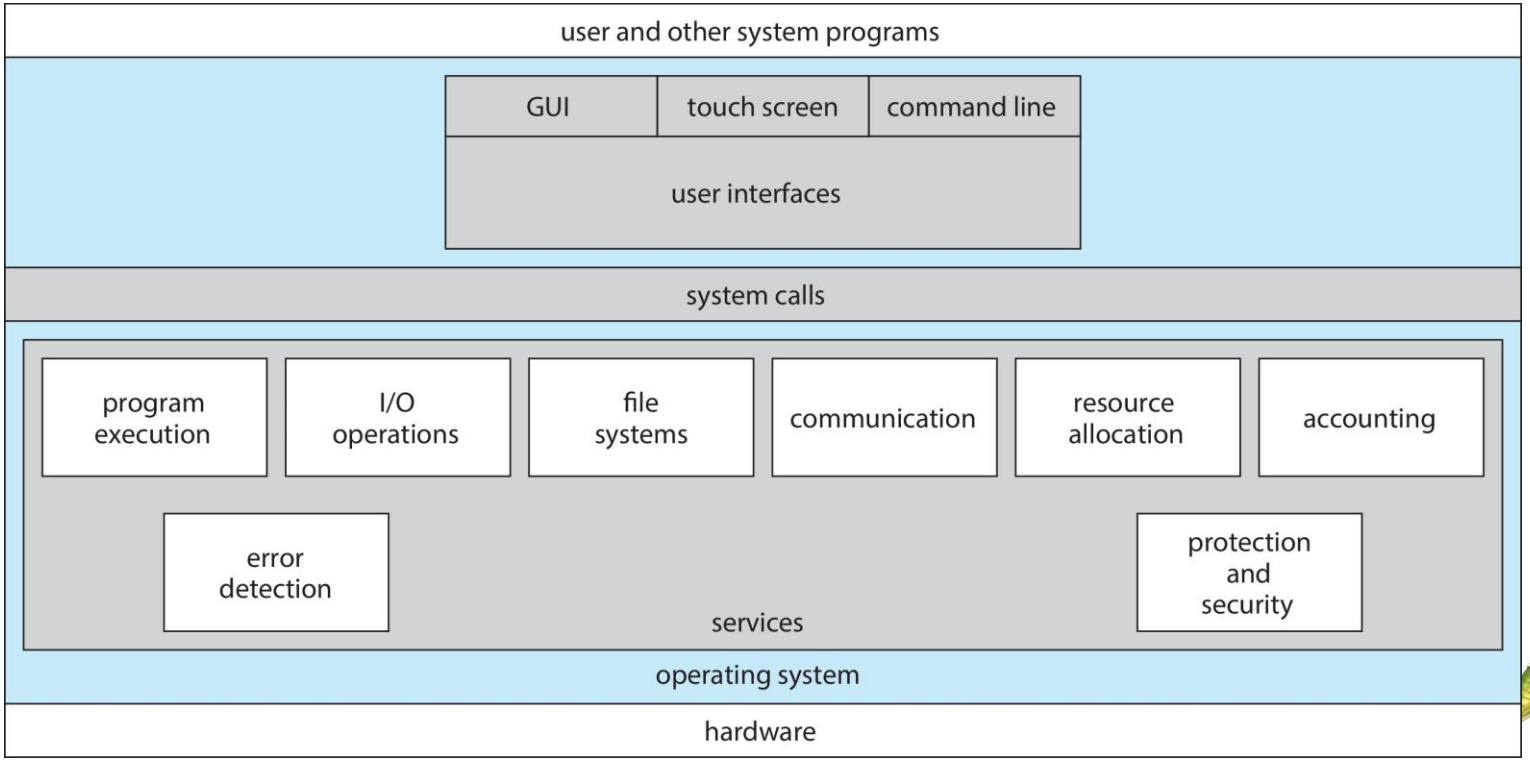
İşletim Sistemi Servisleri

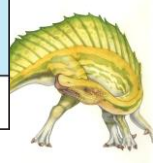
(işletim sistemi servisleri - devam):

- **Logging -** Hangi kullanıcıların ne kadar ve ne tür bilgisayar kaynaklarını kullandığını izlemek için kullanılır.
  - Kullanım istatistikleri, hesaplama servislerini iyileştirecek şekilde yapılandırmak isteyen sistem yöneticileri için değerli bir araç olabilir.
- **Protection and security** Çok kullanıcılı veya ağa bağlı bir bilgisayar sisteminde depolanan bilgilerin sahipleri bu bilgilerin kullanımını kontrol etmek isteyebilir, eşzamanlı işlemler birbirine müdahale etmemelidir.
  - **Protection** - sistem kaynaklarına erişimlerin tümünün kontrol edilmesini sağlamayı içerir.
  - **Security -** Sistemin güvenliği, dışarıdan gelenlere karşı kullanıcı kimlik doğrulamasını gerektirir, ayrıca harici I/O cihazlarına geçersiz erişimlerden korumayı da kapsar.

İşletim Sistemi Servislerinin Görünümü

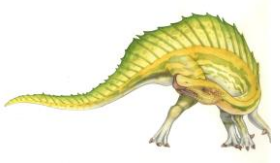
- Şekilde **çoğu işletim sisteminde yaygın olarak bulunan çeşitli işletim sistemi servislerinin** görünümü vardır.
- İşletim sistemleri arasında farklılık gösteren özel servisler de vardır. Bu servisler programcılar için de programlama işini kolaylaştırmaktadır.





Kullanıcı ve İşletim Sistemi Arayüzü
User and Operating System-Interface

- Command Interpreters (Command line Interface)
- Graphical User Interface
- Touch-Screen Interface
- Choice of Interface



Komut Satırı Yorumlayıcısı (Command Line interpreter)

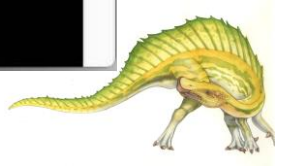
- **Command Line Interpreter** (Comman line Interface-komut satırı arayüzü, command interpreter, CLI) kullanıcıların işletim sistemi tarafından gerçekleştirilecek komutları doğrudan girmesine izin verir.
- Bazen çekirdekte, bazen sistem programı tarafından uygulanır.
- Aralarından seçim yapabileceğiniz birden çok command interpreter olan sistemlerde, interpreterler **shells** olarak bilinir.
- Örneğin, UNIX ve Linux sistemlerinde bir kullanıcı, Cshell, Bourne-Again shell, Korn Shell ve diğerler Shell ler arasından seçim yapabilir.
- CLI, öncelikle kullanıcıdan bir komut alır ve onu çalıştırır.
- Bazen yerleşik komutlar şeklinde bazen de işi yapacak programları sadece çağırma şeklinde kullanılır.
  - Yeni özellikler eklemek shell değişikliği gerektirmez.

Bourne Shell Command Interpreter (mac OS)

1. root@r6181-d5-us01:~ (ssh)

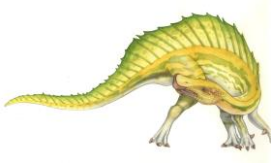
root@r6181-d5-u... ⌘1 | ssh ⌘2 | root@r6181-d5-us01... ⌘3

```
Last login: Thu Jul 14 08:47:01 on ttys002
iMacPro:~ pbg$ ssh root@r6181-d5-us01
root@r6181-d5-us01's password:
Last login: Thu Jul 14 06:01:11 2016 from 172.16.16.162
[root@r6181-d5-us01 ~]# uptime
 06:57:48 up 16 days, 10:52,  3 users,  load average: 129.52, 80.33, 56.55
[root@r6181-d5-us01 ~]# df -kh
Filesystem            Size  Used Avail Use% Mounted on
/dev/mapper/vg_ks-lv_root
                       50G   19G   28G  41% /
tmpfs                 127G  520K  127G   1% /dev/shm
/dev/sda1             477M   71M  381M  16% /boot
/dev/dssd0000         1.0T  480G  545G  47% /dssd_xfs
tcp://192.168.150.1:3334/orangefs
                       12T  5.7T  6.4T  47% /mnt/orangefs
/dev/gpfs-test         23T  1.1T   22T   5% /mnt/gpfs
[root@r6181-d5-us01 ~]#
[root@r6181-d5-us01 ~]# ps aux | sort -nrk 3,3 | head -n 5
root      97653 11.2  6.6 42665344 17520636 ?   S<Ll Jul13 166:23 /usr/lpp/mmfs/bin/mmfsd
root      69849  6.6  0.0      0     0 ?        S    Jul12 181:54 [vpthread-1-1]
root      69850  6.4  0.0      0     0 ?        S    Jul12 177:42 [vpthread-1-2]
root       3829  3.0  0.0      0     0 ?        S    Jun27 730:04 [rp_thread 7:0]
root       3826  3.0  0.0      0     0 ?        S    Jun27 728:08 [rp_thread 6:0]
[root@r6181-d5-us01 ~]# ls -l /usr/lpp/mmfs/bin/mmfsd
-r-x------ 1 root root 20667161 Jun  3  2015 /usr/lpp/mmfs/bin/mmfsd
[root@r6181-d5-us01 ~]#
```



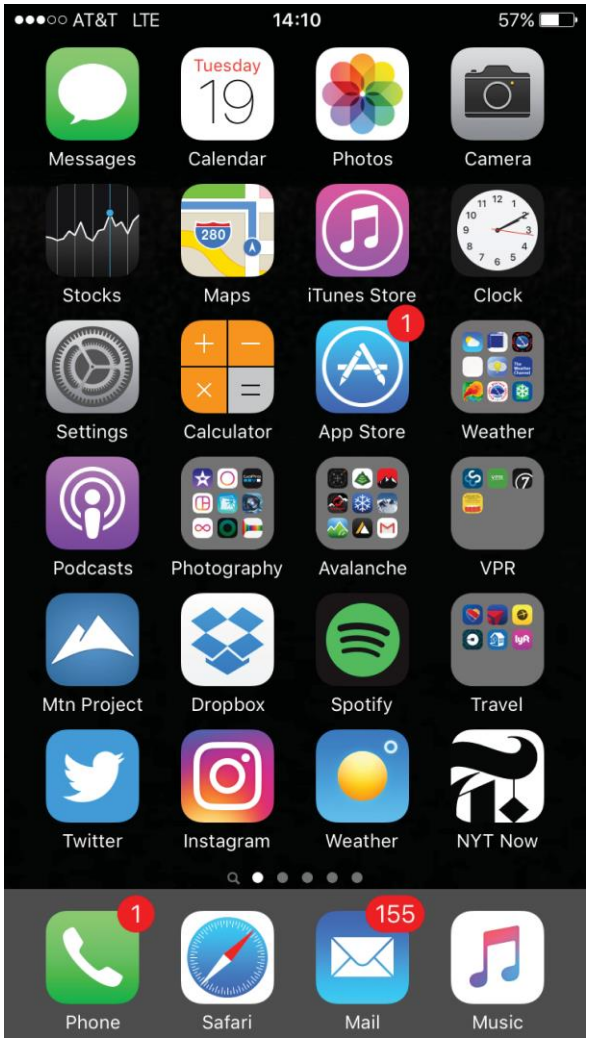
Grafiksel Kullanıcı Arayüzü (Graphical User Interface - GUI)

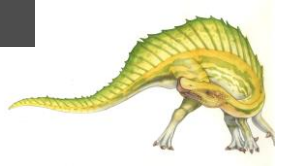
- Kullanıcı dostu **desktop** olarak karakterize edilen arayüzdür.
  - mouse, keyboard, ve monitör ana öğelerdir.
  - **Icon'lar;** dosyaları, programları, aksiyonları gösterir.
  - Bir nesnenin üzerine gelerek çeşitli mouse butonları ile çeşitli işler yapılabilir: nesnenin bilgisini gösterme, komutu yürütme, klasör açma vs.
  - Xerox PARC araştırma laboratuvarında ilk olarak geliştirilmiştir.
- Birçok sistem artık hem CLI hem de GUI içeriyor.
  - Microsoft Windows, "command" Shelle sahip GUI'dir.
  - Apple Mac OS X, altında UNIX kernel ve Shell leri bulunan "Aqua" GUI arabirimidir.
  - Unix ve Linux, isteğe bağlı GUI arabirimlerine (CDE, KDE, GNOME) ve CLI'ye sahiptir.



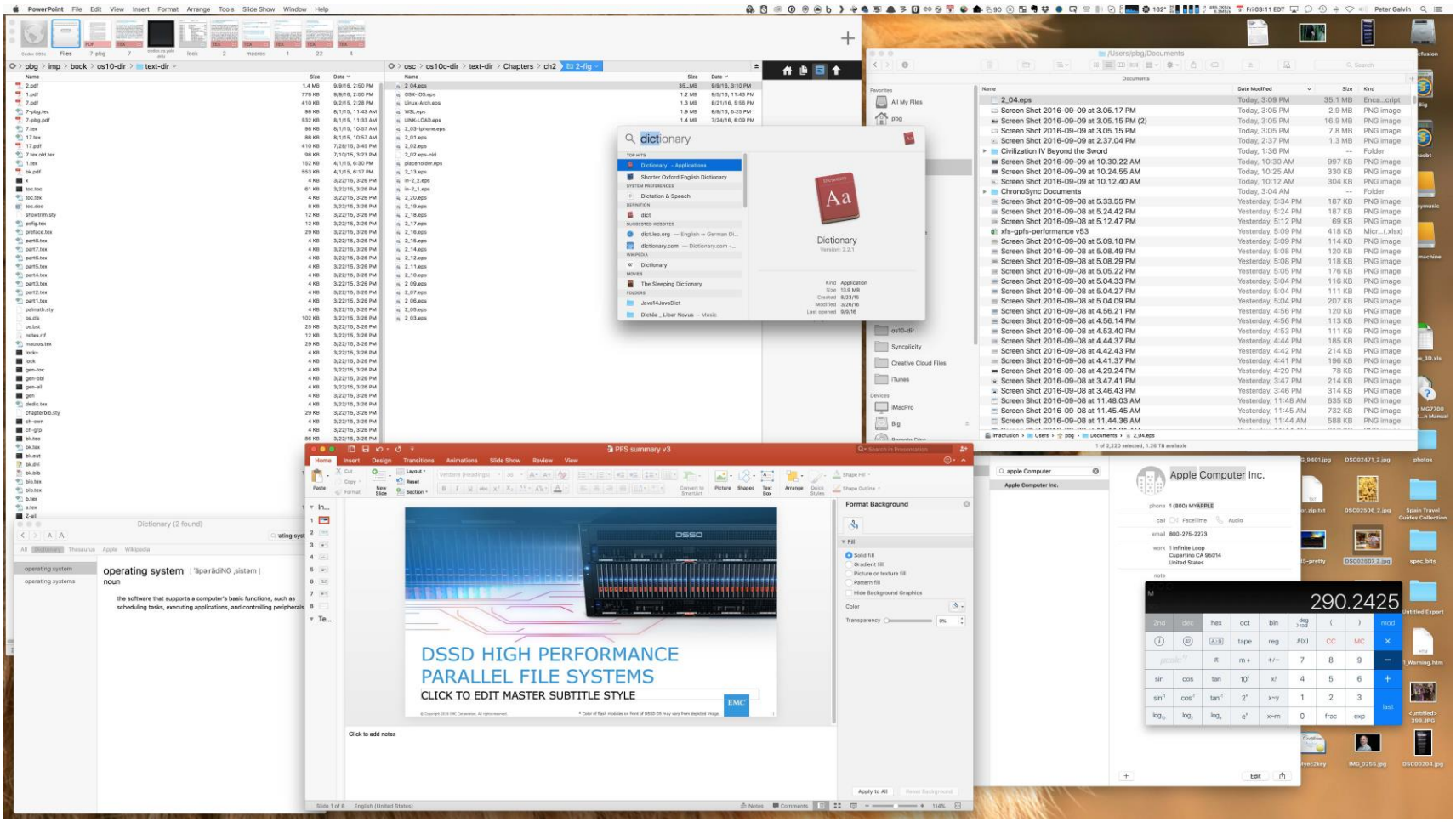
Touchscreen Interfaces

- Dokunmatik ekran cihazları yeni arayüze ihtiyaç duyarlar
  - Mouse mümkün değildir veya pratik değildir.
  - kullanıcılar, dokunmatik ekranda hareketler yaparak etkileşimde bulunurlar - örneğin, parmakla ekrana basmak, ekranda parmağı kaydırmak vb.
  - Metin girmek için sanal klavye vardır.
- Sesli komutlar vardır.



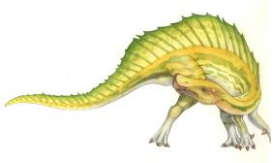


The Mac OS X GUI



Arayüz Seçimi

- CLI veya GUI arabiriminin seçimi çoğunlukla kişisel tercihlere bağlıdır. System yöneticileri ve bir sistem hakkında derin bilgiye sahip kullanıcılar genellikle CLI kullanır.
- Ayrıca, CLI ler, kısmen kendi programlanabilirliklerine sahip oldukları için, genellikle tekrar eden görevleri kolaylaştırır.
- Örneğin, sık tekrar eden bir görev bir dizi komut satırı adımı gerektiriyorsa bu adımlar bir dosyaya kaydedilebilir ve bu dosya tıpkı bir program gibi çalıştırılabilir.
  - Bu Shell scriptler, UNIX ve Linux gibi komut satırı odaklı sistemlerde çok yaygındır.
- Buna karşılık, çoğu Windows kullanıcısı GUI ortamını kullanmaktan memnundur ve neredeyse hiçbir zaman Shell interface kullanmaz.
- macOs işletim sistemi Aqua GUI ve CLI sağlar.

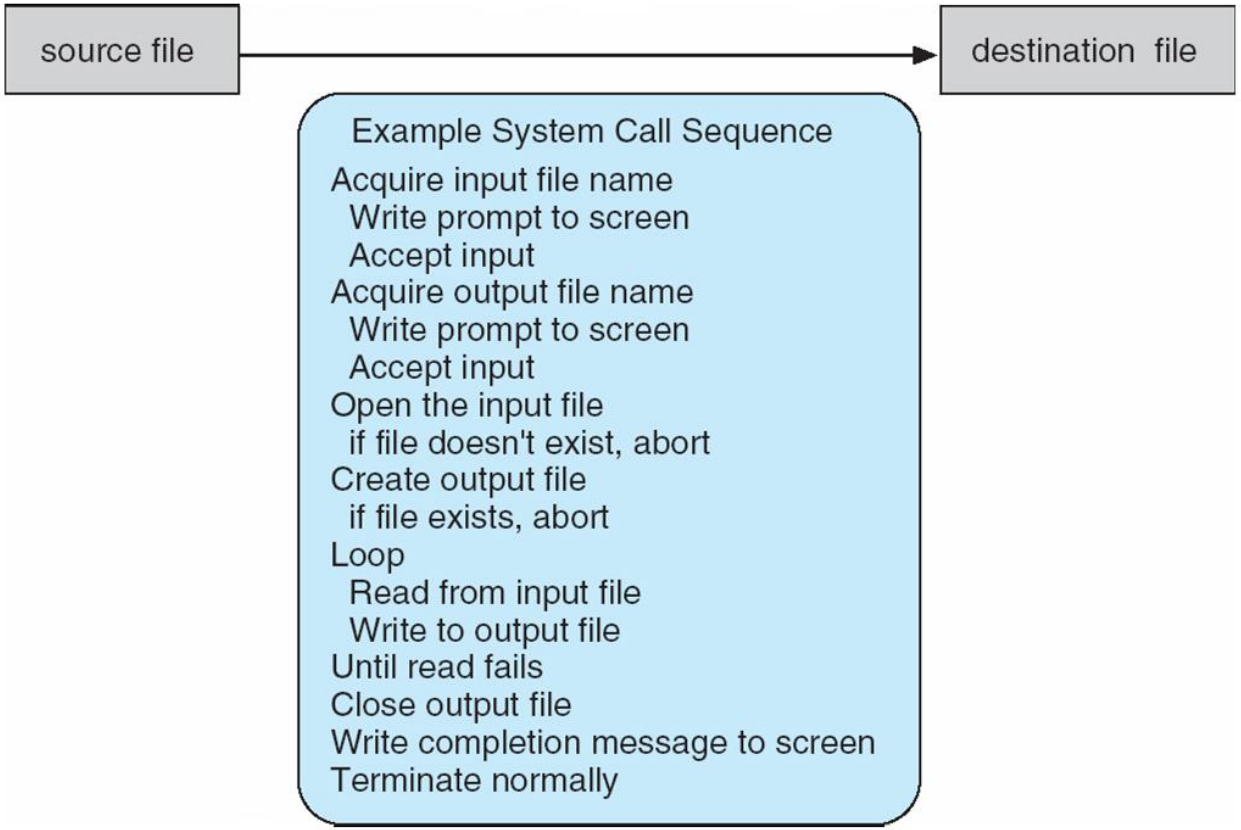


Sistem Çağrıları (System Calls)

- **Sistem çağrıları**, işletim sistemi tarafından sağlanan **sistem servislerine bir arayüz** sağlar.
- Genellikle yüksek seviye bir dilde yazılırlar. (C or C++)

Sistem Çağrı Örneği

- Basit bir dosya kopyalama işleminde bile çok sayıda sistem çağrısının kullanılır.
- Bir dosyanın içeriğini başka bir dosyaya kopyalamak için sistem çağrıları dizisi:



Sistem Çağrıları (System Calls)

- Dosya kopyalama örneğinde görüldüğü gibi, basit programlar bile işletim sistemini yoğun şekilde kullanabilir. Genellikle sistemler, saniyede binlerce sistem çağrısı yürütür. Ancak çoğu programcı asla bu düzeyde ayrıntıyı görmez.
- Uygulama geliştiriciler çoğunlukla doğrudan sistem çağrılarını kullanmak yerine yüksek seviye **Application Programming Interface** (**API**) ler kullanırlar. Örneğin, CreateProcess() Windows fonksiyonu aslında Windows kernelinde NTCreateProcess() sistem çağrısını çağırır.
- Uygulama programcılarının kullanabileceği en yaygın üç API:
  - Win32 API - Windows sistemler için,
  - POSIX API - POSIX-tabanlı sistemler için (neredeyse tüm UNIX, Linux ve Mac OS X sürümleri dahil)
  - Java API - Java virtual machine (JVM) için

Sistem Çağrıları (System Calls)

- Bir uygulama programcısı neden gerçek sistem çağrıları yerine bir API'ye göre programlamayı tercih etsin?
  - **Program taşınabilirliği** – API ile yapılan programın aynı API'yi destekleyen herhangi bir sistemde derlenmesi ve çalışması mümkün olur. (gerçekte mimari farklılıklar bunu göründüğünden daha zor hale getirse de)
  - **Kullanım kolaylığı** – gerçek sistem çağrıları, genellikle bir uygulama programcısı tarafından sunulan API'dan daha ayrıntılı ve çalışılması daha zor olabilir.

Standart bir API Örneği

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 into which the data will be read
- `size_t count`—the maximum number of bytes to be read into the buffer

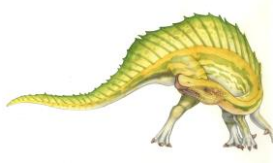
On a successful read, the number of bytes read is returned. A return value of 0 indicates end of file. If an error occurs, `read()` returns –1.

Sistem Çağrılarının Gerçekleştirimi

- Sistem çağrılarının ele alınmasındaki bir **diğer önemli faktör**, çalışma zamanı ortamıdır (**run-time environment** RTE).
- **Run-time environment** : belirli bir programlama dilinde yazılmış uygulamaları yürütmek için gereken derleyiciler veya yorumlayıcılar, kütüphaneler ve yükleyiciler (loaders) gibi diğer yazılımlar dahil olmak üzere tam yazılım paketleridir.
- RTE, işletim sistemi tarafından sağlanan sistem çağrılarına bağlantı görevi gören bir sistem çağrısı arayüzü (**System-call interface**) sağlar.
- **System-call interface**, API'deki fonksiyon çağrılarını yakalar ve işletim sistemi içinde gerekli sistem çağrılarını çağırır.

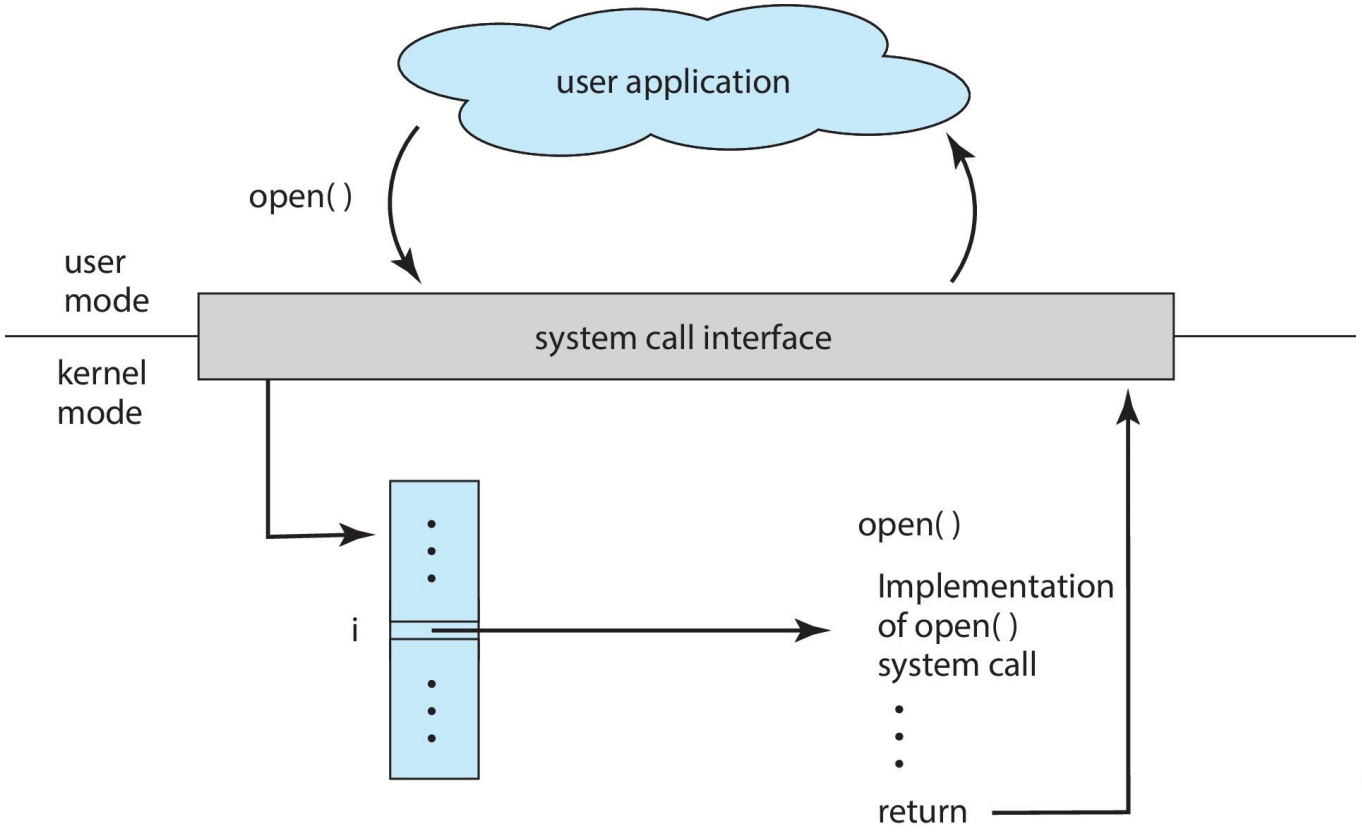
Sistem Çağrılarının Gerçekleştirimi

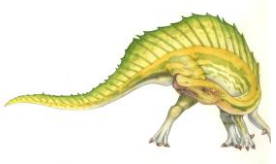
- Sistem çağrısını kullanan kişinin, sistem çağrısının nasıl uygulandığı veya nasıl yürütüldüğü hakkında hiçbir şey bilmesine gerek yoktur.
  - Sadece API'ye uyması ve bu sistem çağrısının yürütülmesi sonucunda işletim sisteminin ne yapacağını anlaması gerekir.
  - İşletim sistemi arayüzünün çoğu detayı, API tarafından programcıdan gizlenmiştir.
    - run-time support library tarafından yönetilir (derleyicide bulunan kütüphanelerde yerleşik fonksiyonlar kümesi ile)



API – Sistem Çağrı Arayüzü– OS İlişkisi

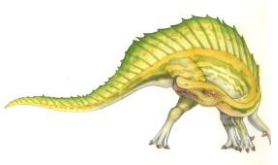
- Şekilde bir kullanıcı uygulaması open() API yi çağırıyor, sytem call interface kullanıcı tarafından gelen open() API isteği için kernel modundaki gerekli system çağrısını çağırıyor.



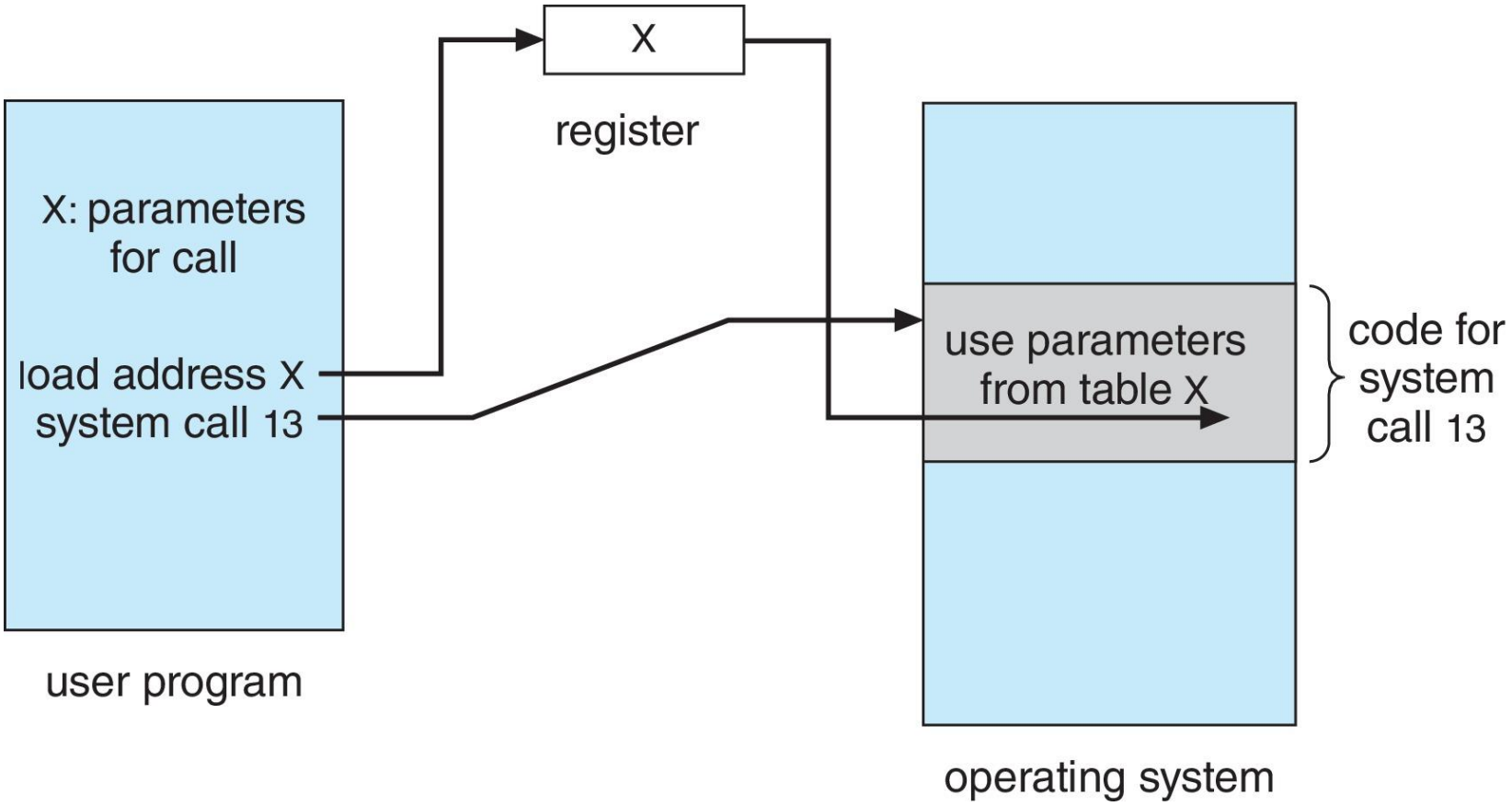


Sistem Çağrısı Parametre Geçişi

- Genellikle, çağrılacak sistem çağrısının kimliğinden daha fazla bilgi gerekir.
  - Tam bilgi türü ve miktarı işletim sistemine ve çağrıya göre değişir.
- işletim sistemine parametre geçişleri için kullanılan üç genel yöntem
  1. En basit yöntem: **registerlerde** parametre geçişi
     - Bazı durumlarda, register lerin kapasitesinden daha fazla parametre olabilir.
  2. Parametreler, hafızada bir **blokta** veya **tabloda** tutulur ve bir registere bloğun adresi parametre olarak verilir.
     - Linux ve Solaris tarafından benimsenen yaklaşımdır.
  3. Parametreler program tarafından **yığıta** eklenir ve işletim sistemi tarafından yığıttan çıkarılır
- Blok ve yığıt yöntemleri, aktarılan parametrelerin sayısını veya uzunluğunu sınırlamaz.



Tablo üzerinden Parametre Geçişi

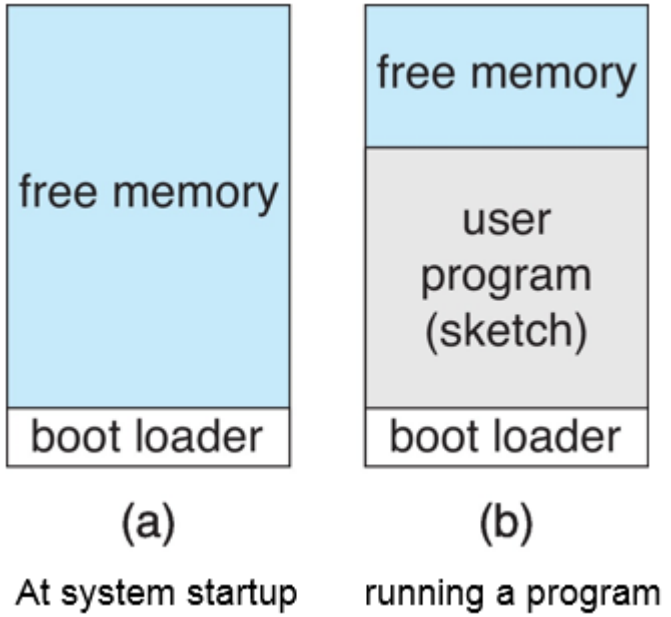


Sistem Çağrı Türleri

- Sistem çağrıları kabaca altı ana kategoriye ayrılabilir: **process control, file management, device management, information maintenance, communications, ve protection**.
- **Process control**
  - create process, terminate process
  - end, abort
    - Dump memory – hata durumunda bellek dökümü alınabilir
    - hataların tespiti ve processleri adım adım çalıştırmak için **Debugger** kullanılır
  - load, execute
  - get process attributes, set process attributes
  - wait for time
  - wait event, signal event
  - allocate and free memory
  - acquire lock and release lock - processler arasında paylaşılan verilere erişimi yönetmek için

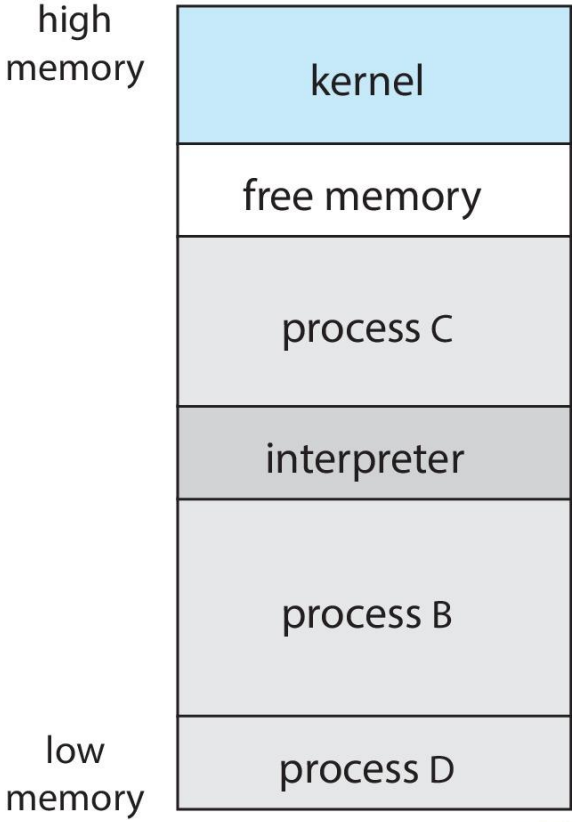
Örnek: Arduino (process control for Single-tasking)

- **Process control** un pek çok yönü ve varyasyonu vardır. Process controlun single-tasking ve multitasking sistemlerine bakacak olursak:
  - Arduino single-tasking bir sistemdir, bir mikrodenetleyicinin yanı sıra ışık, sıcaklık ve barometrik basınçtaki değişiklikler gibi çeşitli olaylara yanıt veren giriş sensörlerinden oluşan basit bir donanım platformudur.
  - Arduino'da işletim sistemi yoktur, boot loader olarak bilinen küçük bir yazılım parçası, PC'de derlenmiş programı Arduino'nun belleğindeki belirli bir bölgeye yükler.
  - Başka bir program yüklenirken var olan programın üzerine yazılır.



Örnek: FreeBSD (process control for Multitasking)

- Berkeley UNIX'ten türetilmiş multitasking bir sistemdir.
- Bir kullanıcı sistemde oturum açtığında, kullanıcının istediği shell çalıştırılır. komutları bekler ve kullanıcının istediği programları çalıştırır.
- Process oluşturmak için
  - Shell, **fork()** sistem çağrısını çalıştırır.
  - Daha sonra çalıştırılacak program **exec()** sistem çağrısı ile belleğe yüklenir ve çalıştırılır.
  - Shell processin sonlanmasını bekler ya da diğer kullanıcı komutlarını ele alır.
- Process yürütüldükten sonra **exit()** sistem çağrısı ile sonlanır, process status kodu geri döndürür:
  - code = 0 – no error
  - code > 0 – error code



Sistem Çağrı Türleri

(sistem çağrı türleri - devam):

- **File management**
  - create file, delete file
  - open, close file
  - read, write, reposition(dosyanın herhangi bir yerine atlamak için örneğin dosyanın sonuna gitmek için)
  - get and set file attributes
- **Device management**
  - request device, release device
  - read, write, reposition
  - get device attributes, set device attributes
  - logically attach or detach devices*

Sistem Çağrı Türleri

(sistem çağrı türleri - devam):

- **Information maintenance**
  - get time or date, set time or date
  - get system data, set system data
  - get and set process, file, or device attributes
- **Communications**
  - create, delete communication connection
  - Processler arası iletişimin iki yaygın modeli vardır: message passing model and the shared-memory model.
  - send, receive sistem çağrıları **message passing model**'de **host name**'ye yada **process name'ye** yapılır.
    - Yaygın oalrak da **client**'tan **server**'e yapılır.
  - **Shared-memory model** processler paylaşılan ortak bir bellek bölgesi üzerinden iletişime geçerler.
  - transfer status information
  - attach and detach remote devices

Sistem Çağrı Türleri

(sistem çağrı türleri - devam):

- **Protection**
  - Kaynaklara erişimi kontrolü
  - İzinlerde get ve set işlemleri
  - Kullanıcı erişimine izin verilmesi veya reddedilmesi

Windows ve Unix Sistem Çağrı Örnekleri

- Şekilde Windows ve UNIX işletim sistemleri için çeşitli eşdeğer sistem çağrıları gösterilmektedir.

EXAMPLES OF WINDOWS AND UNIX SYSTEM CALLS

The following illustrates various equivalent system calls for Windows and UNIX operating systems.

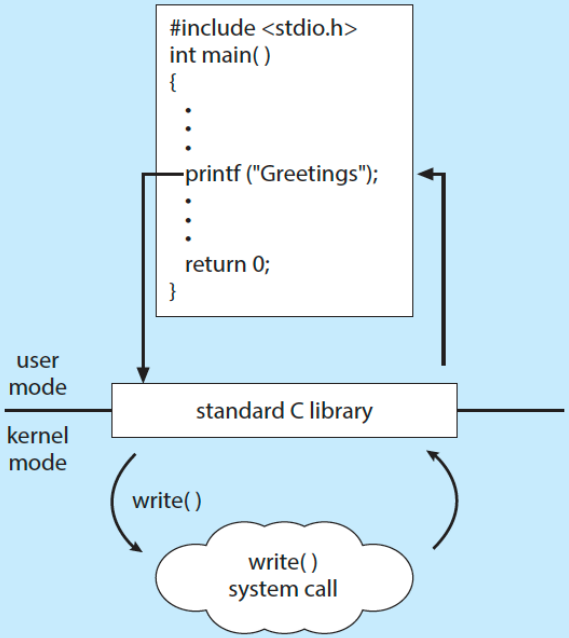
	Windows	Unix
Process control	`CreateProcess()`	`fork()`
	`ExitProcess()`	`exit()`
	`WaitForSingleObject()`	`wait()`
File management	`CreateFile()`	`open()`
	`ReadFile()`	`read()`
	`WriteFile()`	`write()`
	`CloseHandle()`	`close()`
Device management	`SetConsoleMode()`	`ioctl()`
	`ReadConsole()`	`read()`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

Standard C Library Example

- Standart C kütüphanesi, UNIX ve Linux'un birçok sürümü için sistem çağrısı arayüzünün bir bölümünü sağlar. *

THE STANDARD C LIBRARY

The standard C library provides a portion of the system-call interface for many versions of UNIX and Linux. As an example, let's assume a C program invokes the `printf()` statement. The C library intercepts this call and invokes the necessary system call (or calls) in the operating system—in this instance, the `write()` system call. The C library takes the value returned by `write()` and passes it back to the user program:



Sistem Servisleri (Sistem Yardımcı Programları - System Utilities)

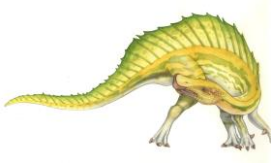
- İlk dersteki bilgisayar sisteminin bileşenlerinin soyut görünümünü hatırlayacak olursak en alt seviyede donanım, onun üstünde işletim sistemi sonra uygulama programları geliyordu. Bu hiyerarşide **Sistem Servisleri** işletim sistemi ile uygulama programları arasında bulunmaktadır, **Sistem Yardımcı Programları (system utilities)** olarak da bilinir.
- Sistem servisleri/sistem yardımcı programları, programların geliştirmesi ve yürütülmesi için uygun bir ortam sağlar. Kategorileri:
  - File manipulation
  - Status information - sometimes stored in a file
  - Programming language support
  - Program loading and execution
  - Communications
  - Background services
  - Application programs
- Çoğu kullanıcı; işletim sistemini, sistem çağrıları yerine uygulama ve sistem programları açısından görür.

Sistem Servisleri (Sistem Yardımcı Programları - System Utilities)

- Sistem servislerinin bazıları basitçe sistem çağrılarına yönelik kullanıcı arayüzleridir; bazıları ise karmaşıktır.
- **File management** - Create, delete, copy, rename, print, dump, list, dosya ve klasörlerle ilgili genel işlemler
- **Status information**
  - Bazı programlar sistemden basit bilgiler ister - date, time, kullanılabilir bellek alanı miktarı, disk boyutu, kullanıcı sayısı
  - Bazıları ise daha karmaşıktır; performance, logging, ve debugging ile ilgili bilgiler sağlar.
  - Tipik olarak, bu programlar çıktıyı formatlar daha sonra terminale veya diğer çıktı aygıtlarına veya dosyalara yazdırır veya GUI'nin bir penceresinde görüntüler.
  - Bazı sistemler, yapılandırma bilgilerini depolamak ve almak için kullanılan bir kayıt defterini (**registry)** de destekler.

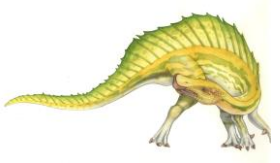
Sistem Servisleri
(Sistem Yardımcı Programları - System Utilities)

- **File modification**
  - Diskte veya diğer cihazlarda dosyaların içeriğini oluşturmak ve değiştirmek için birkaç text editor mevcut olabilir.
  - Dosyaların içeriğinde arama yapma veya metin dönüştürme işlemleri için özel komutlar da olabilir.
- **Programming-language support** - Yaygın programlama dilleri (C, C ++, Java ve Python gibi) için Compilers, assemblers, debuggers ve interpreters genellikle işletim sistemiyle birlikte sağlanır veya ayrı bir indirme olarak sunulur.
- **Program loading and execution**- Bir programın assembling veya compiling işleminden sonra yürütülmesi için belleğe yüklenmesi gerekir. Sistem; Absolute loaders, relocatable loaders, linkage editors, ve overlay-loaders içerebilir. Üst düzey diller veya makine dili için debugging sistemlerine de ihtiyaç vardır.
- **Communications** – Programlar, processler, kullanıcılar ve bilgisayar sistemleri arasında sanal bağlantılar oluşturmak için bir mekanizma sağlar.
  - Kullanıcıların birbirlerinin ekranlarına mesaj göndermelerine, web sayfalarına göz atmalarına, e-posta mesajları göndermelerine, uzaktan oturum açmalarına veya dosyaları bir makineden diğerine aktarmalarına olanak tanır.



Sistem Servisleri
(Sistem Yardımcı Programları - System Utilities)

- **Background Services**
  - Tüm genel amaçlı sistemlerin, booting sırasında belirli sistem programı processlerini başlatmak için yöntemleri vardır.
    - Bazıları sistem başlatıldığında yürütülür, görevleri tamamlandıktan sonra sonlanır.
    - Bazıları sistem kapanıncaya kadar çalışmaya devam eder. **services**, **subsystems**, **daemons (**disk checking, process scheduling, error logging, printing gibi servisler)
  - Tipik sistemlerde düzinelerce arka plan yordamı (**daemons**) bulunur. Önemli etkinlikleri kernel context yerine user contextte çalıştıran işletim sistemleri, bu etkinlikleri çalıştırmak için arka plan yordamlarını kullanabilir.



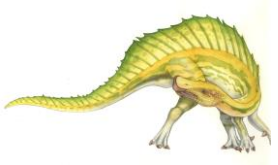
Sistem Servisleri
(Sistem Yardımcı Programları - System Utilities)

- **Application programs -** Sistem programlarının yanı sıra, çoğu işletim sistemi, genel sorunları çözmede veya genel işlemleri gerçekleştirmede yararlı olan programlarla birlikte sağlanır.
- web browsers, word processors ve text formatters, spreadsheets, database systems, compilers, plotting and statistical-analysis packages, and games

  - Sistemle ilgili değildir.
  - Kullanıcılar tarafından çalıştırılır.
  - Tipik olarak işletim sisteminin bir parçası olarak görülmez.
  - Komut satırı, mouse veya parmak hareketleri ile başlatılır.

Linkers and Loaders

- Kaynak kodlar fiziksel belleğin herhangi bir yerine yüklenip çalıştırılması için object dosyalar olarak derlenir. (**relocatable object file**)
- **Linker** ile bu relocatable object dosyalar, tek bir çalıştırılabilir binary dosyada (binary **executable** file) birleştirilir.
  - Linking aşamasında diğer object dosyalar veya kütüphaneler de dahil edilebilir.
- Bir program diskte çalıştırılabilir binary dosya olarak bulunur - örneğin, a.out veya prog.exe
- Çalıştırılması için belleğe **loader** ile getirilmelidir.
  - **Relocation,** program bölümlerine son adresleri atar ve programdaki kodu ve verileri bu adreslerle eşleşecek şekilde ayarlar.

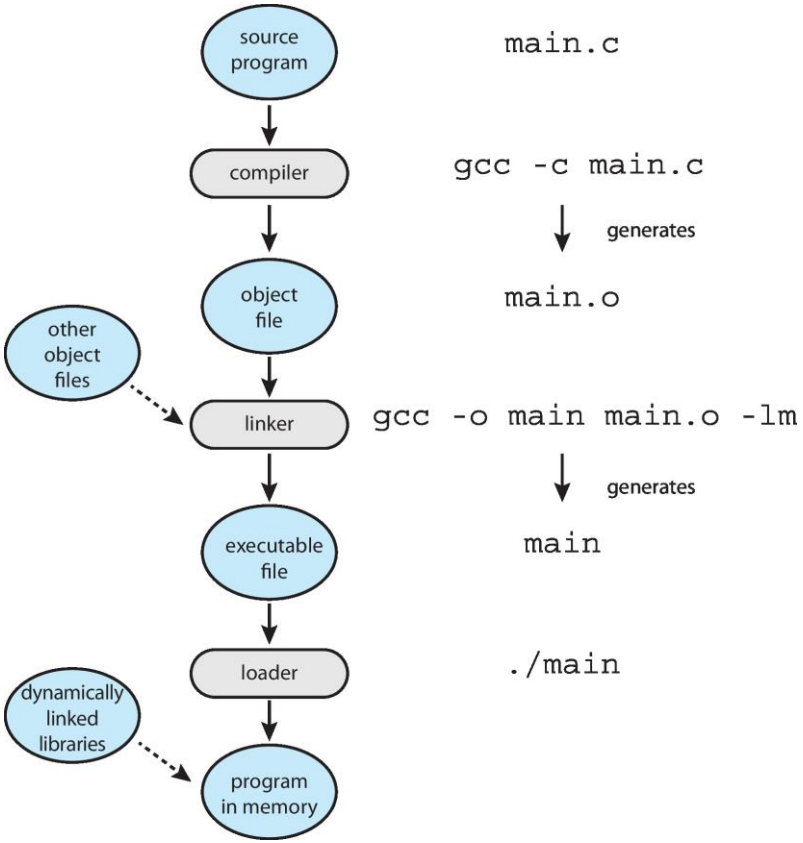


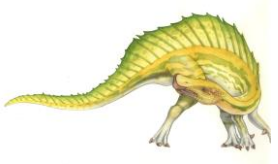
Linkers and Loaders

- Modern genel amaçlı sistemler, kütüphaneleri executable dosyalara link etmez bunu gerektiğinde dinamik olarak link eder.
  - Örneğin Windows, DLL'leri destekler **(dynamically linked libraries)**. Bu yaklaşımın faydası, yürütülebilir bir dosyada kullanılmamasına neden olabilecek kütüphaneleri link etmeyi ve yüklemeyi engellemesidir. Bunun yerine, kütüphane koşullu olarak link edilir ve program çalışma süresi sırasında gerekirse yüklenir.
- Object ve executable dosyaların standart biçimleri vardır, bu nedenle işletim sistemi bunları nasıl yükleyeceğini ve başlatacağını bilir.

The Role of the Linker and Loader

- Örneğin, bir program matematik kütüphanesini kullanıyor olsun. Matematik kütüphanesi çalıştırılabilir ana dosyaya ilk başta bağlanmaz.
- Bunun yerine linker, program yüklenirken dinamik olarak link edilmesine ve yüklenmesine izin veren relocation bilgilerini ekler.





Uygulamalar Neden İşletim Sistemine Özgüdür?

- Bir sistemde derlenen uygulamalar genellikle diğer işletim sistemlerinde çalıştırılamaz
- Her işletim sistemi kendine has sistem çağrılarını sağlar.
  - Kendi dosya formatları, vb.
- Uygulamalar birden çok işletim sistemi için çalıştırılabilir hale üç farklı şekilde getirilebilir:
  1. Uygulamalar, birden çok işletim sistemi için geçerliliği olan yorumlanmış (interpreted) bir dilde (Python veya Ruby gibi) yazılabilir.
  2. Uygulama bir sanal makinede çalıştırılabilecek şekilde yazılabilir.
  3. Uygulama için standart bir dil kullanılır (C gibi), her işletim sisteminde ayrı ayrı derlenerek her işletim sisteminde çalıştırılır.

1. Yorumlanmış Bir Dilde Uygulamanın Geliştirilmesi

- Uygulamalar, birden çok işletim sistemi için geçerliliği olan yorumlanmış bir dilde (Python veya Ruby gibi) yazılabilir.
- Yorumlayıcı, kaynak programın her satırını okur, yerel komut setine eşdeğer komutları yürütür ve yerel işletim sistemi çağrılarını çağırır.
- Performans, yerel uygulamalara göre düşer.
- Yorumlayıcılar, her işletim sisteminin özelliklerinin yalnızca bir alt kümesini sağlar ve muhtemelen uygulamaların bazı özelliklerini sınırlar.

2. Uygulamayı Sanal Makine Üzerinde Çalıştırmak

- Uygulama bir sanal makinede çalıştırılabilecek şekilde yazılabilir.
- Sanal makine, tam teşekküllü RTE dillerinin bir parçasıdır. Bu yöntemin bir örneği Java'dır.
- Java, bir loader, bytecode doğrulayıcı ve Java uygulamasını Java sanal makinesine yükleyen diğer bileşenleri içeren bir RTE'ye sahiptir.
- Java RTE, mainframe bilgisayarlardan akıllı telefonlara kadar birçok işletim sistemi için geliştirilmiştir.
- Teorik olarak herhangi bir Java uygulaması, mevcut olduğu her yerde RTE içinde çalışabilir.
- Bu yaklaşım yorumlayıcılarda olduğu gibi benzer dezavantajlara sahiptir.

3. Uygulamayı Standart bir Dilde Geliştirip Ayrı Ayrı Derlemek

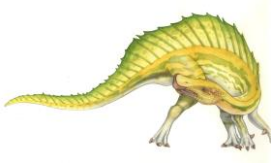
- Uygulama geliştiricisi standart bir dil veya API kullanabilir - derleyici bulunduğu makine ve işletim sistemine özgü binary dosyalar üretir.
- Bu durumda uygulama, üzerinde çalışacağı her işletim sistemine taşınmalı ve derlenmelidir.
- Bu taşıma oldukça zaman alıcı olabilir ve uygulamanın her yeni sürümü için ve devamında test ve hata ayıklama için yapılmalıdır.

Uygulamalar Neden İşletim Sistemine Özgüdür?

- Genel olarak uygulamaların taşınabilirliliğin sağlanamamasında çeşitli nedenler vardır ve bunların tümü, platformlar arası uygulamalar geliştirmeyi hala zorlu bir görev haline getirmektedir.
- Uygulama düzeyinde, işletim sistemiyle sağlanan kütüphaneler, GUI gibi özellikler sağlamak için API'ler içerir
  - Ve bir API kümesini (örneğin, Apple iPhone'da iOS'ta bulunanlar) çağırmak için tasarlanmış bir uygulama bu API'leri sağlamayan başka bir işletim sisteminde (Android gibi) çalışmayacaktır.

Application Binary Interface

- **Application Binary Interface** (**ABI**), binary kodun farklı bileşenlerinin **belirli bir işletim sistemi ve mimari** için nasıl arayüz oluşturabileceğini tanımlar.
- Bir ABI, adres genişliği, parametreleri sistem çağrılarına geçirme yöntemleri, çalışma zamanı yığınının organizasyonu, sistem kütüphanelerinin binary formatı ve veri türlerinin boyutu dahil olmak üzere düşük seviyeli ayrıntıları belirtir.
- Tipik olarak, belirli bir mimari için bir ABI belirtilir (örneğin, ARMv8 işlemci için bir ABI vardır). **Dolayısıyla, bir ABI, bir API'nin mimari düzeyindeki eşdeğeridir**.
- Bir binary çalıştırılabilir dosya derlenmiş ve belirli bir ABI'ya göre bağlanmışsa, bu ABI'yı destekleyen farklı sistemlerde çalışabilmelidir.
- Ancak, belirli bir ABI, belirli bir mimaride çalışan belirli bir işletim sistemi için tanımlandığından, ABI'ler platformlar arası uyumluluk sağlamak için çok az şey yapar.

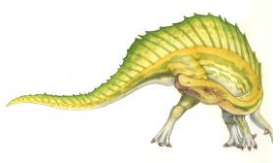


Tasarım ve Gerçekleştirim

- İşletim sisteminin tasarımında ve gerçekleştiriminde karşılaşılan sorunlar için hala tam bir çözüm ortaya konmasa da, ancak bazı yaklaşımların başarılı olduğu kanıtlanmıştır.
- Farklı İşletim Sistemlerinin iç yapısı büyük ölçüde değişebilir.
- Hedefler ve özellikler belirlenerek tasarıma başlanır.
- Donanım seçimi ve sistem türünden etkilenir.
- **Kullanıcı** hedefleri ve **Sistem** hedefleri
  - Kullanıcı hedefleri - işletim sistemi kullanımı öğrenmesi kolay, güvenilir(reliable), güvenli (safe) ve hızlı olmalıdır.
  - Sistem hedefleri - işletim sistemi, esnek, güvenilir, hatasız ve verimli olmasının yanı sıra tasarlanması, uygulanması ve bakımı kolay olmalıdır.
- Bir işletim sistemini belirlemek ve tasarlamak, yazılım mühendisliğinin son derece yaratıcı bir görevidir.

Mekanizma ve Kurallar

- **Kural (Policy):** Ne yapılması gerekiyor?
  - Örnek: Her 100 saniyede bir interrupt olsun
- **Mekanizma (Mechanism):** Nasıl yapılır?
  - Örnek: timer kullanarak
- **Önemli ilke**: Kural mekanizmadan ayrılmalıdır.
- Kuralın mekanizmadan ayrılması çok önemli bir ilkedir ve kural kararlarının daha sonra değiştirilmesi durumunda maksimum esnekliğe izin verir.
  - örnek: interrupt süresi 100 sn yerine 200 sn

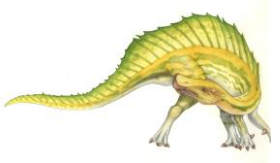


Gerçekleştirim

- Çok varyasyonu vardır.
  - İlk OS ler assembly dilinde
  - Daha sonra Algol, PL/1 gibi sistem programlama dillerinde
  - Şimdi C, C++ dillerinde gerçekleştirilmektedir.
- Aslında genellikle dillerin bir karışımıdır
  - En düşük seviyeler assembly dilinde
  - Ana gövde C dilinde
  - Sistem programları C, C++ dillerinde, PERL, Python, shell scripts gibi scripting dillerinde gerçekleştirilir.
- Daha yüksek seviyeli dil, başka donanıma daha kolay taşınabilir (**to port**)
  - Ama daha yavaştır.
- **Emulation** bir işletim sisteminin yerel olmayan donanımda çalışmasına izin verebilir.

İşletim Sistemi Yapısı

- Genel amaçlı işletim sistemleri oldukça büyük programlardır
- Bunları yapılandırmanın çeşitli yolları
  - Simple structure – MS-DOS
  - Monolithic structure– UNIX
  - Layered – an abstraction
  - Microkernel – Mach
  - Module
  - Hypbrid

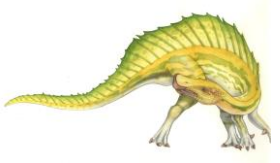


İşletim Sistemi Yapısı

- Basit Yapı (Simple structure)
  - Çoğu işletim sistemi, küçük, basit ve sınırlı bir şekilde geliştirilmeye başlar, daha sonra gelişir. MS-DOS bu şekilde bir işletim sistemidir.
  - Başlangıçta az sayıda kişi tarafından geliştirilmiş ve **modüler yapı** dikkatli bir şekilde oluşturulmamıştır.

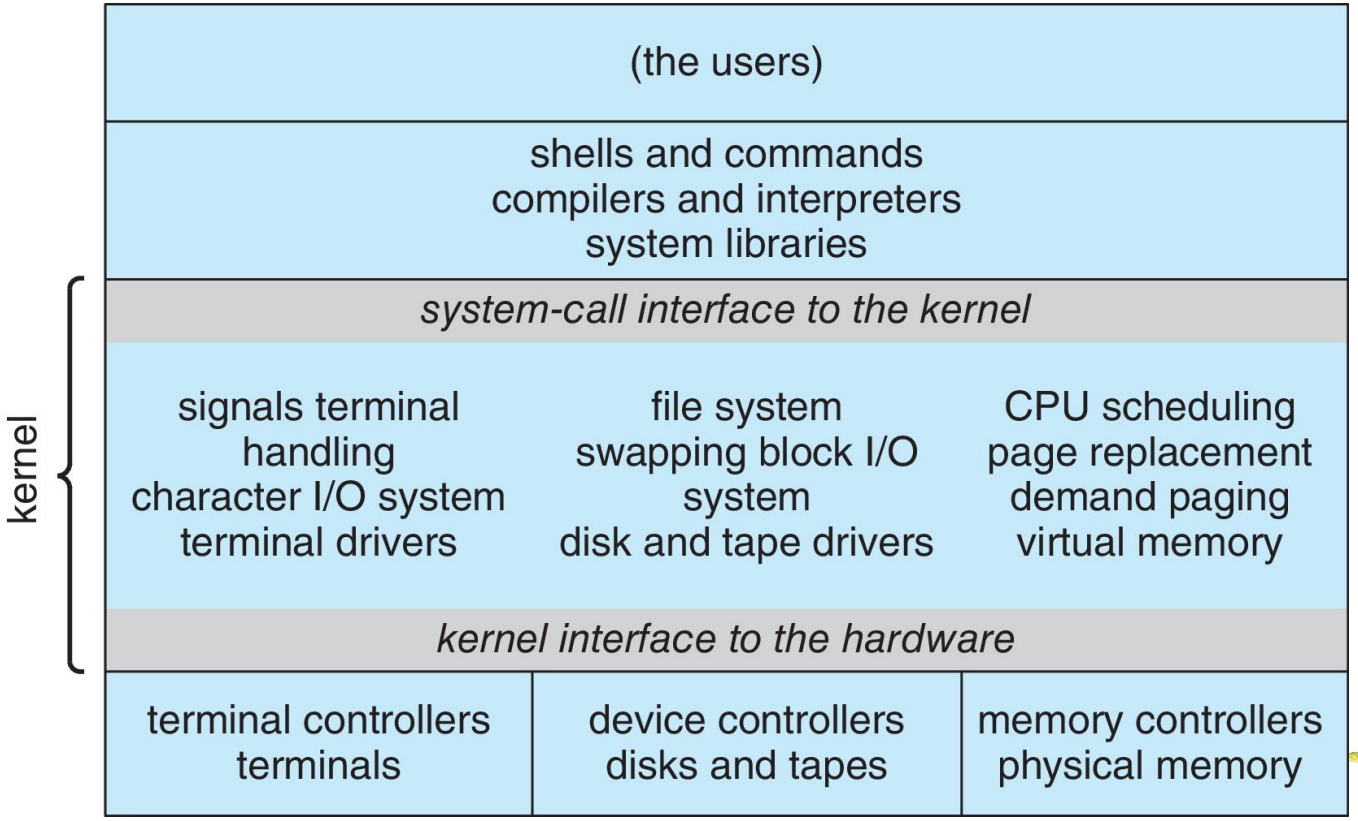
Monolithic (Tek düze) Yapı– Orjinal UNIX

- **Monolithic (Tek düze) Yapı**: işletim sistemini organize etmenin en basit yapısıdır. Kernelin tüm işlevselliği, tek bir adres uzayında çalışan tek bir statik binary dosyaya yerleştirilir.
- Bu tür sınırlı yapılanmaya bir örnek, **orjinal** UNIX işletim sistemidir.
- UNIX OS iki ayrılabilir kısımda oluşur:
  - Sistem programları
  - Kernel
    - Sistem çağrısı arayüzünün altında ve fiziksel donanımın üstünde bulunan her şeyden oluşur.
    - Çok sayıda fonksiyon bir seviyededir: file system, CPU scheduling, memory management ve diğer işletim sistemi fonksiyonları



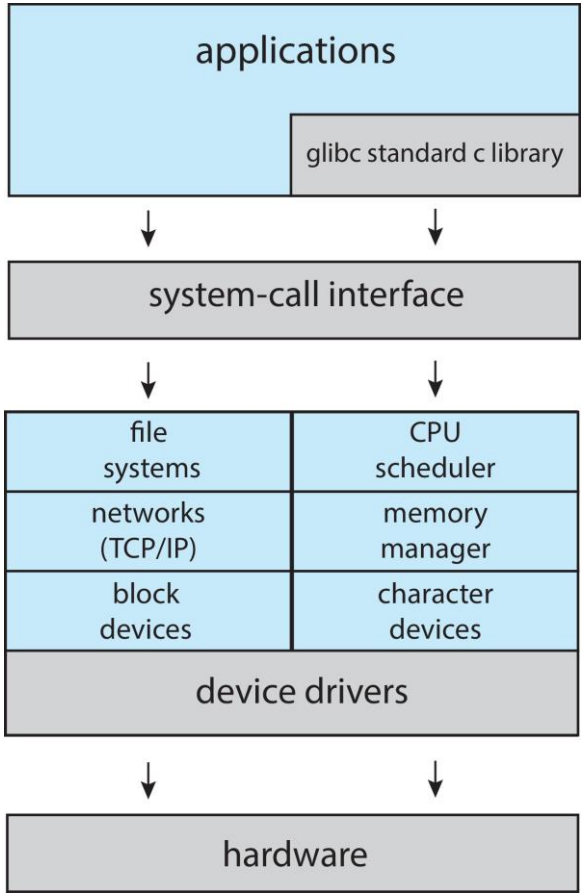
Geleneksel UNIX Sistem Yapısı

- Orijinal UNIX kernel'i geliştikçe yıllar içinde eklenen ve genişletilen bir dizi arabirim ve aygıt sürücüsüne ayrılmıştır, bir dereceye kadar katmanlı yapıdadır.
- Geleneksel UNIX işletim sistemin kernel'i, tek bir adres uzayında birleştirmek için oldukça çok sayıda fonksiyon içerir.



Linux System Structure

- Linux işletim sistemi UNIX tabanlıdır ve şekilde gösterildiği gibi benzer şekilde yapılandırılmıştır:
- Uygulamalar, çekirdeğe giden sistem çağrısı arabirimiyle iletişim kurarken genellikle **glibc** standart C kütüphanesini kullanır.
- Linux kernel, tek bir adres alanında tamamen kernel modunda çalıştığı için monolitiktir, ancak kernelin çalışma süresi boyunca değiştirilmesine izin veren modüler bir tasarıma da sahiptir.
  - Monolithic + moduler tasarım

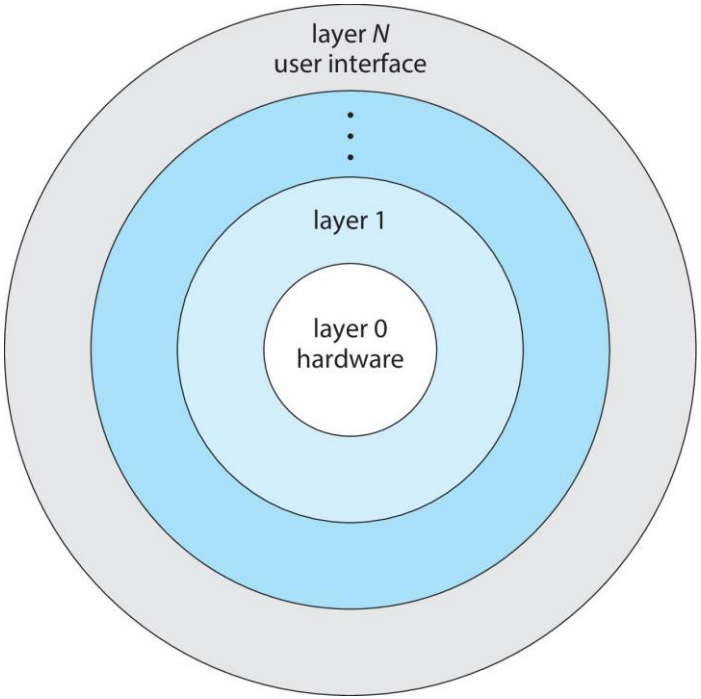


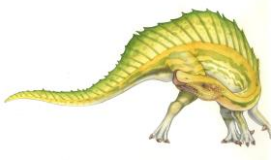
Katmanlı Yaklaşım (Layered Approach)

- Monolitik yaklaşım genellikle sıkı bağlı bir sistem (tightly coupled) olarak bilinir çünkü sistemin bir bölümünde yapılan değişikliklerin diğer kısımlar üzerinde geniş kapsamlı etkileri olabilir.
- Alternatif olarak, gevşek bağlı sistem (loosely coupled) tasarlanabilir. Böyle bir sistem, belirli ve sınırlı işlevselliğe sahip ayrı, daha küçük bileşenlere bölünmüştür.
  - Tüm bu bileşenler birlikte kerneli oluşturur.
  - Bu modüler yaklaşımın avantajı, bir bileşendeki değişikliklerin yalnızca o bileşeni etkilemesidir ve diğerlerini etkilememesidir,
  - Sistem uygulayıcılarına sistemin iç işleyişini yaratma ve değiştirme konusunda daha fazla özgürlük sağlar.

Katmanlı Yaklaşım (Layered Approach)

- Bir sistem birçok yönden modüler hale getirilebilir. Yöntemlerden biri **katmanlı yaklaşımdır**.
- İşletim sistemi, her biri alt katmanların üzerine inşa edilen birkaç katmana (düzey) bölünmüştür. En alt katman (katman 0) donanımdır; en yüksek (katman N) kullanıcı arayüzüdür.
- Modülerlik ile katmanlar, her biri yalnızca alt katmanların işlevleri ve hizmetlerini kullanacağı şekilde oluşturulur.





Katmanlı Yaklaşım(Layered Approach)

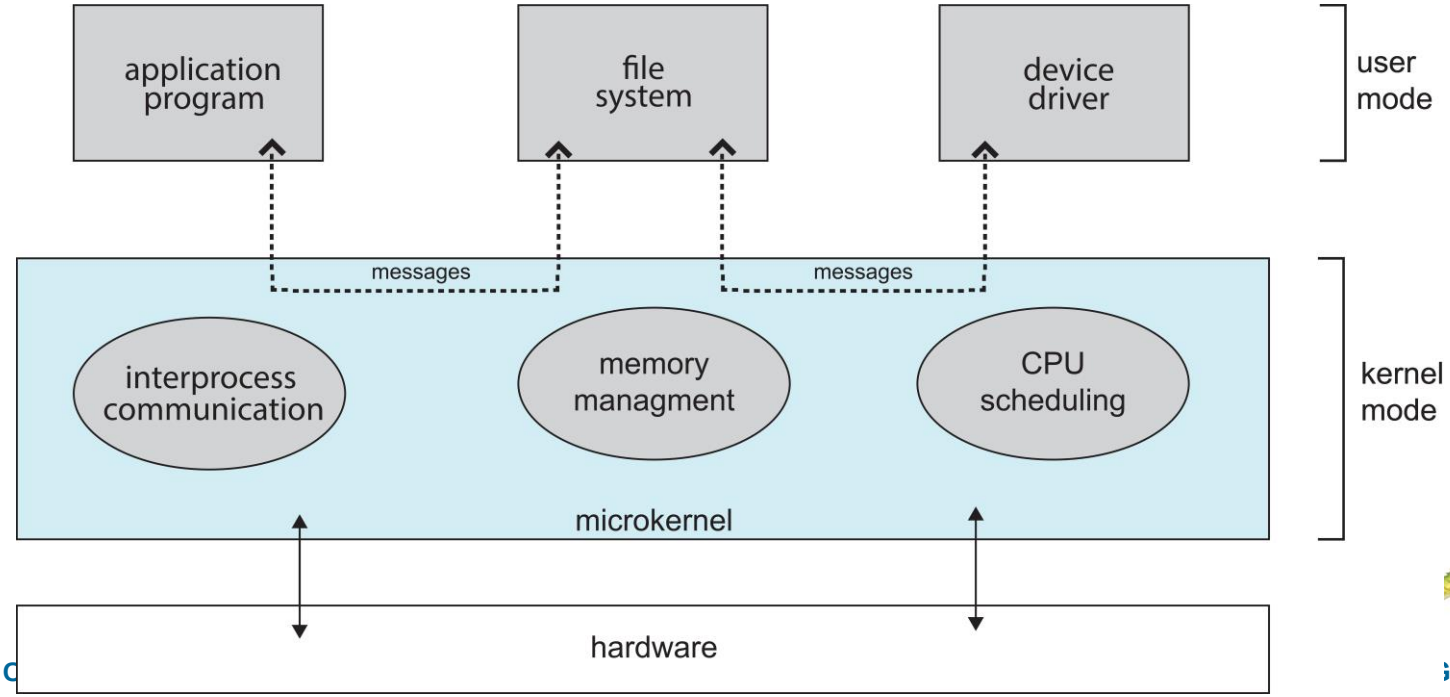
- Katmanlı yapıda, tasarım ve gerçekleştirim kolay yapılır.
- Her katman ayrı ayrı debug edilip doğrulama yapılabilir.
- Her katmanda yapılacak işlevlerin çok iyi planlanması gereklidir.
- Her katman alt katmandaki fonksiyonu çalıştıracağından dolayı performans düşme eğilimindedir.
- Katman sayısını olabildiği kadar az olacak şekilde tasarım yapmak, performans açısından gereklidir.

Microkernels

- Orijinal UNIX sisteminin monolithic bir yapıya sahip olduğunu görmüştük. UNIX genişledikçe, çekirdek büyümüş ve yönetimi zorlaşmıştır.
- Microkernel yapının avantajları:
  - Mikro çekirdeği genişletmek daha kolaydır.
  - İşletim sistemini yeni mimarilere taşımak daha kolaydır.
  - Daha güvenilirdir(reliable) ve daha güvenlidir (secure) (çoğu servis, kernel yerine user modunda çalışır)*
- Microkernel yapının dezavantajları:
  - User mode ile kernel mode arasındaki iletişim, performansı olumsuz etkileyebilir.

Microkernel Sistem Yapısı

- Mikrokernelin temel işlevi, client programı ile çeşitli servisler arasında iletişim sağlamaktır, bu servisler user spacede olsa dahi.
- Kullanıcı modülleri arasında iletişim message passing ile yapılır.
- Bu yaklaşımda, kernel içerisinde ikinci derece önem seviyesindeki tüm bileşenler sistem programlarına aktarılmıştır.
- Mikrokernel kısmında, iletişim bileşenleri, hafıza yönetimi, cpu scheduling gibi çok az bileşen kalmıştır.



Microkernels

- 1980 yılında Carnegie Mellon Üniversitesinde **microkernel** yaklaşımıyla **Mach** adında işletim sistemini geliştirilmiştir.
  - Mac OS X kernel (**Darwin**) kısmen Mach tabanlıdır.
- Windows NT'nin İlk sürümü, katmanlı bir mikrokernel organizasyonuna sahiptir. Bu sürümün performansı, Windows 95'e kıyasla düşük olmuştur.
  - Windows NT 4.0'da katmanlar kullanıcı alanından kernel alanına taşınmış ve katmanlar daha yakın bütünleştirilerek performans sorununu kısmen düzeltilmiştir.
  - Windows XP ile birlikte ağırlıklı olarak monolithic yapıya dönülmüştür.

Modül Yaklaşımı (Module Approach)

- İşletim sistemi **tasarımında** günümüzdeki **en iyi teknoloji** yüklenebilir **kernel modülleri** (**loadable kernel modules-LKMs**) kullanmaktır.
- Kernel, boot sırasında veya sistemin çalışması devam ederken yüklenen bileşenlere sahiptir.
- Modern işletim sistemlerinde, (UNIX, Linux, Mac OS X, Windows) bu tür yaklaşım yaygındır.
- CPU yönetimi ve hafıza yönetimi doğrudan kernel kısmındadır, dosya sistemleri gibi farklı destek bileşenleri yüklenebilir kernel modülleri ile sağlanmaktadır.
- Bir modül kendisi yüklendikten sonra başka modülleri çağırabilir veya iletişime geçebilir.
- Linux, cihaz sürücüleri ve dosya sistemleri için yüklenebilir kernel modüllerini kullanır.

Modül Yaklaşımı (Module Approach)

- Modül yaklaşımı
  - object-oriented yaklaşımı kullanır.
  - Her bileşen ayrıdır.
  - İletişim belli arayüzler ile sağlanır.
  - Herbiri gerektiğinde kernele yüklenebilir.
- Genel olarak katmanlı yapıya benzer ancak daha esnektir.
  - Linux, Solaris, vb.

Hibrit (Hybrid) Sistemler

- Günümüzde işletim sistemleri, tek yapıya bağlı bir şekilde geliştirilmemektedir.
  - Hibrit sistemlerde farklı yapılar birleştirilerek; performans, güvenlik ve kullanılabilirliği artırmak amaçlanmıştır.
  - Linux ve Solaris daha çok monolithic yapıdadır, ancak kernel kısmına yeni fonksiyonlar dinamik olarak eklenebilmektedir.
  - Windows çoğunlukla monolithic yapıdadır, ancak bazı kısımlar user tarafında çalıştığından kısmen de mikrokernel yapısındadır.

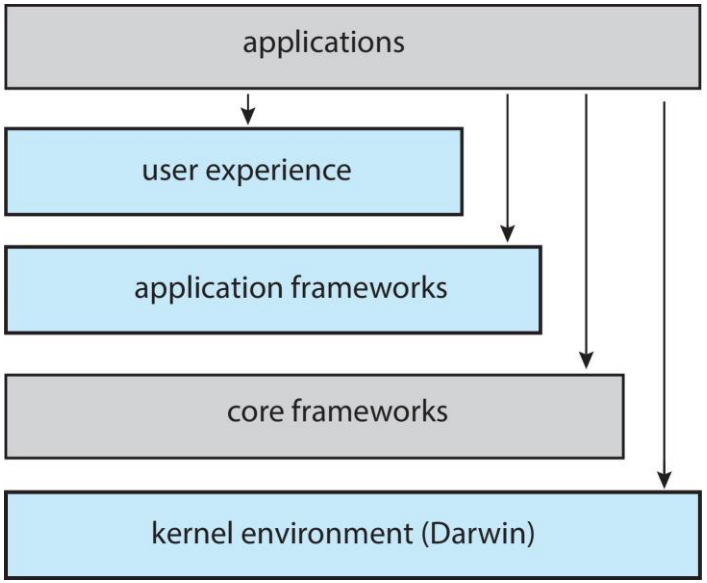
Hibrit (Hybrid) Sistemler

- Apple Mac OS X işletim sistemi hibrit yapıya sahiptir.
- En üst katmanda Aqua kullanıcı arayüzü vardır.
- İkinci katmanda, uygulama geliştirme platformları ve servisler vardır.
- Daha alt katman; kernel, Mach microkernel ve BSD Unix kernelden oluşur.
  - Ayrıca I/O kitleri ve dinamik yüklenebilen modüller (**kernel extensions**) de vardır.

macOS and iOS Structure

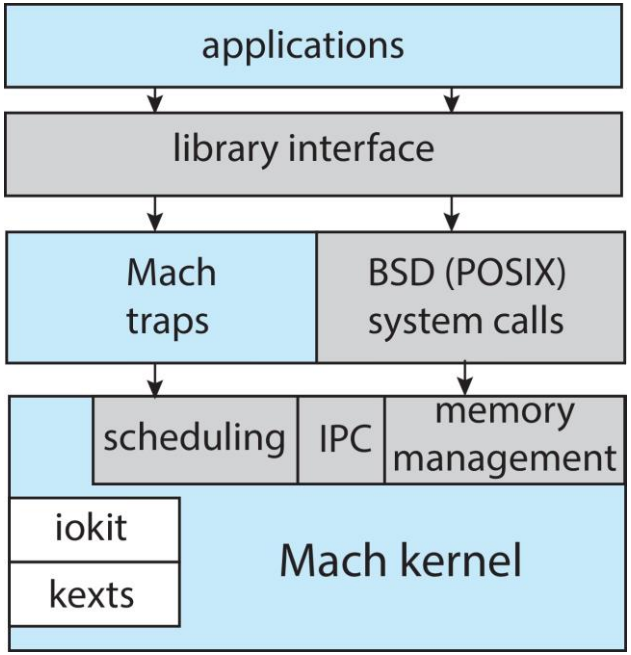
- Mimari olarak, macOS ve iOS'un pek çok ortak noktası vardır. Bu iki sistemin genel mimarisi şekilde gösterilmektedir.

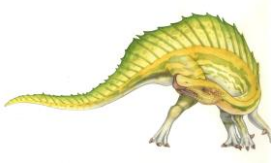
- **User experience layer** Bu katman, kullanıcıların hesaplama cihazlarıyla etkileşime girmesine izin veren yazılım arayüzünü tanımlar.
- **Application frameworks layer** Bu katman, Objective-C ve Swift programlama dilleri için bir API sağlayan Cocoa ve Cocoa Touch frameworklerini içerir.
- **Core frameworks** Bu katmanda Quicktime ve OpenGL dahil olmak üzere grafik ve medyayı destekleyen frameworkleri vardır.
- **Kernel environment** Darvin olarak bilinen bu katman Mach microkernel ve the BSD UNIX kerneli içerir.



Darwin

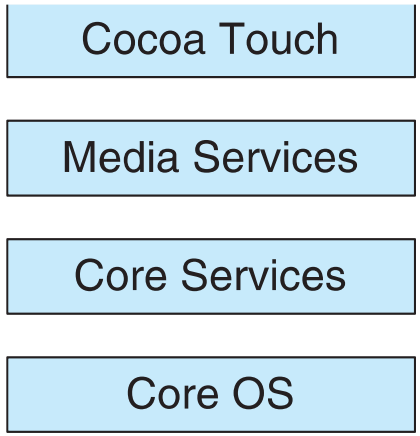
- Darwin, öncelikle Mach microkernel ve BSD UNIX kernelden oluşan katmanlı bir sistemdir.
- Çoğu işletim sistemi kernel için tek bir sistem çağrısı arayüzü sağlarken Darwin iki sistem çağrısı arayüzü sağlar: Mach sistem çağrıları ve BSD sistem çağrıları.
- Mach, memory management, CPU scheduling gibi temel işletim sistemi servislerini sağlar.
- Kernel ortamı ayrıca aygıt sürücülerinin geliştirilmesi için I/O kit ve dinamik yüklenebilir modüller için kernel extensions (kexts) içerir.





iOS

- iOS, Mac OS X üzerine yapılandırılmış, üzerine başka işlevsellikler eklenmiş bir işletim sistemidir.
- iPhone, ve iPad ürünlerinde çalışmak üzere tasarlanmıştır.
  - **Cocoa Touch**, Objective C programlama dili için API sağlar.
  - **Media services**, grafik, video ve ses servislerini sağlar.
  - **Core services**, cloud computing ve veritabanı servislerini sağlar.
  - **Core OS**, en alt katmandır ve kernel işlevlerini sağlar.

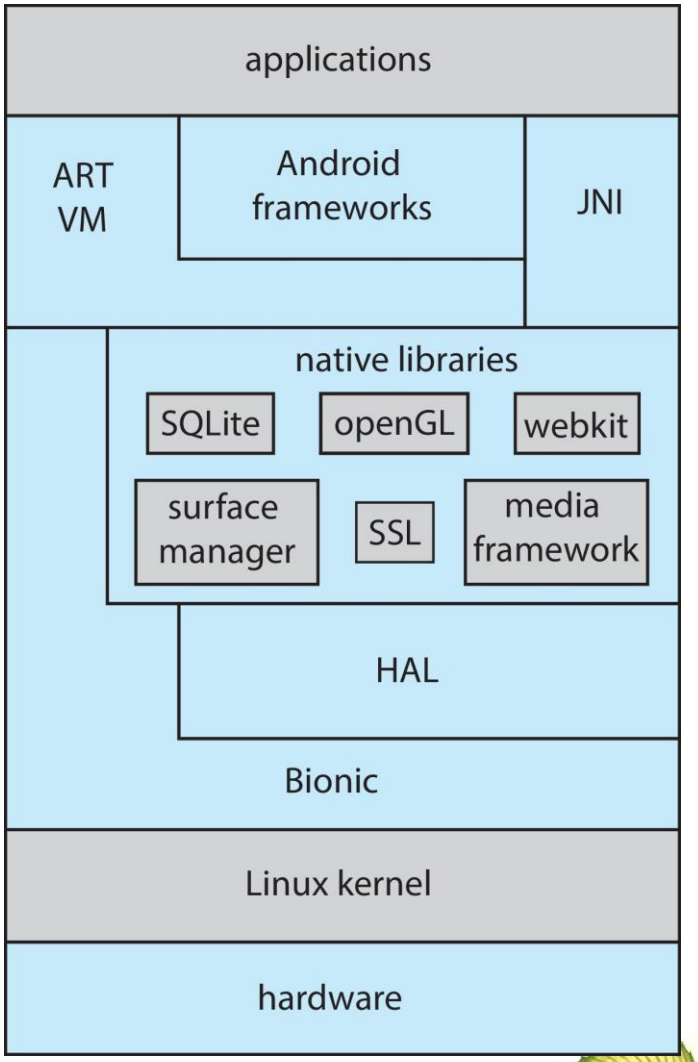


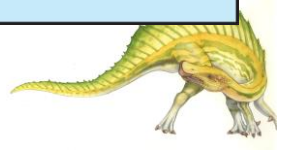
Android

- Android işletim sistemi Open Handset Alliance tarafından tasarlanmış ve Android akıllı telefonlar ve tablet bilgisayarlar için geliştirilmiştir.
  - Open Handset Alliance, bir araya gelen 84 firmanın oluşturduğu mobil cihazlar için açık standartlara dayanan bir uluslararası birliktir. (esas olarak Google liderliğindedir)
- Android işletim sistemi açık kaynaklıdır ve farklı platformlarda çalışabilir.
- Android, iOS'a benzer, katmanlı bir yazılım yığınıdır (layered stack of software), grafik, ses ve donanım özelliklerini destekleyen zengin bir framework kümesi sağlar.
- Anroid runtime, Java programlama diliyle uygulama geliştirilmesi için gerekli kütüphaneleri sağlar.
- Dalvik virtual machine, Java executable dosyalarını çalıştırır.

Android Architecture

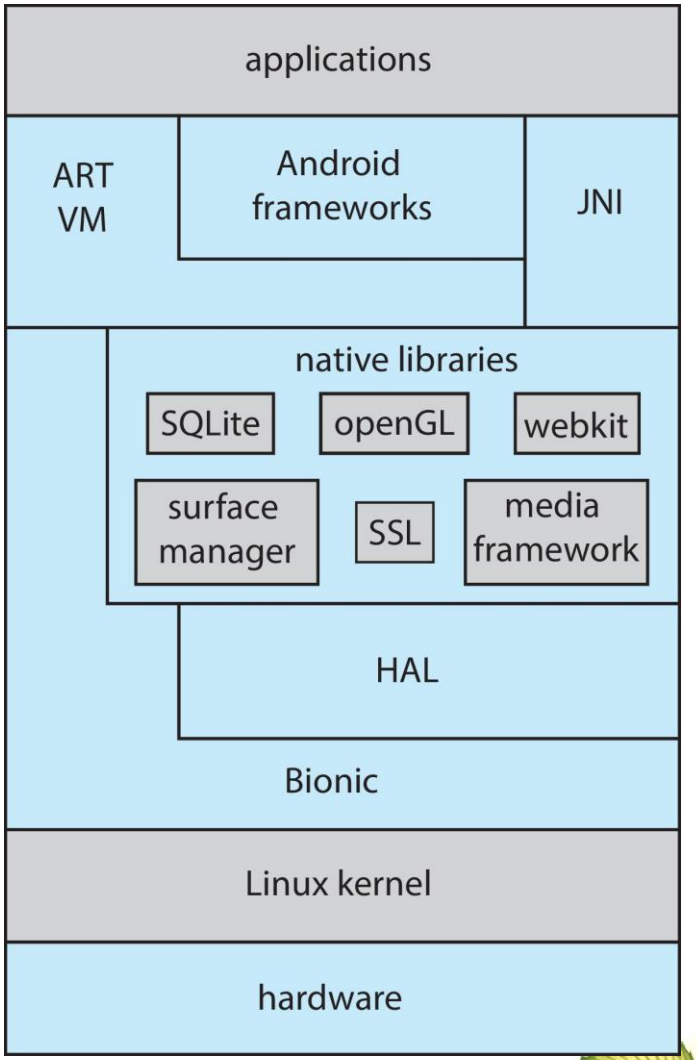
- Android yazılım yığınının altında Linux çekirdeği bulunur.
- Google, güç yönetimi gibi mobil sistemlerin özel ihtiyaçlarını desteklemek için Android'de kullanılan Linux çekirdeğini çeşitli alanlarda değiştirmiştir.
- **Linux çekirdeği** Hafıza yönetimi, process yönetimi, cihaz sürücüleri ve power management işlemleri için kullanılır.
- **Bionic-** Linux sistemleri tarafından kullanılan standart C kütüphanesi olan GNU C kütüphanesinin (glibc), Google tarafından Android için uyarlanmış halidir.
  - Bionic hem küçük bellekler hem de mobil cihazları karakterize eden daha yavaş CPU'lar için tasarlanmıştır.

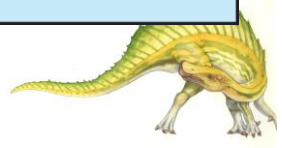




Android Architecture

- **Kütüphaneler** farklı uygulamaların geliştirilmesi için gerekli olabilecek bileşenleri bulundurur.
  - web browser için webkit
  - veritabanı işlemleri için SQLite
  - multimedia ve oyunlar için media framework ve openGL
  - standart C kütüphanesine benzeyen libc
  - ekran erişimlerinin yönetimi için surface manager
- **ART VM** - Android cihazlar için yazılım tasarımcıları, Java dilinde uygulamalar geliştirir, ancak genellikle standart Java API'sini kullanmazlar.
  - Google, Java geliştirme için ayrı bir Android API tasarlamıştır.
  - Java uygulamaları, Android için tasarlanmış ve sınırlı belleğe ve CPU işleme yeteneklerine sahip mobil cihazlar için optimize edilmiş bir sanal makine olan Android RunTime ART üzerinde yürütülebilecek bir formda derlenmiştir.



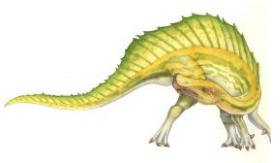


Bir İşletim Sisteminin Oluşturulması ve Önyüklenmesi (Building and Booting an Operating System)

- İşletim sistemleri genellikle çeşitli çevre birimlerine sahip bir sistem sınıfı üzerinde çalışmak üzere tasarlanmıştır.
- Genellikle, işletim sistemi satın alınan bilgisayarda zaten yüklüdür.
  - Ancak başka işletim sistemleri de kurulabilir.
  - Sıfırdan bir işletim sistemi oluşturuyorsanız aşağıdaki adımları takip etmelisiniz
    1. İşletim sistemi kaynak kodunu yazın (veya önceden yazılmış kaynak kodu edinin)
    2. Çalışacağı sistem için işletim sistemi konfigürasyonunu yapın
    3. İşletim sistemini derleyin
    4. İşletim sistemini kurun
    5. Bilgisayarı yeni işletim sistemi ile boot edin (önyükleyin).

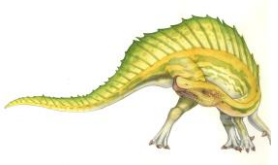
Building and Booting Linux

- Linux kaynak kodunu indirilir (http://www.kernel.org)
- "`make menuconfig`" komutu ile kernel konfigürasyonu yapılır.
  - "`make`" komutu ile kerneli derlenir. Make komutu, kerneli .config dosyasında tanımlanan yapılandırma parametrelerine göre derler ve kernel'ın imajı olan **vmlinuz** dosyasını üretir.
  - "`make modules`" komutu ile kernel modüllerini derlenir, Çekirdeğin derlenmesinde olduğu gibi, modül derlemesi .config dosyasında belirtilen yapılandırma parametrelerine bağlıdır.
  - "`make modules_install`" komutu ile kernel modülleri vmlinuz içine kurulur.
  - Kernel sistem üzerinde "`make install`" komutu ile kurulur.



System Boot

- Kernel'ın yüklenerek bilgisayarın başlatılmasına **booting** denilmektedir.
- Sisteme güç verildiğinde, sistem belirli bir bellek konumundan yürütülmeye başlanır.
- Donanımın başlatabilmesi için işletim sistemi donanıma sunulmalıdır.
- **bootstrap loader** ya da **bootstrap program**, **ROM** ya da **EEPROM** üzerindedir ve işletim sisteminin kernel kısmını belleğe yükler ve kerneli başlatır.
- bootstrap programı kerneli yüklemenin yanı sıra, belleği ve CPU'yu incelemek ve aygıtları keşfetmek gibi işler de yapar. Makine başlatmaya uygunsa program booting adımlarına devam edebilir.
- Ayrıca Bootstrap, CPU registerlerinden aygıt denetleyicilerine ve ana belleğin içeriğine kadar sistemin tüm yönlerini başlatabilir.
- Son adımda işletim sistemini başlatır ve root dosya sistemini bağlar. Bu noktadan itibaren sistemin çalıştığı söylenir.



System Boot

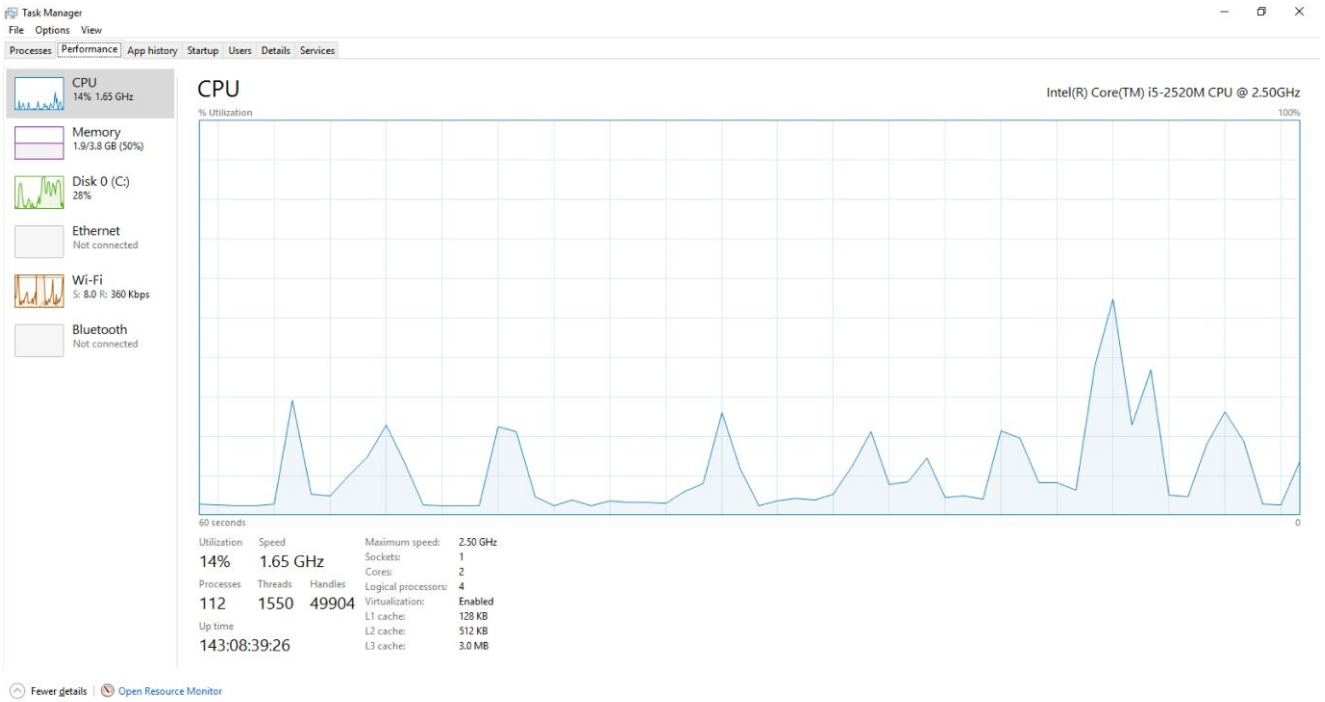
- Bazı bilgisayar sistemlerinde çok aşamalı bir **booting** işlemi kullanılır: Bilgisayar ilk kez çalıştırıldığında, **BIOS** olarak bilinen küçük bir önyükleyici (ROM üzerinde) çalıştırılır. BIOS ,işletim sistemini yükleyecek asıl kısım olan **boot block** u (sabit disk üzerinde) yükler.
- Birçok yeni bilgisayar sistemi, BIOS tabanlı önyükleme sürecini **Unified Extensible Firmware Interface** (**UEFI**) ile değiştirmiştir.
- UEFI, 64 bit sistemler ve daha büyük diskler için BIOS'a göre daha iyi destek sağlar. En önemli avantajı UEFI'nin **tek ve eksiksiz bir booting** yöneticisi olması sebebiyle çok aşamalı BIOS booting sürecinden **daha hızlı** olmasıdır.
- Çoğu işletim sistemi için boot loader, donanım sorunlarını tanılamak, bozuk dosya sistemlerini onarmak ve hatta işletim sistemini yeniden yüklemek için **recovery moduna** ya da **single user moduna** booting yapmayı sağlar.
- **GRUB**, Linux ve UNIX sistemleri için açık kaynaklı yaygın **bootstrap** programdır. Birden çok diskten, sürümden, çekirdek seçeneklerinden çekirdek seçimine izin verir.

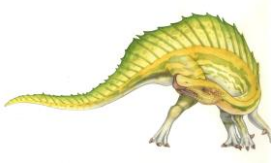
Operating-System Debugging

- **Debugging,** genel olarak sistemdeki hataların bulunması ve giderilmesi işlemlerini ifade eder.
- **Debugging** bug'lar ile ortaya çıkan performans sorunlarını gidermeyi de (**performance tuning**) amaçlar.
- Bir process başarısız olduğunda işletim sistemleri log dosyasına (**log files**) hata bilgisini kaydeder.
- İşletim sistemi ayrıca bir bellek dökümü (**core dump** processin belleğinin yakalanması) alabilir ve daha sonra analiz etmek üzere bir dosyada depolayabilir.
- Kernel'da ortaya çıkan hata **crash** olarak adlandırılır. Bir **crash** meydana geldiğinde, hata bilgileri bir log dosyasına kaydedilir ve bellek durumu bir **crash dump**a kaydedilir.
- **crash**lerin ötesinde performance tuning, sistem performansını optimize edebilir.
  - Bazen analiz için ***trace listings*** kaydedilir.
  - **Profiling** - istatistiksel eğilimleri araştırmak için instruction pointer in periyodik örneklemesidir.
- Debugging, bir kodu yazmaktan çok daha zordur.

Performance Tuning

- Performance tuning, processler yürütülürken oluşan darboğazları (processing bottlenecks) ortadan kaldırarak performansı iyileştirmeyi amaçlar.
- İşletim sistemi, sistem davranışının ölçümlerini hesaplamak ve görüntülemek için araçlar sağlamalıdır
- Örneğin Windows Task Manager (application information, processes, CPU and memory usage, and networking statistics)





Tracing

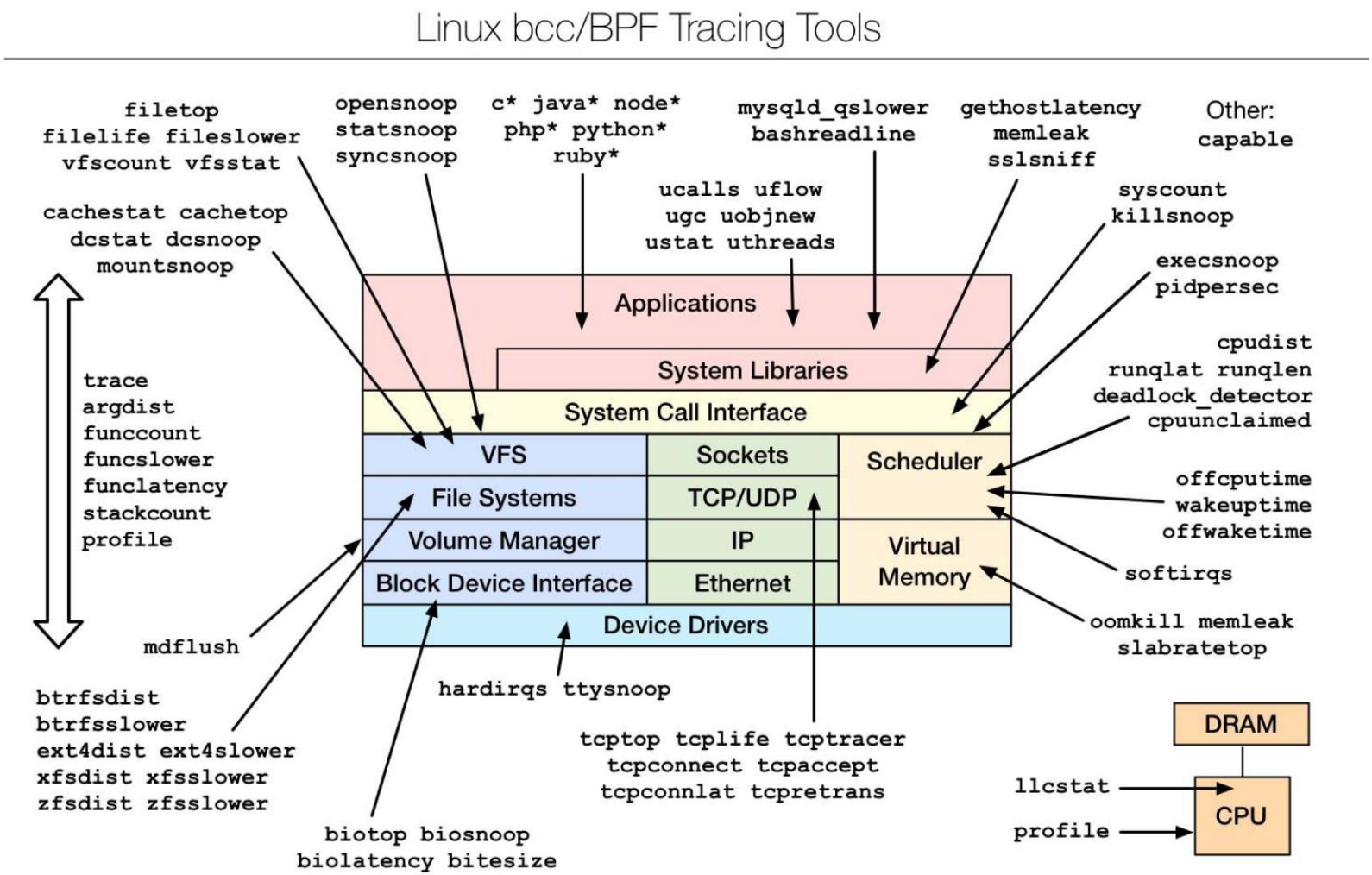
- Tracing tools bir sistem çağrısı yapıldığında yer alan adımlar gibi belirli bir olay için verileri toplar.
- Bazı Linux trace tool örnekleri
  - strace – bir process tarafından başlatılan sistem çağrılarını izler.
  - gdb – kaynak düzeyinde debugger
  - perf –Linux performans araçları kolleksiyonudur
  - tcpdump – network paketlerini toplar

BCC

- User-level ile kernel kodu arasındaki etkileşimlerde debugging yapmak, her iki kod setini de anlayan ve etkileşimlerini düzenleyebilen bir toolset olmadan neredeyse imkansızdır.
- BCC (BPF Compiler Collection) Linux için tracing işlevlerini sağlayan zengin bir toolkit
- Örneğin , disksnoop.py diskteki I/O aktivilerini izler.

```
TIME(s)          T      BYTES      LAT(ms)
1946.29186700    R      8              0.27
1946.33965000    R      8              0.26
1948.34585000    W      8192           0.96
1950.43251000    R      4096           0.56
1951.74121000    R      4096           0.35
```

Linux bcc/BPF Tracing Tools



End of Chapter 2