

# Operating Systems

Isfahan University of Technology  
Electrical and Computer Engineering Department  
1401-1 semester

Zeinab Zali

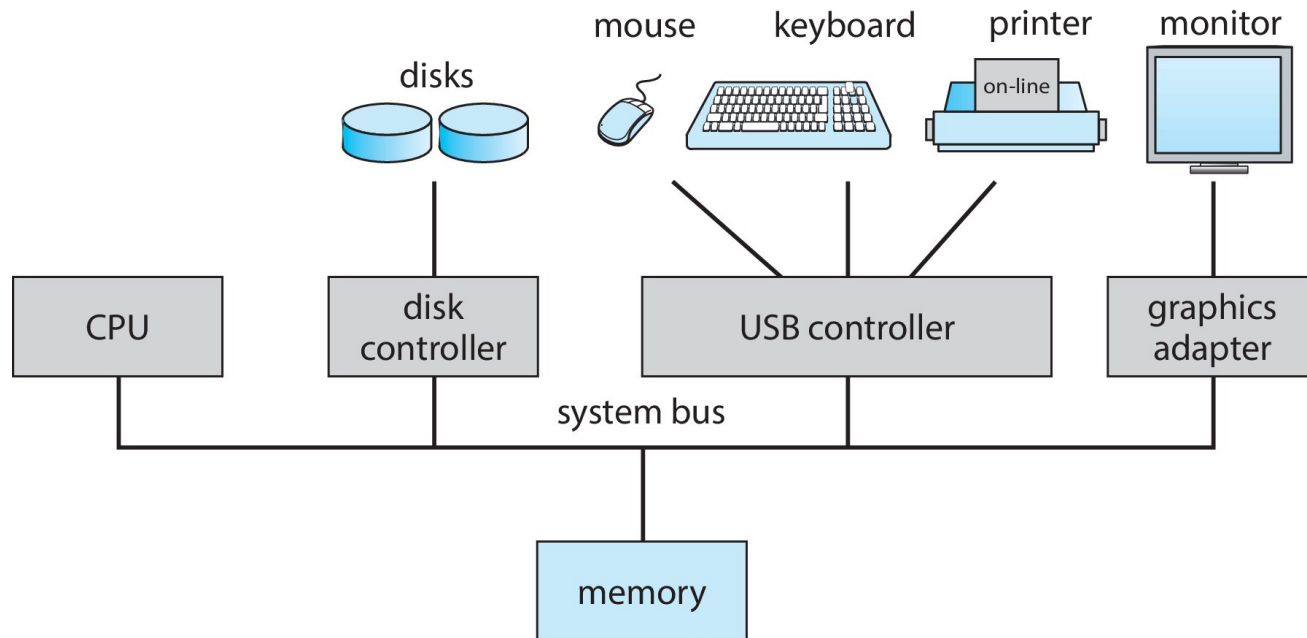
**Session 2: Computer and Operating System structure**

جلسه 2: ساختار کامپیوتر و سیستم عامل



# Computer System Organization

- One or more CPUs, device controllers connect through common **bus** providing access to shared memory
- Concurrent execution of CPUs and devices competing for memory cycles

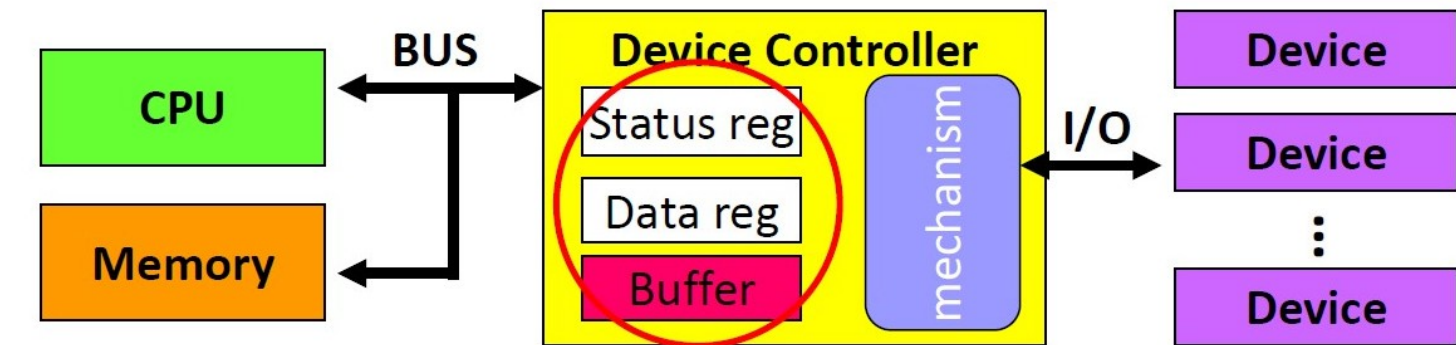


دستگاه های کامپیوتر از طریق یک باس بهم وصل شده اند که ارتباط این ها با هم برقرار باشد و از طریق این باس همشون به مموری وصل هستند



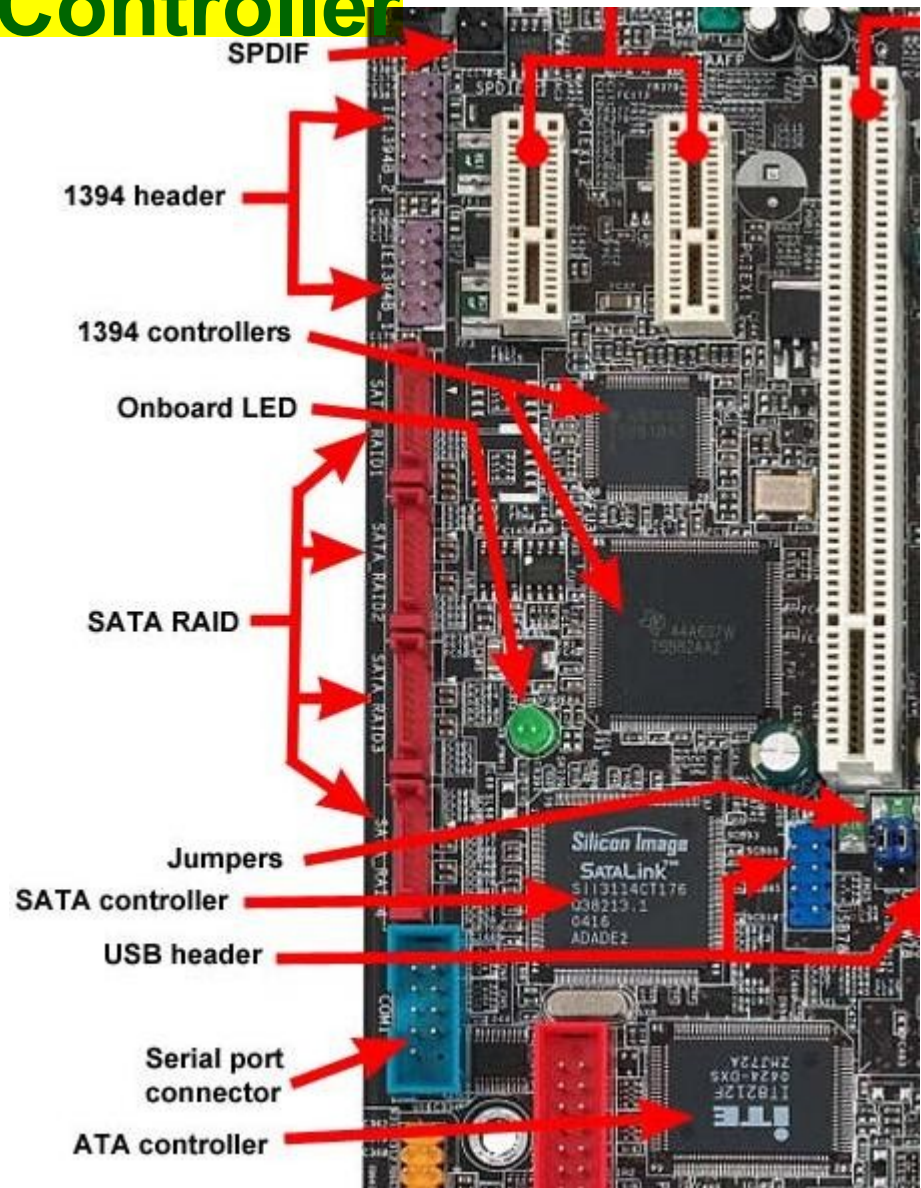
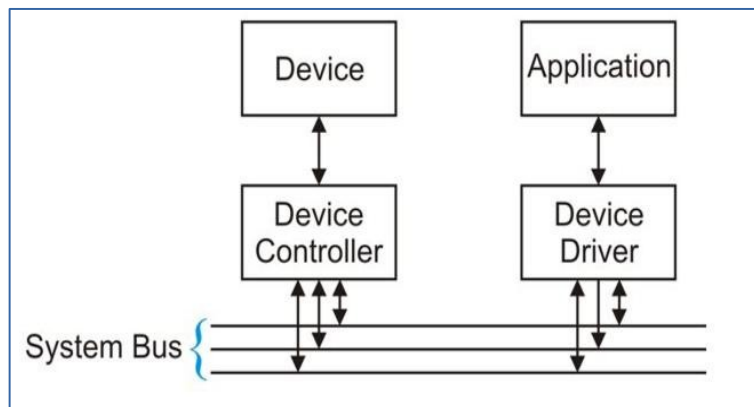
# Device Controller

- I/O devices and the CPU can execute concurrently
- Each device controller is in charge of a particular device type
- Each device controller has a **local buffer**
- Each device controller type has an operating system **device driver** to manage it
- CPU moves data from/to main memory to/from local buffers
- I/O is from the device to local buffer of controller
- Device controller informs CPU that it has finished its operation by causing an **interrupt**



همه دیوایس هایی که به سیستم وصل هستند یک **Controller** دارند روی برد لپ تاپ و ..  
و **Controller** یک بخش سخت افزاری است و وظیفه اش اینه که یک سری بافر و رجیستر داره و  
یک بافر داره برای اینکه اطلاعاتی که از دیوایس می خواد وارد سیستم یا خارج بشه می تونه از  
طریق این بافر انجام بشه و یک سری رجیسترهایی که کنترل این دیوایس رو فراهم می کنه و اجازه  
میده سیستم بتونه بفهمه که الان چیزی آماده است برای خوندن یا وضعیت دیوایس چجوریه و ...  
پس بین دیوایس ها و سیستم یک بخش سخت افزاری داریم به اسم **Device Controller**  
غیر از اینها ما یک **divice driver** هم داریم که توی بخش نرم افزارمون دیوایس رو داریم از  
طریق **divice driver** کنترلش میکنیم که توی صفحه بعد شکلش هست  
پس برای کنترل نوع ارتباطمون اینکه ما چی می خوایم از اون دیواس **divice driver** رو توی  
بخش نرم افزارمون داریم

# Device Controller



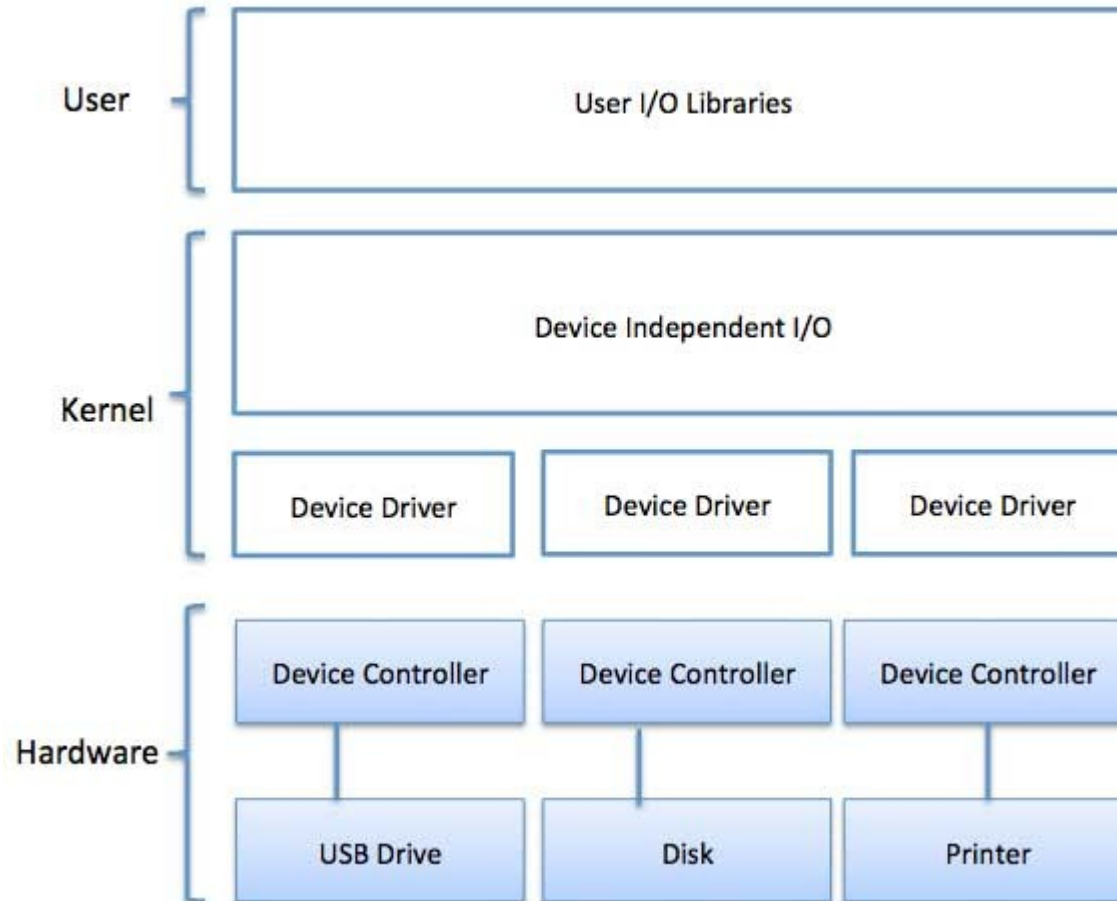
این یک قسمتی از برد کامپیوتر است

که روی برد اینو می بینیم

که یه سری جاهش Controller می بینیم و اینا امکان اینو فراهم می کنن که ما یک دیوایس خارجی رو وصل کنیم به کامپیوتر و I/O داشته باشیم



# Device Controller







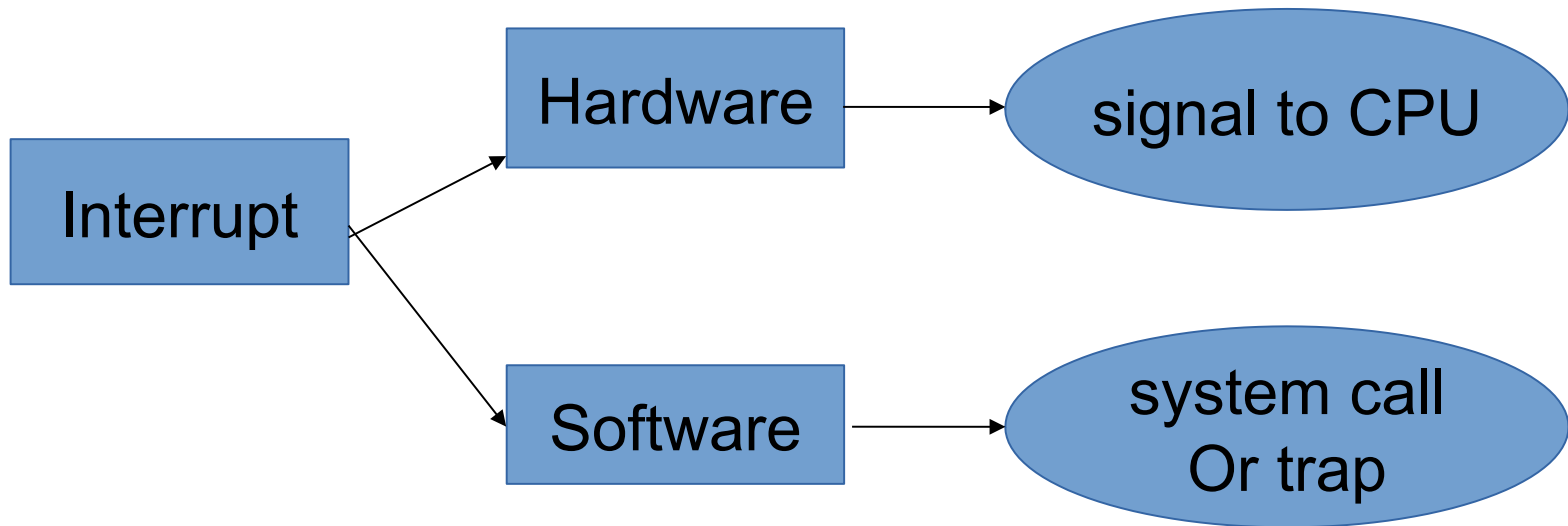
# Common Functions of Interrupts

- Interrupt transfers control to the interrupt **service routine** generally, through the **interrupt vector**, which contains the addresses of all the service routines
- Interrupt architecture must save the address of the interrupted instruction
- A **trap** or **exception** is a software-generated interrupt caused either by an error or a user request
- **An operating system is interrupt driven**



سیستم عامل یک نرم افزار interrupt driven است و اساس کارش روی وقفه ها است  
توی سیستم وقتی که cpu داره یک پروسس رو انجام میده و بعد یک دفعه یک دیوایسی که به  
سیستم وصله می خواد یک چیزی رو ارسال کنه به سیستم و اینجا با وقفه این کارو انجام میدیم چرا؟  
چون معمولاً انتظار داریم که ارتباطاتی که از طریق وقفه انجام میشه سریع باشه ینی برامون مهمه  
همون موقع انجام بشه  
هر وقفه ای با یک service routine هندل میشه ینی اگر یک وقفه ای اتفاق افتاد وابسته به اینکه  
اون وقفه چی هست service routine متناسب خودش اجرا میشه تا به یه کاری رسیدگی بشه  
و همه service routine هارو ما ادرسشونو می داریم توی یک وکتوری به اسم interrupt  
vector  
و این interrupt vector رو همیشه یه جایی توی رم نگهداریم میکنیم

# Interrupt types



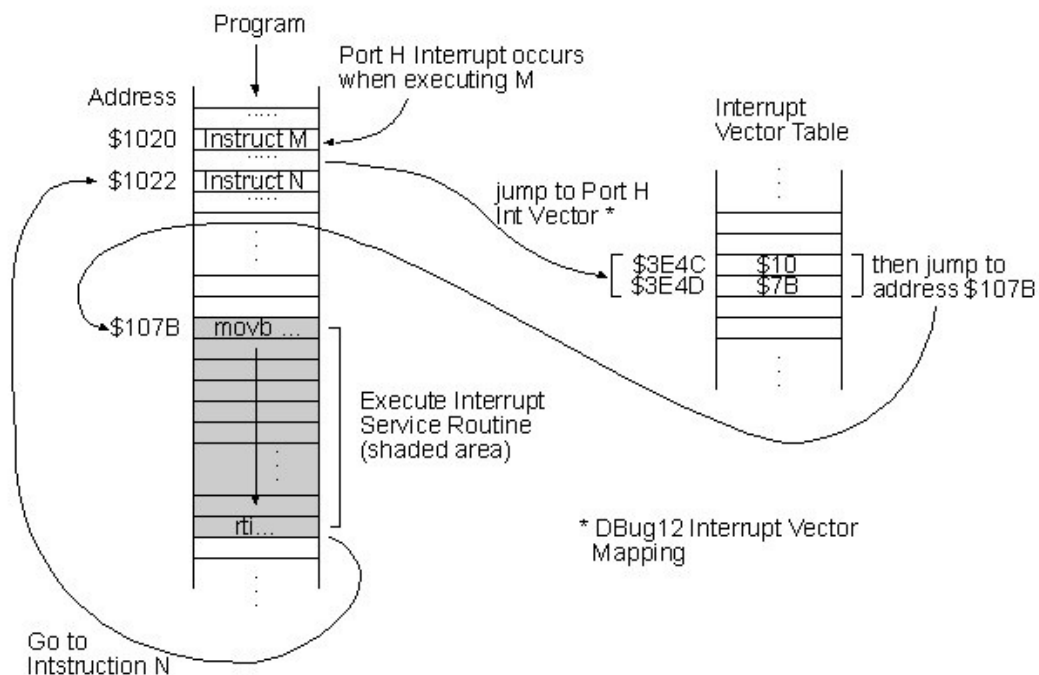
دو نوع وقفه داریم:

وقفه های سخت افزاری معمولاً یک نوع سیگنال به cpu است از سمت سخت افزار ولی وقفه های نرم افزاریمون جایی است که یک اکسپشنی توی نرم افزار اتفاق می افته ینی مثلاً توی اجرای اون قسمت کد یک اتفاقی می افته که ما باید به اون رسیدگی کنیم یا اینکه یک سیستم کال اتفاق می افته



# Interrupt Handling

- The operating system preserves the state of the CPU by storing the registers and the program counter
- Determines which type of interrupt has occurred:
- Separate segments of code determine what action should be taken for each type of interrupt









# Two I/O handling methods

- After I/O starts, control returns to user program only upon I/O completion
  - Wait instruction idles the CPU until the next interrupt
  - Wait loop (contention for memory access)
  - At most one I/O request is outstanding at a time, no simultaneous I/O processing
- After I/O starts, control returns to user program without waiting for I/O completion
  - **System call** – request to the OS to allow user to wait for I/O completion
  - **Device-status table** contains entry for each I/O device indicating its type, address, and state
  - OS indexes into I/O device table to determine device status and to modify table entry to include interrupt



- به دو روش می‌تونیم I/O رو هندل کنیم توی سیستمون:

1- بعد از اینکه I/O شروع شد کنترل اجرا از برنامه متوقف میشه و به انجام I/O می‌پردازه ینی cpu که قبلاً داشته یک برنامه user اجرا می‌کرده اونو متوقف میکنه و صرفاً میاد روی انجام I/O وقتی که ارسال کامل شد باز باید یک وقفه داده بشه تا سیستم بفهمه که تموم شده --> توی صفحه بعد شکل a میشه

2- توی صفحه بعد میشه شکل b که به این حالت میگن DMA : توی این حالت یک مشکلی که توی حالت a بوده رو رفع کردیم که این حالت رو باید سخت افزار داشته باشه ینی سخت افزار باید وقفه بده : توی این حالت یک وقفه ای داده شده و cpu می‌فهمه که یک انتقال اطلاعاتی قراره شروع بشه ولی بعدش میتونه سریع برگرده و برنامه قبلی یا یک برنامه دیگه رو اجرا بکنه و کاری به وقفه نداره ینی بعد از اینکه I/O شروع شد کنترل برمیگرده به اجرای برنامه بدون اینکه صبر بکنه که I/O کامل بشه که این همون حالتیه که توی سیستم کال ازش استفاده می‌کنیم مشکل حالت a این بود که زمان هدر می‌ره ینی cpu توی این فاصله نمی‌تونه چیز دیگه ای رو اجرا بکنه و همینطوری باید منتظر I/O باشه

Device-status table : چون ممکنه دیوایس‌های زیادی داشته باشیم که سیستم باهاشون در ارتباطه پس یک جدولی مدیریت میشه که این جدول اطلاعات رجیسترهای مختلف دیوایس‌های مختلف رو داره مثلاً می‌فهمه الان کدوم دیوایس توی حالت بیزیه و داره اطلاعات منتقل میکنه به رم یا کدوم دیوایس توی حالت idel است و از طریق این جدول مدیریت این دیوایس‌های مختلف رو انجام میده

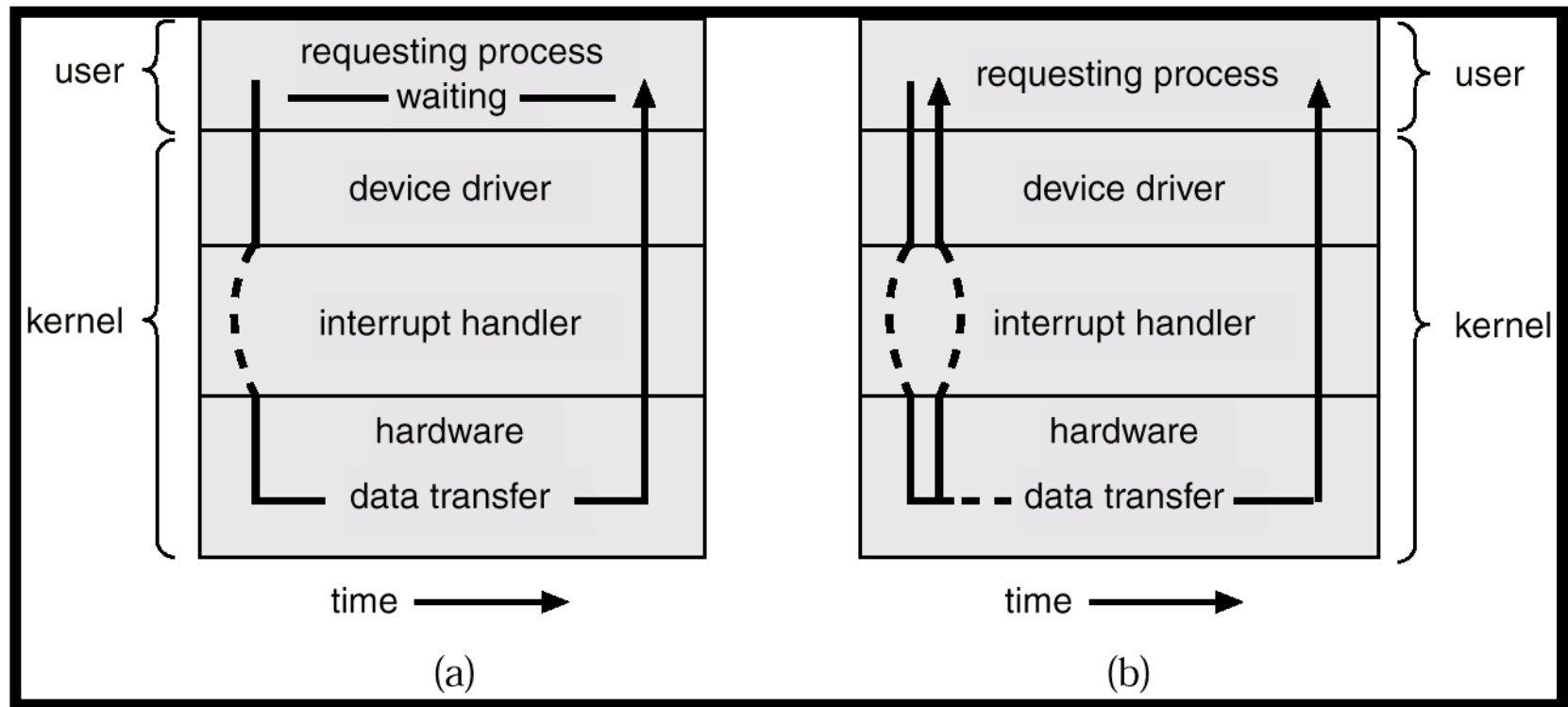


# Two I/O Methods

## Direct Memory Access (DMA)

Synchronous

Asynchronous

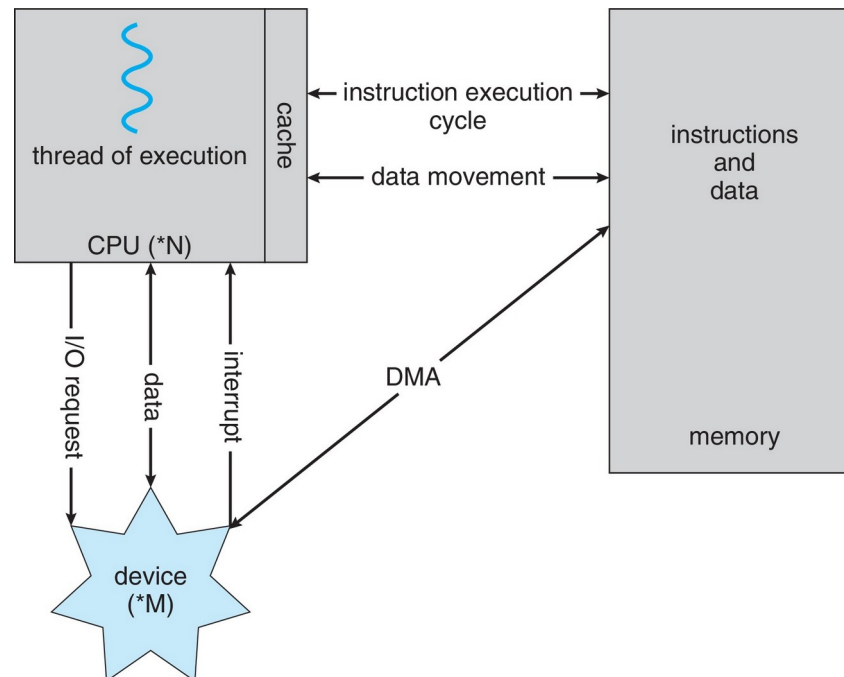






# Direct Memory Access Structure

- Used for high-speed I/O devices able to transmit information at close to memory speeds
- Device controller transfers blocks of data from buffer storage directly to main memory without CPU intervention
- Only one interrupt is generated per block, rather than the one interrupt per byte



وقتی که وقفه داده میشه که مشخص میشه این دیوایس قراره یه سری اطلاعات رو بفرسته روی رم یک ارتباط مستقیم بین دیوایس و مموری برقرار میشه به اسم DMA و دیگه سی پی یو کاری نداره که داره یک DMA اینجا انجام میشه و اون می تونه بقیه برنامه هاشو اجرا کنه و همزمان این اطلاعات داره از روی دیوایس به مموری منتقل میشه



# Kernel

- “The one program running at all times on the computer” is the **kernel**, part of the operating system
- Everything else is either
  - A **system program** (ships with the operating system, but not part of the kernel) , or
  - An **application program**, all programs not associated with the operating system
- Today’s OSES for general purpose and mobile computing also include **middleware** – a set of software frameworks that provide additional services to application developers such as databases, multimedia, graphics



ساختار سیستم عامل:

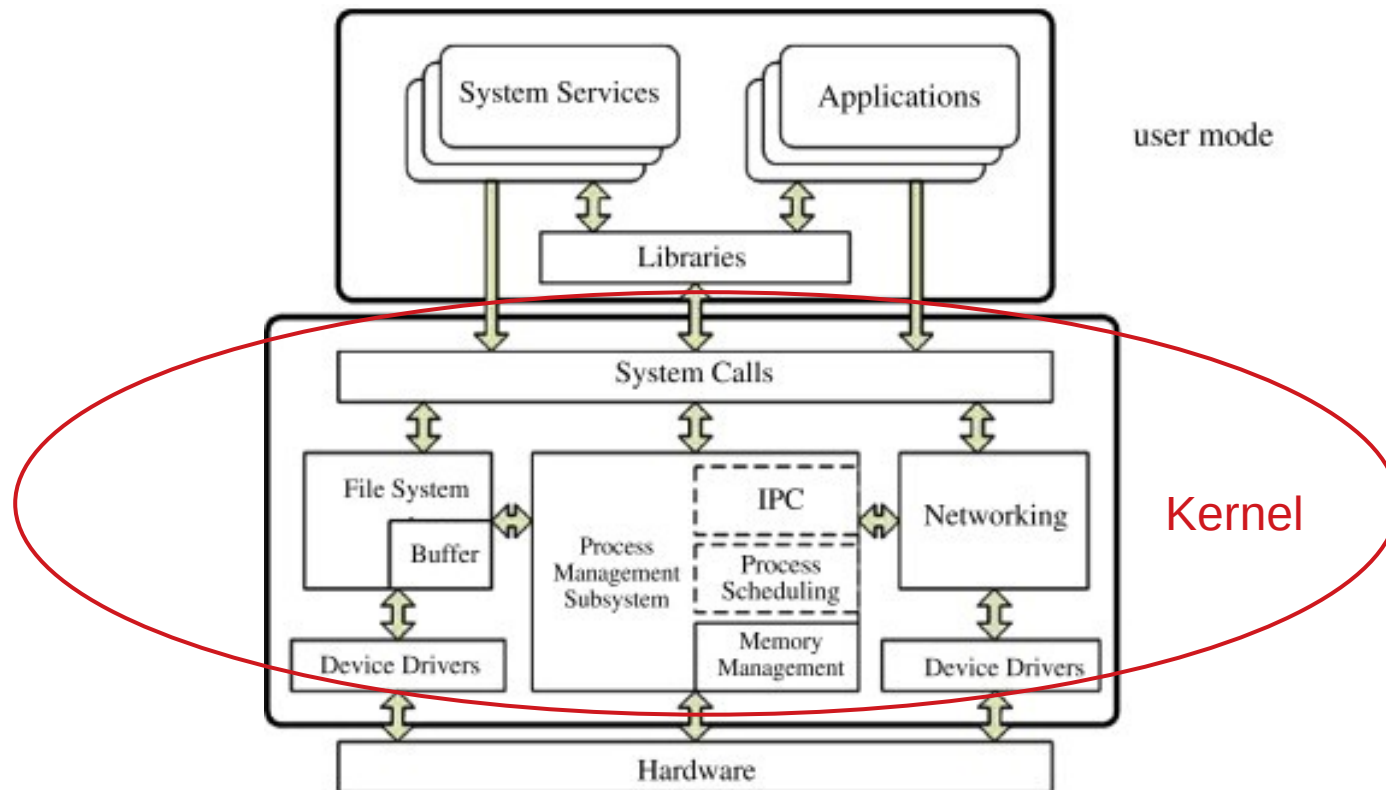
وقتی یک سیستم عامل نوشته میشه یک بخشی از سیستم عامل است که همیشه در حال اجرا است  
ینی این بخش حتما باید توی رممون باشه و دائما در حال اجرا باشه به این بخش میگیم کرنل  
کرنل ینی بخش اصلی سیستم عامل

بقیه چیزها که کنار کرنل می خریم یا نصب می کنیم یا system program یا application program

بیشتر با application program ها کار کردیم  
system program یکسری برنامه های سیستمی هستند که همراه با سیستم عامل دریافت میکنیم  
این middleware ها گاهی با همون os دریافت میشه: توضیحشو بالا بخون



# Kernel



داره کرنل رو نشون میده  
زیر کرنل سخت افزار رو داریم و بالای کرنل هم لایه یوزر رو داریم



# Operating System Services

- Operating systems provide an environment for execution of programs and services to programs and users
- One set of operating-system services provides functions that are helpful to the user:
  - **User interface** - Almost all operating systems have a user interface (**UI**).
    - ▶ Varies between **Command-Line (CLI)**, **Graphics User Interface (GUI)**, **touch-screen**, **Batch**
  - **Program execution** - The system must be able to load a program into memory and to run that program, end execution, either normally or abnormally (indicating error)
  - **I/O operations** - A running program may require I/O, which may involve a file or an I/O device
  - **File-system manipulation** - The file system is of particular interest. Programs need to read and write files and directories, create and delete them, search them, list file information, permission management.



User interface بالای سیستم کال است و جز Operating System است : توی شکل صفحه  
19 نشون داده شده



# Operating System Services (Cont.)

- One set of operating-system services provides functions that are helpful to the user (Cont.):
  - **Communications** – Processes may exchange information, on the same computer or between computers over a network
    - ▶ Communications may be via shared memory or through message passing (packets moved by the OS)
  - **Error detection** – OS needs to be constantly aware of possible errors
    - ▶ May occur in the CPU and memory hardware, in I/O devices, in user program
    - ▶ For each type of error, OS should take the appropriate action to ensure correct and consistent computing
    - ▶ Debugging facilities can greatly enhance the user's and programmer's abilities to efficiently use the system







# Operating System Services (Cont.)

- Another set of OS functions exists for ensuring the efficient operation of the system itself via resource sharing
  - **Resource allocation** - When multiple users or multiple jobs running concurrently, resources must be allocated to each of them
    - ▶ Many types of resources - CPU cycles, main memory, file storage, I/O devices.
  - **Logging** - To keep track of which users use how much and what kinds of computer resources
  - **Protection and security** - The owners of information stored in a multiuser or networked computer system may want to control use of that information, concurrent processes should not interfere with each other
    - ▶ **Protection** involves ensuring that all access to system resources is controlled
    - ▶ **Security** of the system from outsiders requires user authentication, extends to defending external I/O devices from invalid access attempts









# System Programs

---

- Once the kernel is loaded and executing, it can start providing services to the system and its users.
- If there are no processes to execute, no I/O devices to service, and no users to whom to respond, an operating system will sit quietly, waiting for something to happen ( a system call from user)
- Some services are provided outside of the kernel by **system programs (system services or system utilities)** that are loaded into memory at boot time to become system daemons, which run the entire time the kernel is running.
- system utilities provide a convenient environment for program development and execution. Some of them are simply user interfaces to system calls.







# System programs examples

---

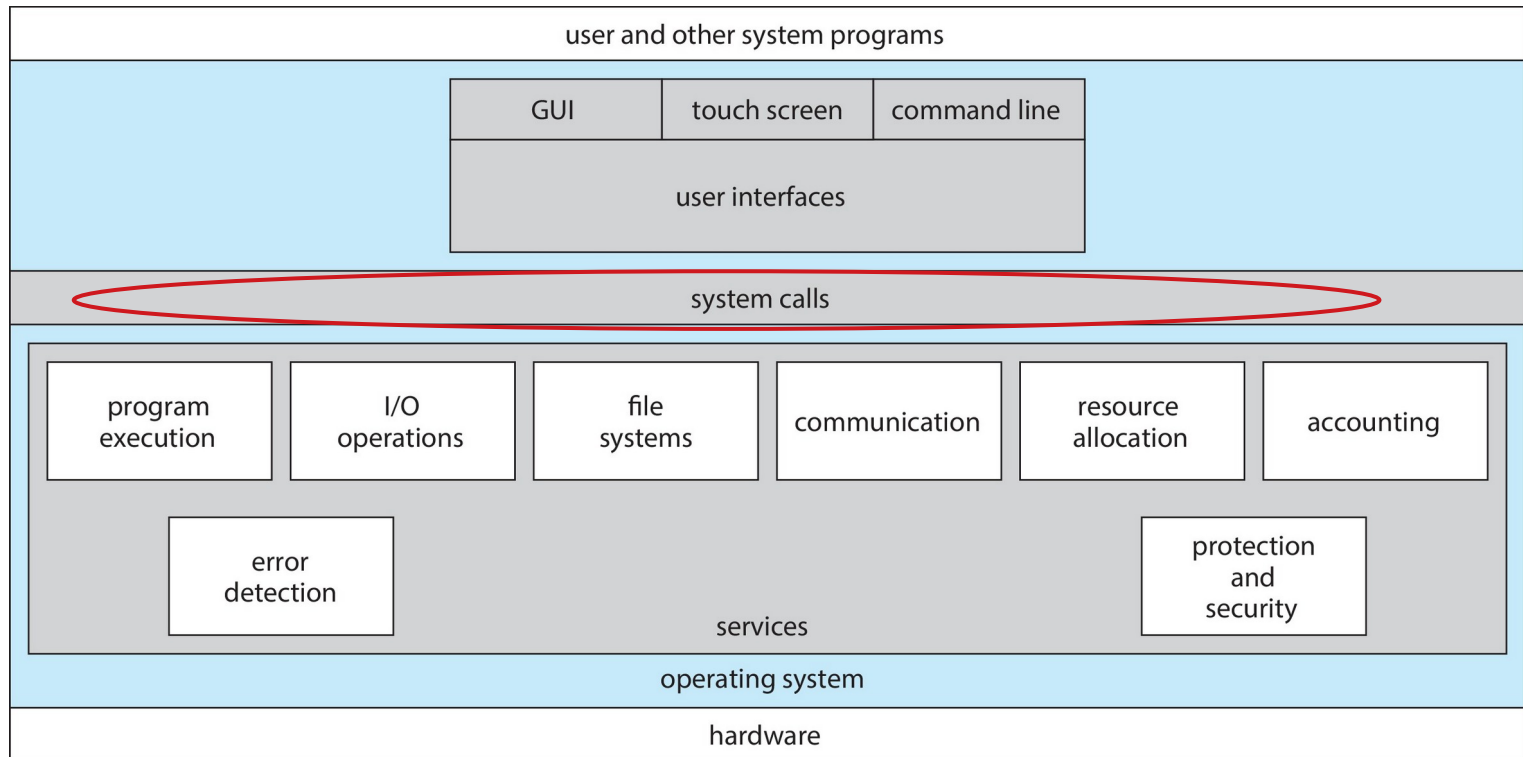
- **Systemd:** the first system program On Linux is“ systemd ,” and it starts many other daemons.
  - If there are no processes to execute, no I/O devices to service, and no users to whom to respond, an operating system will sit quietly, waiting for something to happen ( a system call from user)
  - List all the services controlled by systemd: `systemctl list-unit-files --type service`
- **Cron service:** Cron is a system that helps Linux users to schedule any task. However, a cron job is any defined task to run in a given time period
- **Gnome display manager:** a program that manages graphical display servers and handles graphical user logins







# System Calls



System program ها تو بخش یوزر است

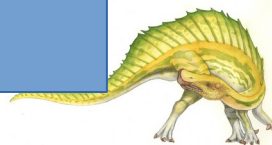
توی بخش یوزر از طریق لایه ای به اسم سیستم کال ها است که داریم از سیستم عامل استفاده می کنیم ینی یوزر لازم نیست که مستقیم با بخش های سیستم عامل حرف بزنه بلکه سیستم عامل اومده خودش یک لایه ایجاد کرده که خیلی ساده است که اصطلاحا API است که هر فانکشنی که سیستم عامل توی services برای ما فراهم میکنه api خودش رو داره و ما صرفا داریم از اون api استفاده می کنیم تو بخش یوزر



# System Calls

- **Programming interface to the services provided by the OS**
- Typically written in a high-level language (C or C++)
- Mostly accessed by programs via a high-level **Application Programming Interface (API)** rather than direct system call use
- Three most common APIs
  - Win32 API for Windows
  - POSIX API for POSIX-based systems (including virtually all versions of UNIX, Linux, and Mac OS X),
  - Java API for the Java virtual machine (JVM)

System call is the method used by a process to request action by the OS. The system call service routine is a part of the OS



سیستم کال چی هست؟ یک Programming interface است که مال os است پس اسم Programming interface سیستم عامل رو گذاشتیم سیستم کال که معمولا با یک زبان سطح بالا مثل c, ++c امکانش رو به ما داده که ارزش استفاده بکنیم

بعضی وقتا هم مستقیم از این سیستم کال استفاده نمی کنیم توی شکل صفحه قبل بالای سیستم کال یک سری کتابخونه داریم و اون کتابخونه ها اومدن توی پیاده سازیشون از سیستم کال ها استفاده کردن و ما داریم api اون کتابخونه رو استفاده میکنیم

پس سیستم کال چیزی نیست جز یک سری فانکشن و یک سری api که به ما می گه چجوری از سیستم عامل استفاده بکنیم

معروف ترین api هایی که داریم یکی win32 که برای ویندوز استفاده میشه و یکی هم posix که برای لینوکس استفاده میشه

پس سیستم کال یک متدی که پروسس استفاده میکنه تا یک اکشنی رو از سیستم عامل درخواست کنه  
ینی بهش بگه سیستم عامل چی کار کن برای من





# Dual-mode Operation

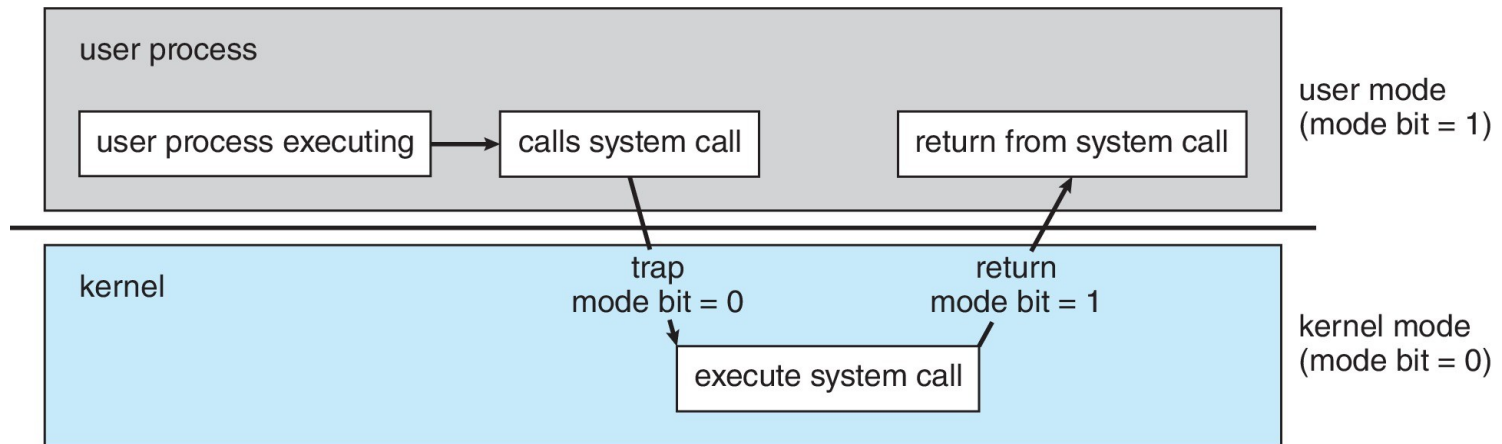
- **Dual-mode** operation allows OS to protect itself and other system components
  - **User mode** and **kernel mode**
- **Mode bit** provided by hardware
  - Provides ability to distinguish when system is running user code or kernel code.
  - When a user is running mode bit is “user”
  - When kernel code is executing mode bit is “kernel”
- How do we guarantee that user does not explicitly set the mode bit to “kernel”?
  - System call changes mode to kernel, return from call resets it to user
- Some instructions designated as **privileged**, only executable in kernel mode



- privileged : اون حالتی که دستورها حساسیت بیشتری روشن هستند رو بهشون می گیم  
privileged مد در واقع ما یوزر مدمون privileged نیست ینی محافظت شده نیست ولی کرنل  
مدمون privileged است



# Transition from User to Kernel Mode



یک برنامه داره توی یوزر اجرا میشه

حالا یک جاهایی این برنامه کارش ممکنه با سیستم عامل انجام بشه ینی این برنامه یک جاهایش اجراش مستلزم اینه که یک سیستم کال فراخوانی بشه منظور چیه؟

مثلا قراره که عملیات نوشتن توی یک فایل انجام بشه

پس اینجا ما باید سیستم کالی که مربوط به نوشتن هست رو فراخوانی بکنیم که ما مستقیما این کارو نمیکنیم ینی اون برنامه داره اجرا میشه و این تو دل برنامه است

حالا وقتی که می خواد این سیستم کال اجرا بشه ینی رایت می خواد اجرا بشه که رایت یک سیستم

کالی که مستقیما مال کرنل پس وقتی که میخواد این سیستم کال اجرا بشه اصطلاحا می گیم می ریم

تو مد کرنل ینی سی پی یو دیگه برنامه یوزر رو اجرا نمیکنه و خط کدهای مربوط به کرنل رو داره اجرا میکنه

چرا این دو تا لایه رو داریم از هم جدا می کنیم؟ یکی از وظایف سیستم عامل اینه که نباید اجازه بده

یوزرها هر جوری که دلشون خواست از سخت افزار استفاده بکنند چون باعث میشه مدیریتش قابل

کنترل نباشه پس ما اینجا دو تا مد تعریف میکنیم یک مد یوزر و یک مد کرنل و این دوتا مدل با یک

بیت سخت افزاری مشخص میشه توی سی پی یو ینی اگر مد یک باشه ینی سی پی یو الان در حال

اجرای یک دستور از بخش یوزر است و اگر صفر باشه ینی در حال اجرای یک دستور از مد کرنل است

نکته: این سیستم کال به صورت یک سرویس روتین نوشته شده ینی وقتی که ما api رایت رو

فراخوانی میکنیم این به این معنی که یک وقفه نرم افزاری اتفاق افتاده و باید بره توی اون اینتراپت

وکتوری که ادرس های سیستم کال های مختلف توشه رو بهش مراجعه میکنه و بعد از اون مراجعه

میکنه به اون سیستم کال که مثلا مثال توی صفحه 28 مربوط به open بوده و این اجرا میشه و بعد

که اجرا تموم شد سی پی یو برمی گرده رو مد یوزر و بقیه کد مربوط به بخش یوزر رو اجرا میکنه

نکته: سی پی یو دقیقا یک بیتی داره که مشخص میکنه توی چه مدی داره اجرا میشه

نکته: نباید وقتی که سی پی یو تو حالت یوزر مد است دستوری که مربوط به سیستم کال ها است

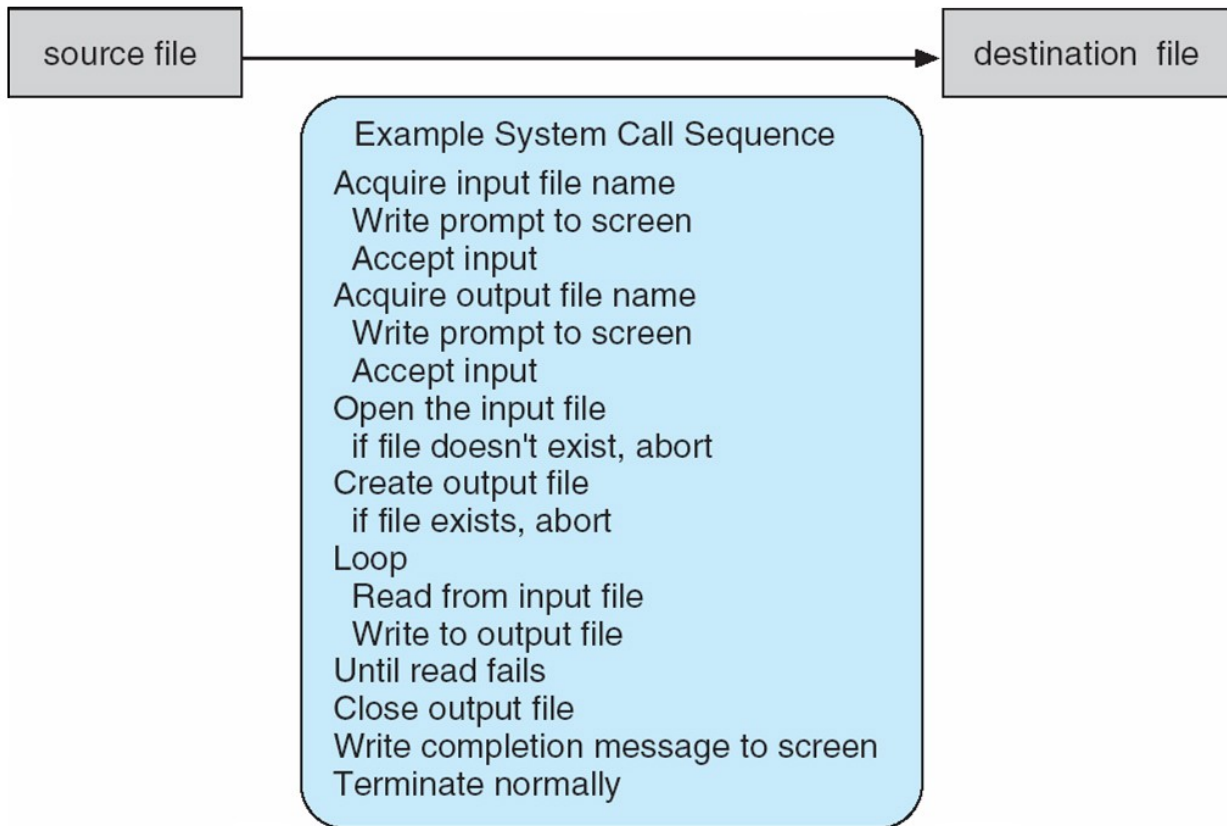
اجرا بشه و این ممنوع است

نکته: بعضی از سیستم ها بیشتر از دو تا مد دارند



# Example of System Calls

- System call sequence to copy the contents of one file to another file







# Example of Standard API

## 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:

<pre>#include &lt;unistd.h&gt;</pre>		
<pre>ssize_t</pre>	<pre>read</pre>	<pre>(int fd, void *buf, size_t count)</pre>
<div></div>	<div></div>	<div></div>
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`.









# System Call Implementation

---

- Typically, a number is associated with each system call
  - **System-call interface** maintains a table indexed according to these numbers
- The system call interface invokes the intended system call in OS kernel and returns status of the system call and any return values
- The caller need know nothing about how the system call is implemented
  - Just needs to obey API and understand what OS will do as a result call
  - Most details of OS interface hidden from programmer by API
    - ▶ **Managed by run-time support library (set of functions built into libraries included with compiler)**



- این هم بالاتر گفتیم  
کلا مستقیم با api سیستم کال کار نمی کنیم بلکه با کتابخونه ها کار میکنیم



# System Call Parameter Passing

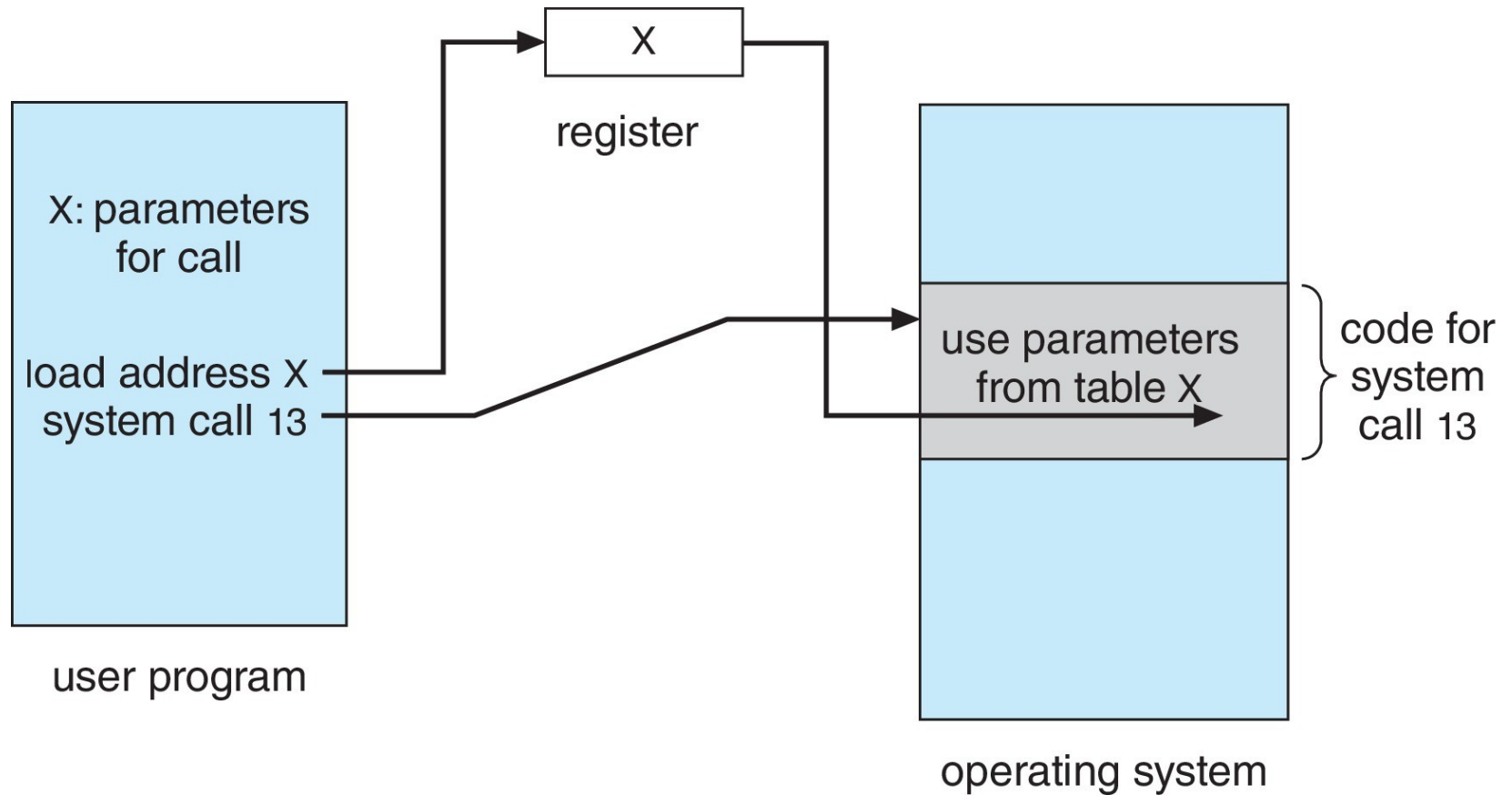
- Often, more information is required than simply identity of desired system call
  - Exact type and amount of information vary according to OS and call
- Three general methods used to pass parameters to the OS
  - Simplest: pass the parameters in registers
    - ▶ In some cases, may be more parameters than registers
  - Parameters stored in a block, or table, in memory, and address of block passed as a parameter in a register
    - ▶ This approach taken by Linux and Solaris
  - Parameters placed, or **pushed**, onto the **stack** by the program and **popped** off the stack by the operating system
  - Block and stack methods do not limit the number or length of parameters being passed







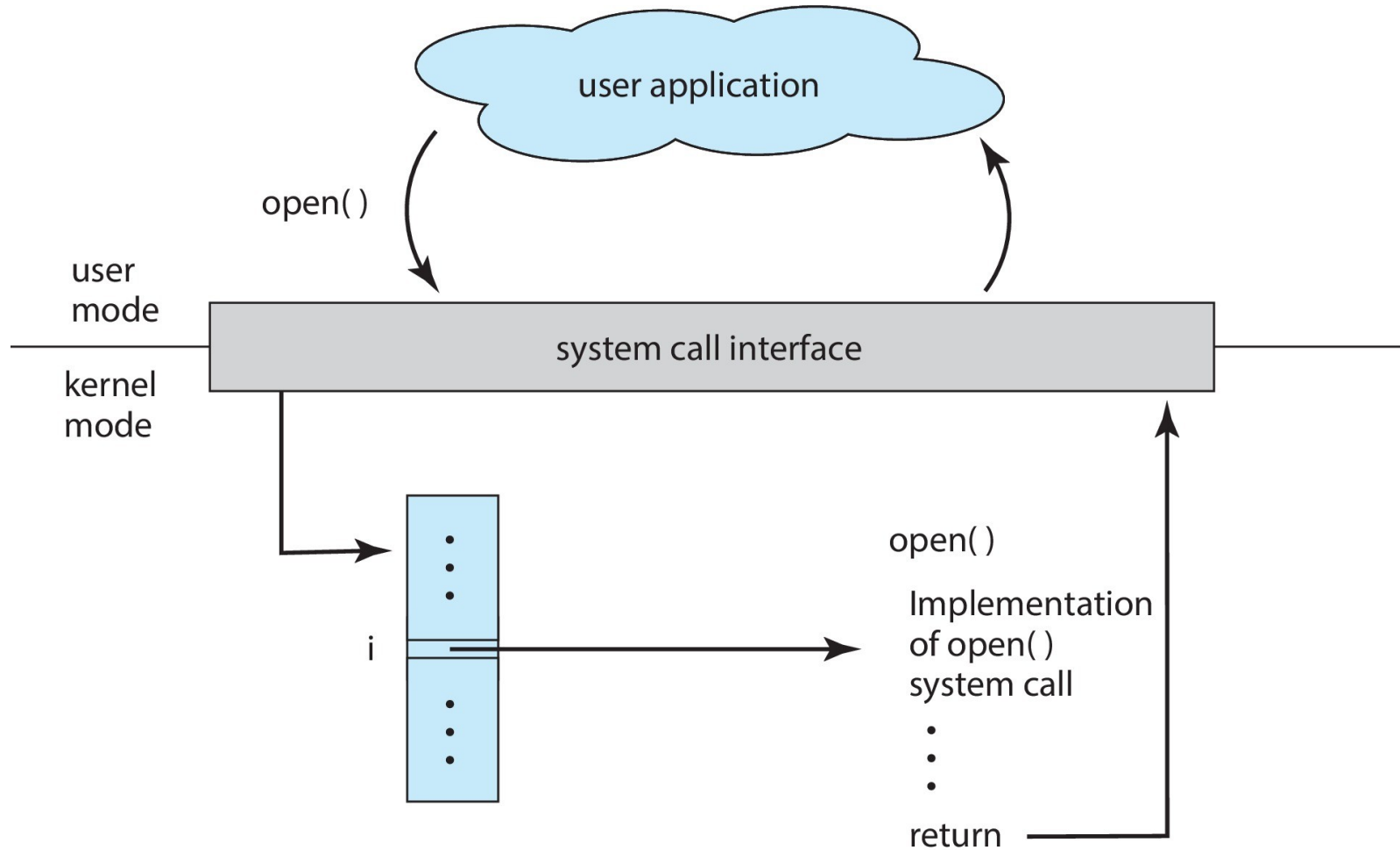
# Parameter Passing via Table







# API – System Call – OS Relationship



API – تماس سیستم – رابطه سیستم عامل

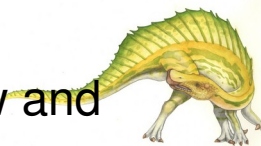




# Types of System Calls

---

- Process control
  - Create, terminate, end, abort, load, execute, ...
  - Debugger for determining bugs, single step execution
  - Locks for managing access to shared data between processes
- File management
  - create file, delete file, open, close, read, write, ...
- Device management
  - request, release, read, write, attach or detach devices
- Information maintenance
  - get time or date, set time or date, get system data, ...
- Communications
  - create, delete communication connection, ...
- Protection
  - Control access to resources, Get and set permissions, Allow and deny user access, ...







# Examples of Windows and Unix System Calls

## EXAMPLES OF WINDOWS AND UNIX SYSTEM CALLS

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

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

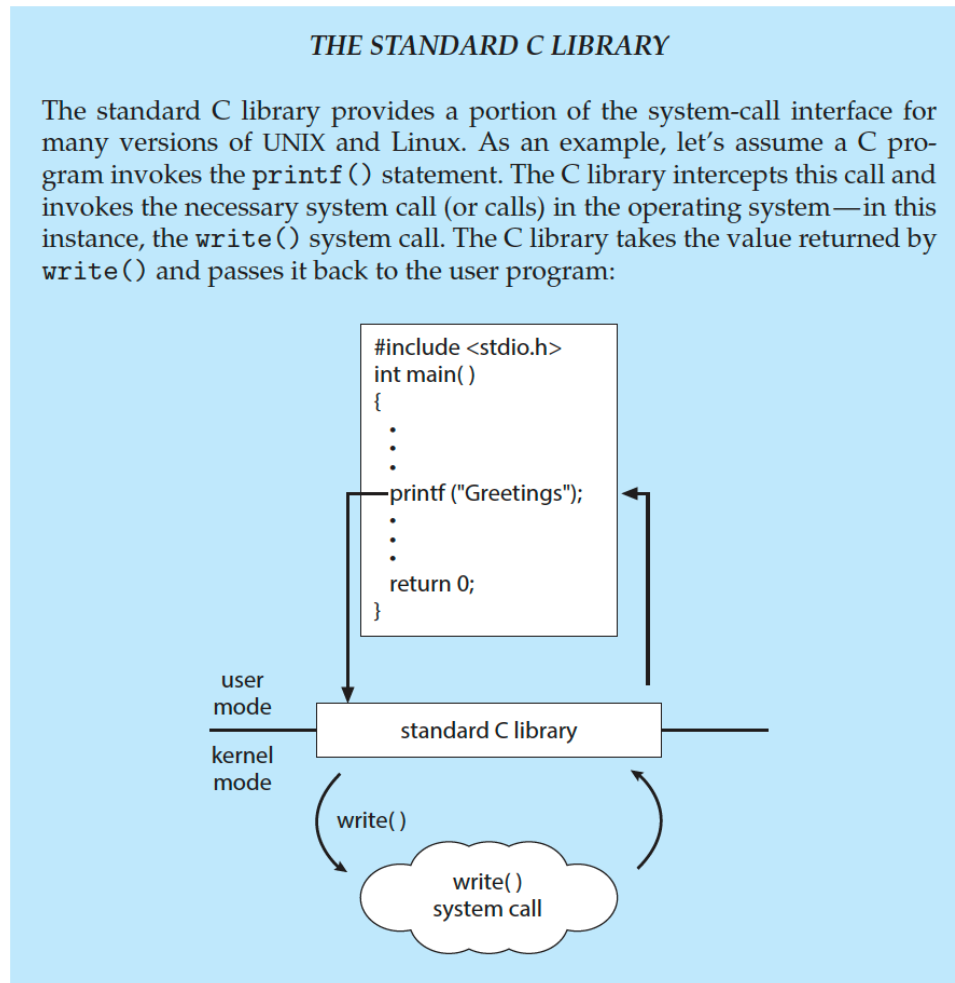


نمونه هایی از تماس های سیستمی ویندوز و یونیکس



# Standard C Library Example

- C program invoking `printf()` library call, which calls `write()` system call



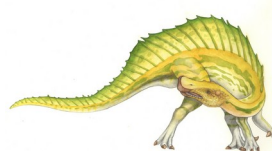
مثال:



# Command Line interpreter

- CLI or **command interpreter** allows direct command entry
- **Function:** Primarily fetches a command from user and executes it
- **Organization:** Sometimes implemented in kernel, sometimes by systems program
- **Multiple flavors** implemented – **shells**
  - Bourne shell, bash, third-party shells
- **Implementation approaches:**
  - commands are built-in
  - Commands are just names of programs
    - Benefit: flexible for adding new features without shell modification

```
echo $SHELL
```



-

Command Line interpreter : کارش اینه که یک دستور رو از یوزر می گیره و اجرا میکنه  
شل ها دو مدل هستند:

1- یا اصطلاحا اون کامندهایی که قراره توشون اجرا بشه built-in است

2- یا built-in نیست و در واقع اسم یکسری برنامه است





# Bourne Shell Command Interpreter

```
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/mmfstd
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/mmfstd
-r-x----- 1 root root 20667161 Jun  3  2015 /usr/lpp/mmfs/bin/mmfstd
[root@r6181-d5-us01 ~]#
```







# User Operating System Interface - GUI

---

- User-friendly **desktop** metaphor interface
  - Usually mouse, keyboard, and monitor
  - **Icons** represent files, programs, actions, etc
  - Various mouse buttons over objects in the interface cause various actions (provide information, options, execute function, open directory (known as a **folder**))
  - Invented at Xerox PARC
- Many systems now include both CLI and GUI interfaces
  - Microsoft Windows is GUI with CLI “command” shell
  - Apple Mac OS X is “**Aqua**” GUI interface with UNIX kernel underneath and shells available
  - Unix and Linux have CLI with optional GUI interfaces (**CDE**, **KDE**, **GNOME**)

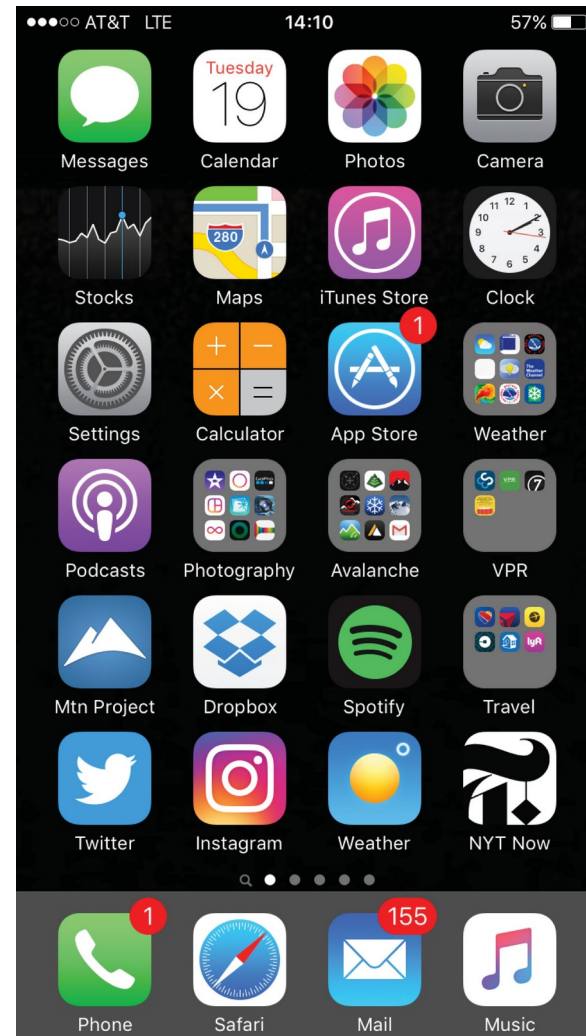






# Touchscreen Interfaces

- Touchscreen devices require new interfaces
  - Mouse not possible or not desired
  - Actions and selection based on gestures
  - Virtual keyboard for text entry
- Voice commands





# Quick student research

Please find  
system call table in the kernel of your linux  
Provide two samples of system call interface code

