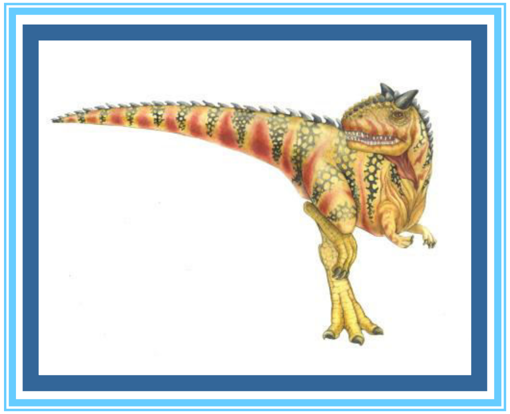
Section 2: Introduction



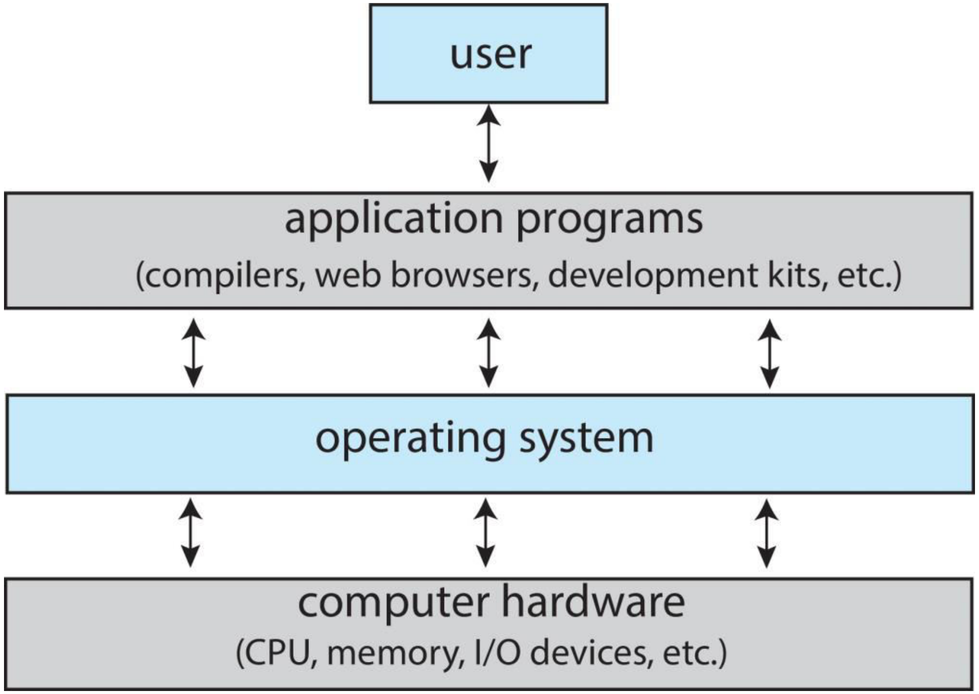
Section 2: Overview

- **Chapters 1 and 2 in textbook *Operating System Concepts***
  - Overall computer system
  - Computer hardware
  - User interfaces
  - OS versus kernel
  - User mode / kernel mode
  - System calls
  - Interrupts

Computer system structure

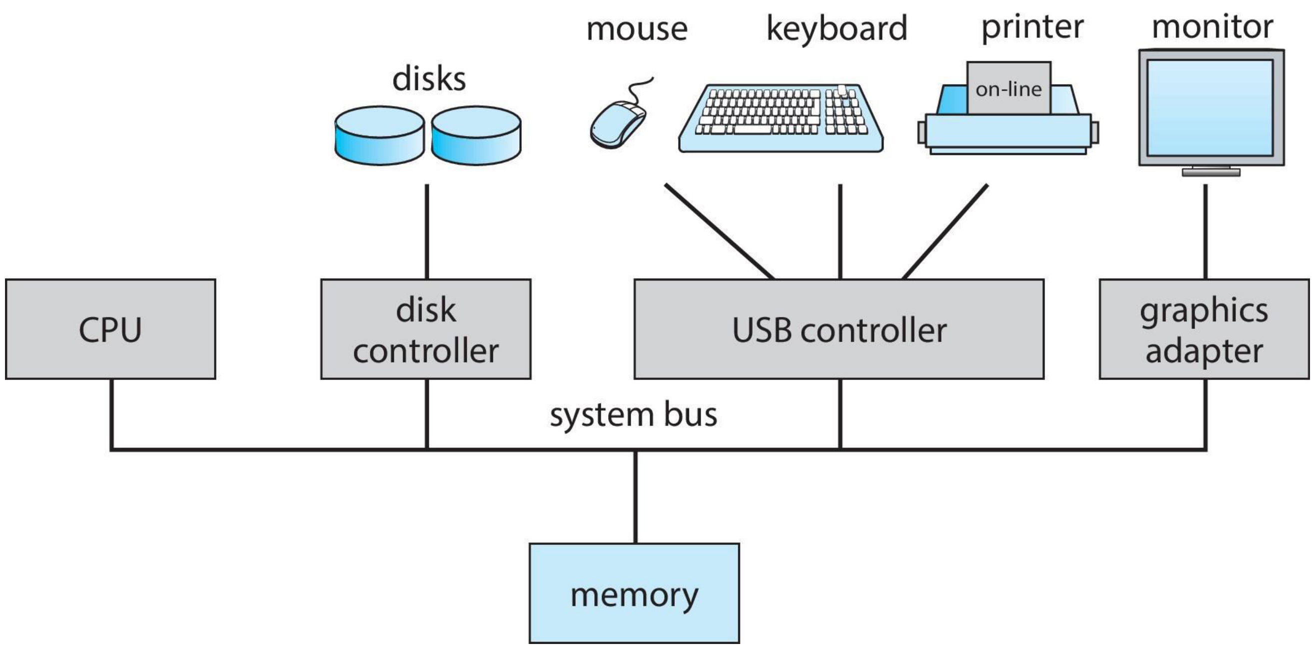
- Computer system can be divided into three components:
  - Application programs – define the ways in which the system resources are used to solve the computing problems of the users
    - Word processors, compilers, web browsers, database systems, video games
  - Operating system
    - Controls and coordinates use of hardware among various applications and users
  - Hardware – provides basic computing resources
    - CPU, memory, I/O devices

View of computer system structure

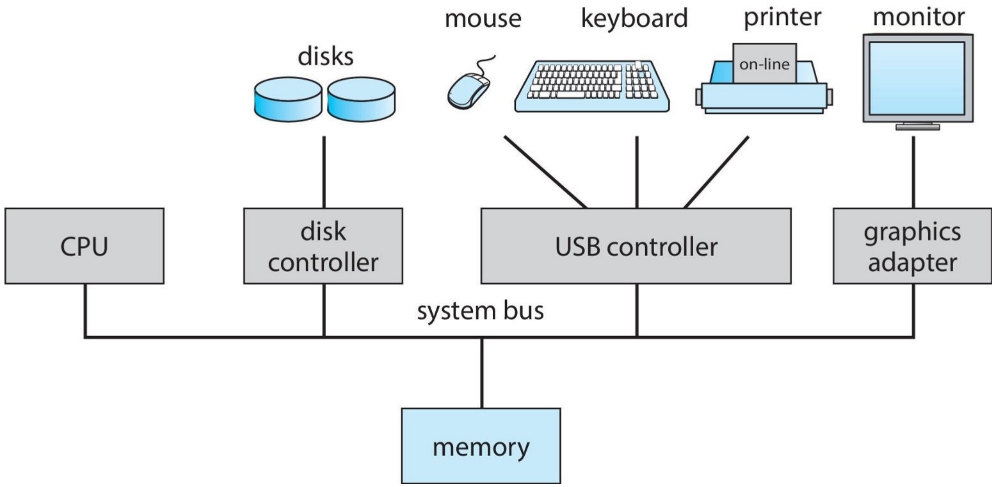


Computer System Hardware

- Computer-system hardware
  - One or more CPUs, device controllers connect through common **bus** providing access to the memory and other I/O devices



Input/Output (I/O) devices

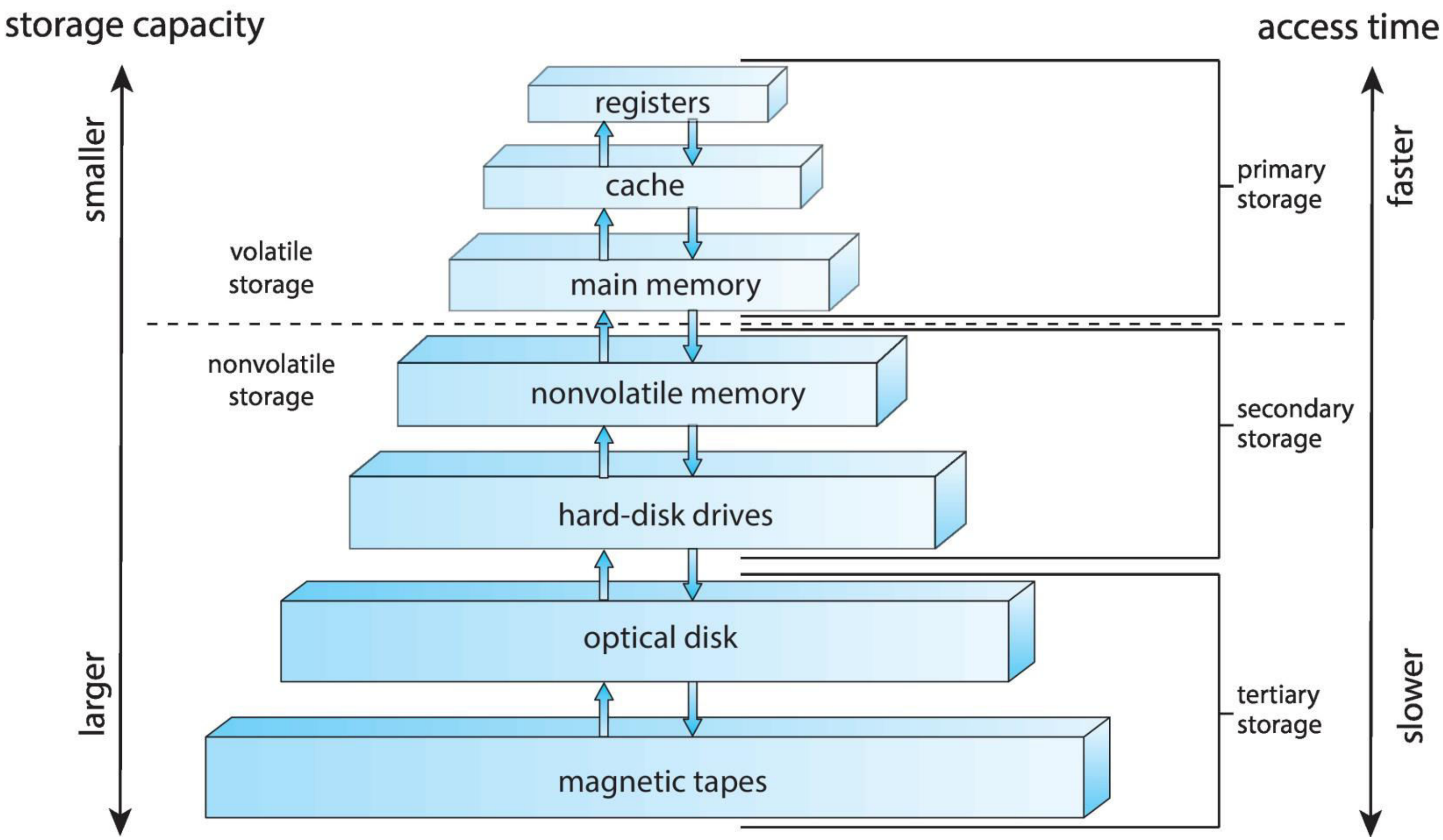


- I/O devices consist of storage devices (disks, tapes), transmission devices (network connections, Bluetooth), and human-interface devices (screen, keyboard, mouse,audio in and out)
- I/O devices have two physical parts: a controller and the device itself
- A device controller is a chip that physically controls the device

Storage (memory)

- Storage system organized in a hierarchy; each level of the hierarchy has different
  - Speed
  - Cost
  - Volatility

Storage-device hierarchy



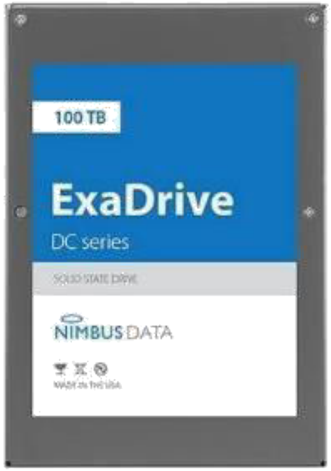
Main memory

- Main memory – is the only large storage media that the CPU can access directly
  - Typically **volatile**
  - Typically **random-access memory** in the form of **Dynamic Random-access Memory (DRAM)**

External memory

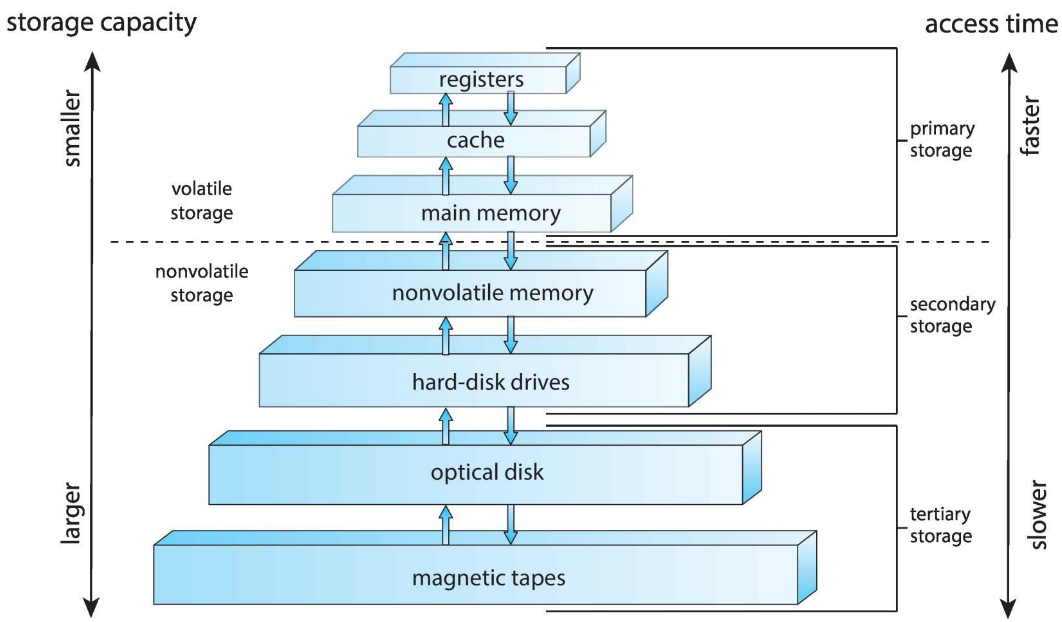
- Either hard disk drive, range from 30GB to 9TB, such as X18 with 9 platters, 18 heads

- Or Solid-State Drive SSD with up to 100TB



Storage Structure

- Main memory – is the only large storage media that the CPU can access directly
  - Typically **volatile**
  - Typically **random-access memory** in the form of **Dynamic Random-access Memory (DRAM)**
- Secondary storage – extension of main memory that provides large **nonvolatile** storage capacity



Characteristics of Various Types of Storage

Level	1	2	3	4	5
Name	registers	cache	main memory	solid-state disk	magnetic disk
Typical size	< 1 KB	< 16MB	< 64GB	< 1 TB	< 10 TB
Implementation technology	custom memory with multiple ports CMOS	on-chip or off-chip CMOS SRAM	CMOS SRAM	flash memory	magnetic disk
Access time (ns)	0.25-0.5	0.5-25	80-250	25,000-50,000	5,000,000
Bandwidth (MB/sec)	20,000-100,000	5,000-10,000	1,000-5,000	500	20-150
Managed by	compiler	hardware	operating system	operating system	operating system
Backed by	cache	main memory	disk	disk	disk or tape

Movement between levels of storage hierarchy can be explicit or implicit

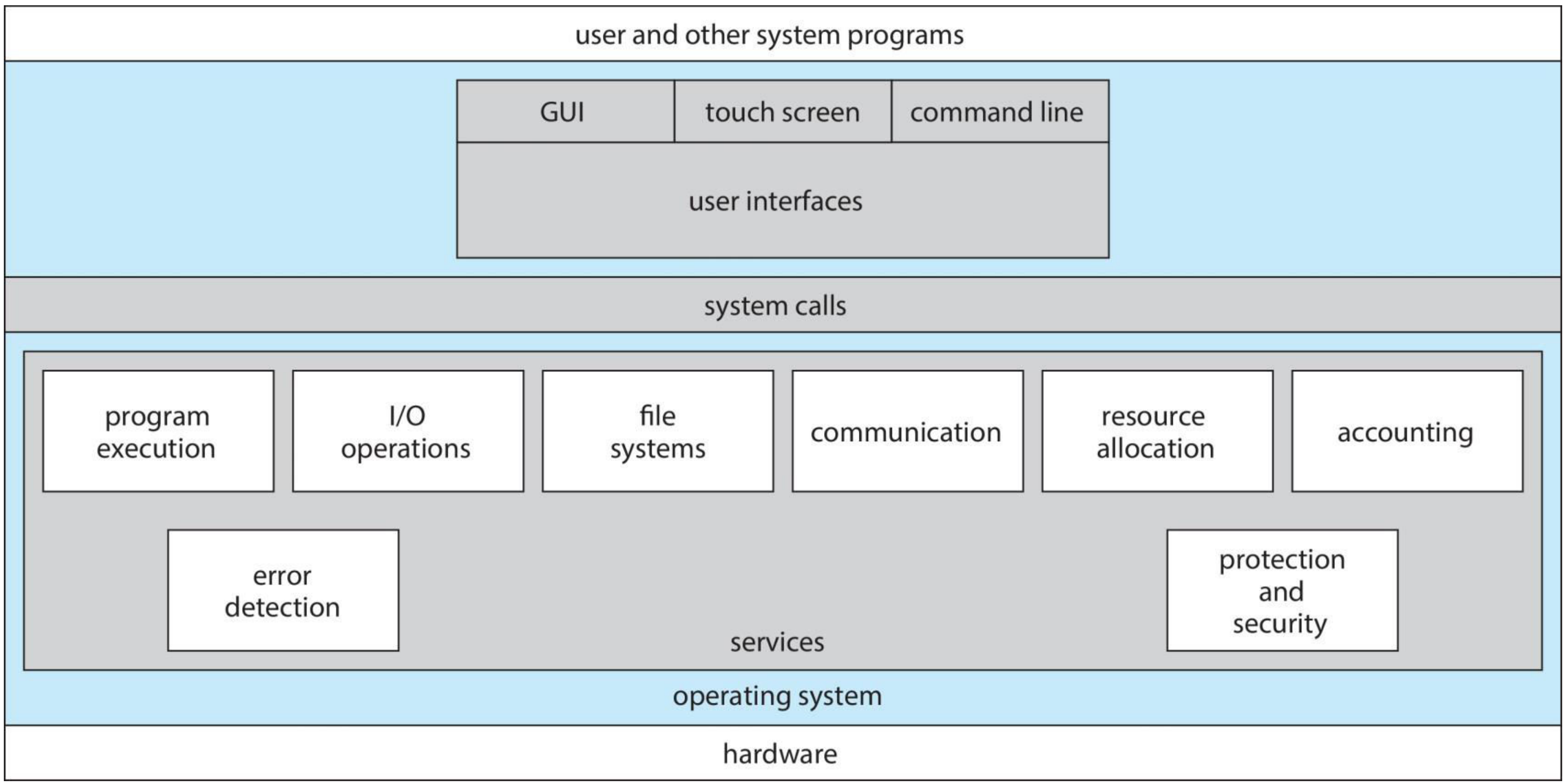
Computer time units at human scale

- 1 ns = 1/1 billion of a second
- A 2Ghz CPU executes one instruction every 0.5ns
- In order to bring execution time of computer operations to human scale, let assume that 0.5n = 1 second
- Note, the fastest typist can type a keystroke every 100ms

Item	Time		Time in human terms
Processor cycle	0.5ns	2Ghz	1 second
Memory access	15ns		30 seconds
Context switch	5,000ns	$5\mu s$	167 minutes
Disk access	7,000,000ns	7ms	162 days
One keystroke	100,000,000ns	100ms	6.3 years

Table 1: Time scales (A fast typist can type a keystroke every 100 milliseconds)

User interfaces



Command line 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/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 ~]#
```

Command Line interpreter

- CLI allows direct command entry
- Primarily fetches a command from user and executes it
- Sometimes commands built-in, sometimes just names of programs
  - If the latter, adding new features doesn't require shell modification
- Sometimes implemented in kernel, sometimes by systems program
- Sometimes multiple flavors implemented – **shells**
  Ubuntu has Bourne shells (bash, sh, zsh)

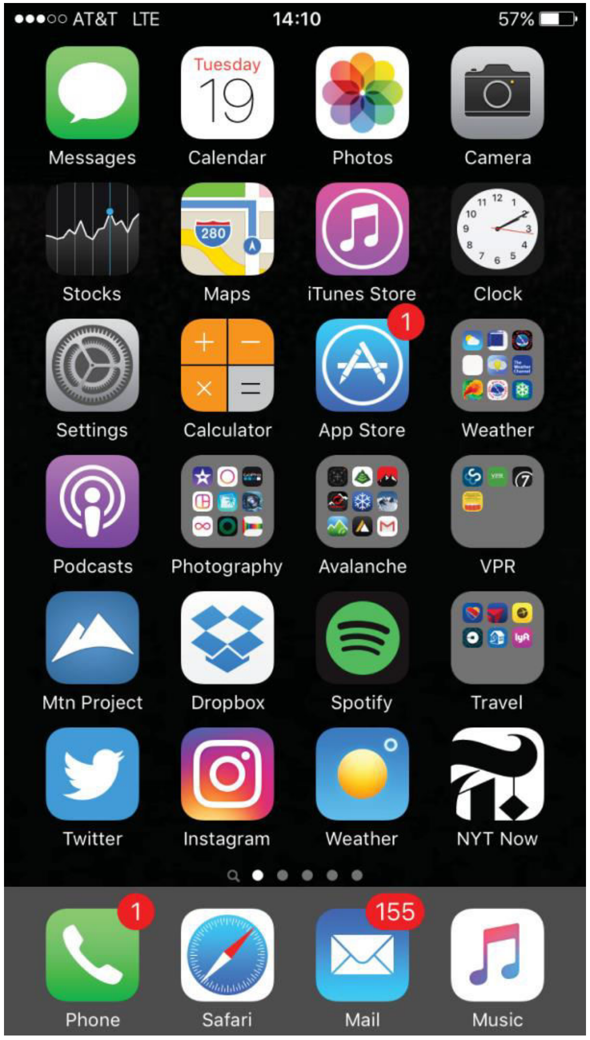
Graphical User Interfaces - GUI

- 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 GUI interfaces (CDE, KDE, GNOME)

Windows 10 screen desktop

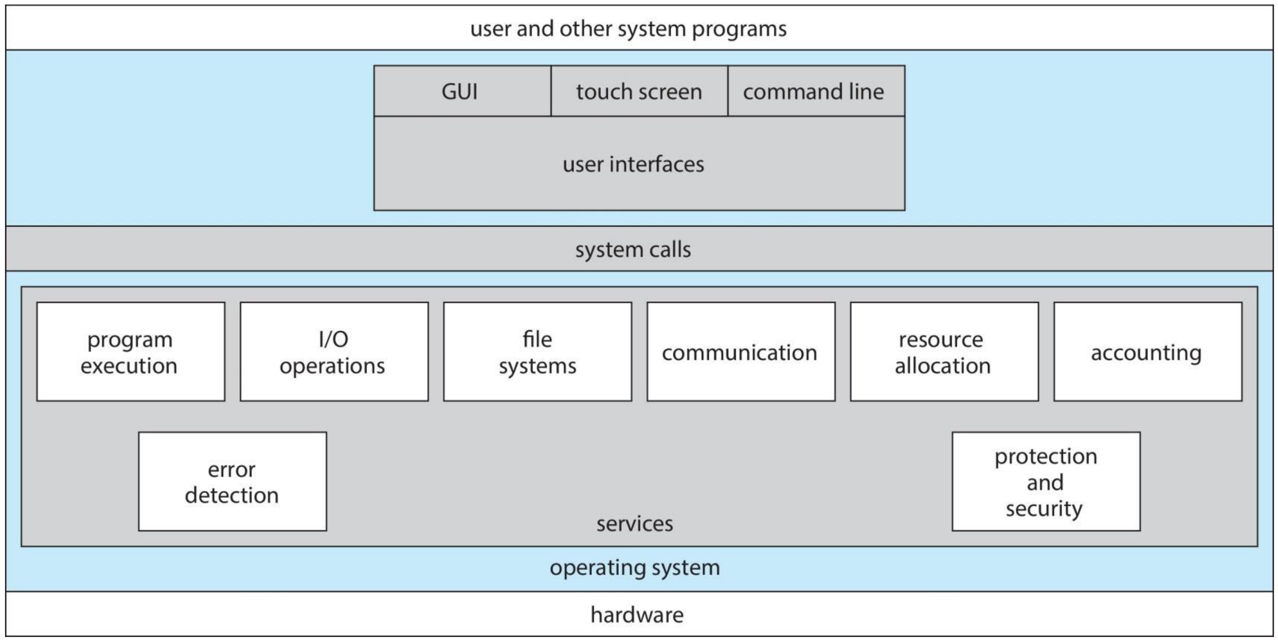
Touchscreen Interfaces

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



Difference between OS and kernel

- The kernel is a program (object) that is part of the OS



- Process and memory management is done by the kernel while interfacing with the user such as desktop icons or command line interpreters (terminals) are part of the OS but not in the kernel
- Note, the kernel is stored in a protect area of the main memory called the kernel space, no other programs can access this part of main memory

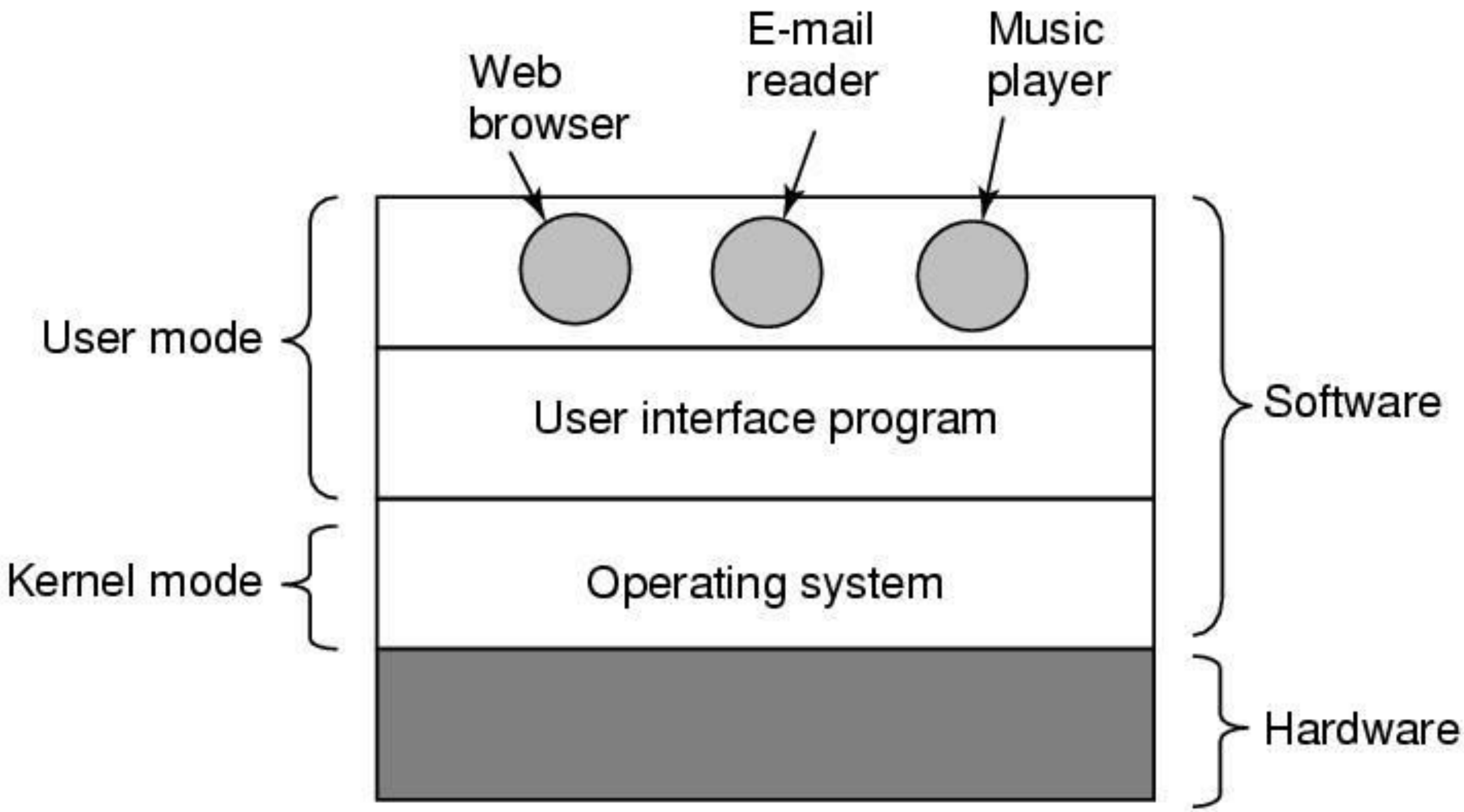
User mode / kernel mode

- Most computers have at least two modes of operation: **kernel mode** and **user mode**
- The code of most of the services provided by the OS is accessed in kernel mode
- This code is executed in kernel mode since instructions in this code have access to all the hardware and can execute any machine instruction the computer is capable of executing
- The rest of the software, in particular the user code, always runs in user mode where only a subset of the machine instructions is available

C code example

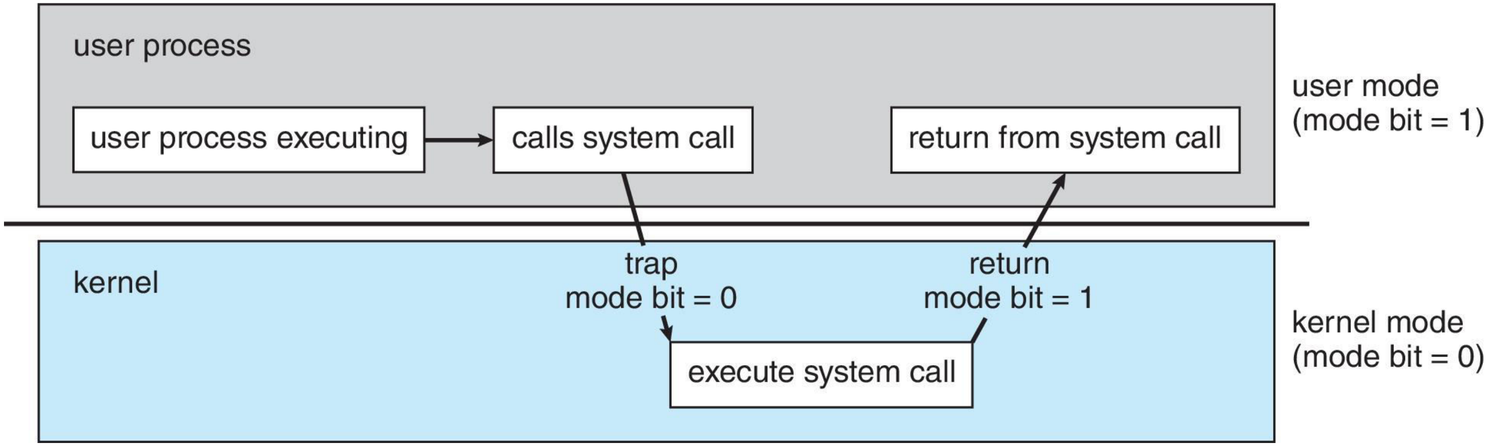
```
#include <stdio.h> #include <stdlib.h> #include <unistd.h>#include<sys/wait.h>
int main (int argc, char *argv[]) {
        pid_t childpid = 0;
        int i, n;
        n = atoi(argv[1]);
        for (i = 1; i < n; i++)
                if (childpid = fork())
                        break;
        fprintf(stderr, "i:%d process ID:%ld parent ID:%ld child ID:%ld\n",
        i, (long)getpid(), (long)childpid, (long)getppid());
        wait(NULL);
        return 0;
}
```

User mode / kernel mode



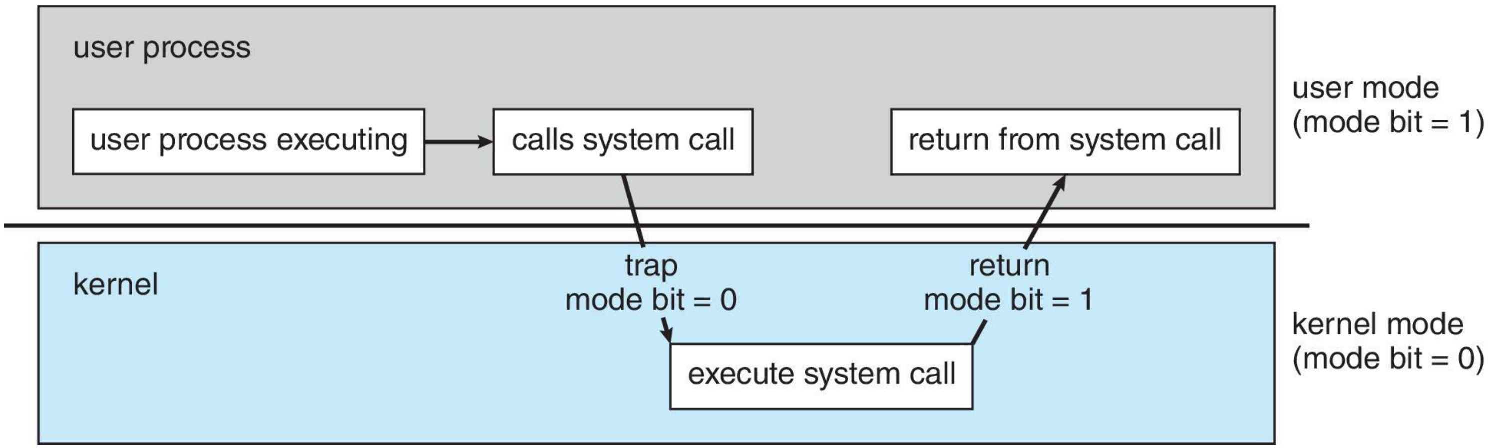
Transition from user to kernel mode

- To obtain services from the operating system, a user program must make a **system call**, which invokes the kernel.
- In order to execute kernel mode instructions the system must switch from user mode to kernel mode and starts running code from the kernel in order to answer the system call



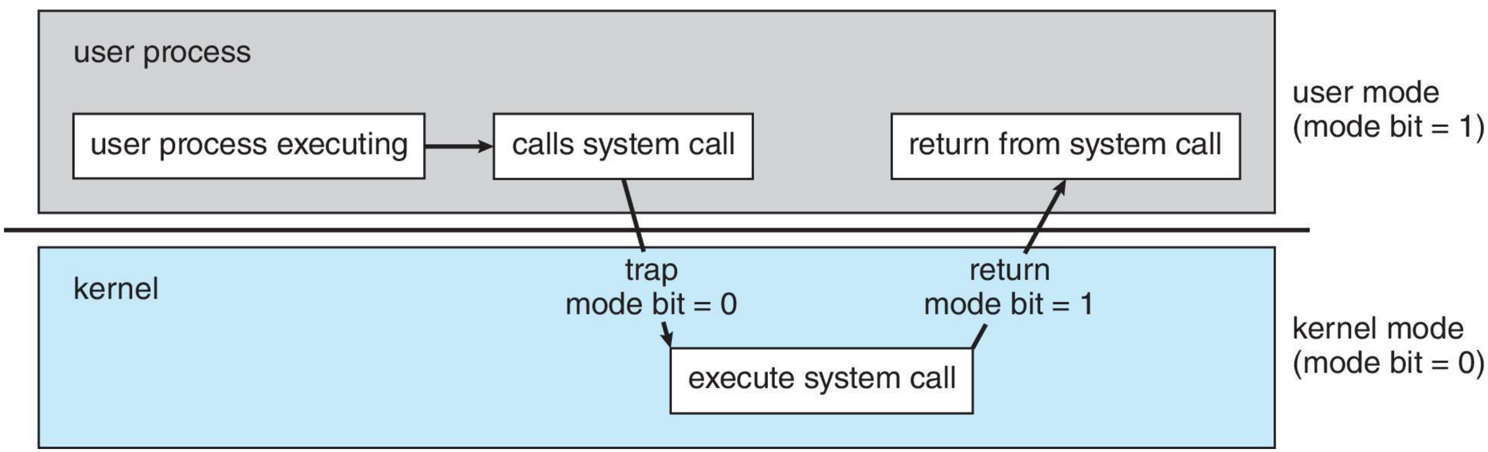
Transition from user to kernel mode

- The system call executes a “trap instruction” which causes a change in the value of the **mode bit**
- The mode bit is stored in a register of the CPU and is used by the system to decide whether a hardware instruction is allowed to execute



Transition from user to kernel mode

- Once the mode bit is set to 0, the kernel instructions related to the service requested by the system call are then executed
- Prior to return to user mode, the mode bit is reset to user mode status

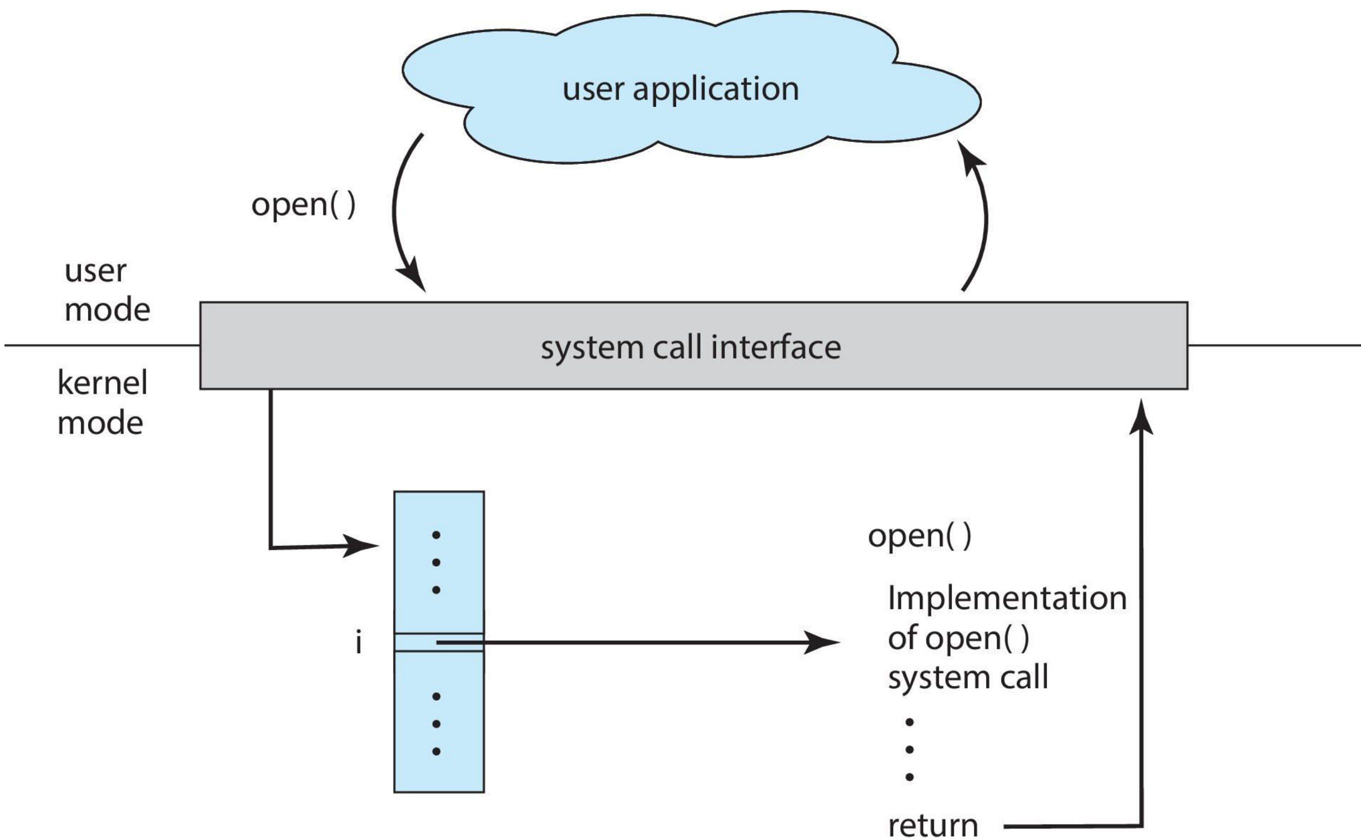


System calls

- **System calls** provide an interface to the services made available by an operating system
- For example, if a user program want to read a file, it cannot read the file directly, it has to ask the OS to do it, through system calls
- Same for opening a file, loading a program, changing directory, executing a program, and many more
- There are many system calls, for example one particular Linux kernel has 393 different system calls:
- getitimer(), getpageside(), getpid(), fork(), open(), close(), read(), reboot(), getcpu(), write()
- You can find the source code of Linux system calls in the Linux kernel source which anyone can download

System Call Implementation

- Typically, a number is associated with each system call
  - **System-call interface** maintains a table indexed according to these numbers



Types of system calls

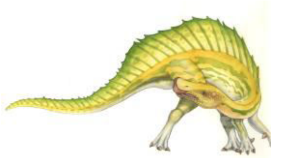
- System calls can be grouped roughly into six major categories: process control, file management, device management, information maintenance, communications, protection.
- Process control
  - create process, terminate process
  - end, abort
  - load, execute
  - get process attributes, set process attributes
  - wait for time
  - wait event, signal event
  - allocate and free memory
  - **Locks** for managing access to shared data between processes

Types of system calls (Cont.)

- File management
  - create file, delete file
  - open, close file
  - read, write, reposition
  - 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

Types of system calls (Cont.)

- 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
  - send, receive messages if **message passing model** to **host name** or **process name**
    - From **client** to **server**
  - **Shared-memory model** create and gain access to memory regions
  - transfer status information
  - attach and detach remote devices



Types of system calls (Cont.)

- 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

Process control	`CreateProcess()`
	`ExitProcess()`
	`WaitForSingleObject()`
		`fork()`
		`exit()`
		`wait()`

File management	`CreateFile()`
	`ReadFile()`
	`WriteFile()`
	`CloseHandle()`
		`open()`
		`read()`
		`write()`
		`close()`

Device management	`SetConsoleMode()`
	`ReadConsole()`
	`WriteConsole()`
		`ioctl()`
		`read()`
		`write()`

Information maintenance	`GetCurrentProcessID()`
	`SetTimer()`
	`Sleep()`
		`getpid()`
		`alarm()`
		`sleep()`

Communications	`CreatePipe()`
	`CreateFileMapping()`
	`MapViewOfFile()`
		`pipe()`
		`shm_open()`
		`mmap()`

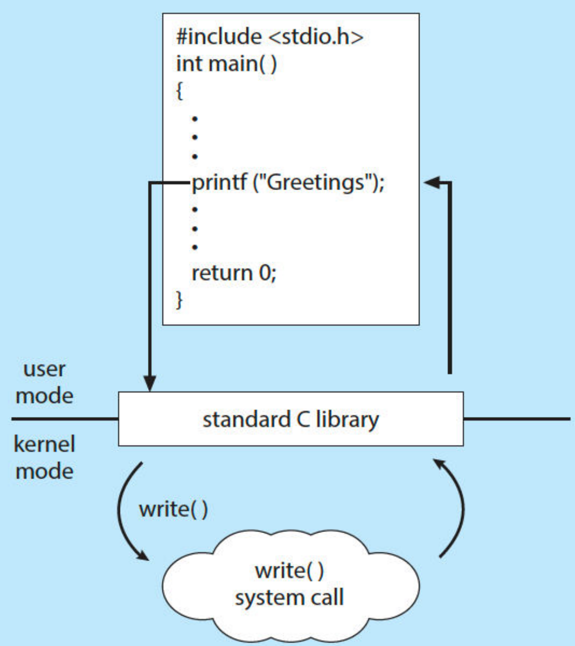
Protection	`SetFileSecurity()`
	`InitlializeSecurityDescriptor()`
	`SetSecurityDescriptorGroup()`
		`chmod()`
		`umask()`
		`chown()`

Standard C Library Example

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

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:



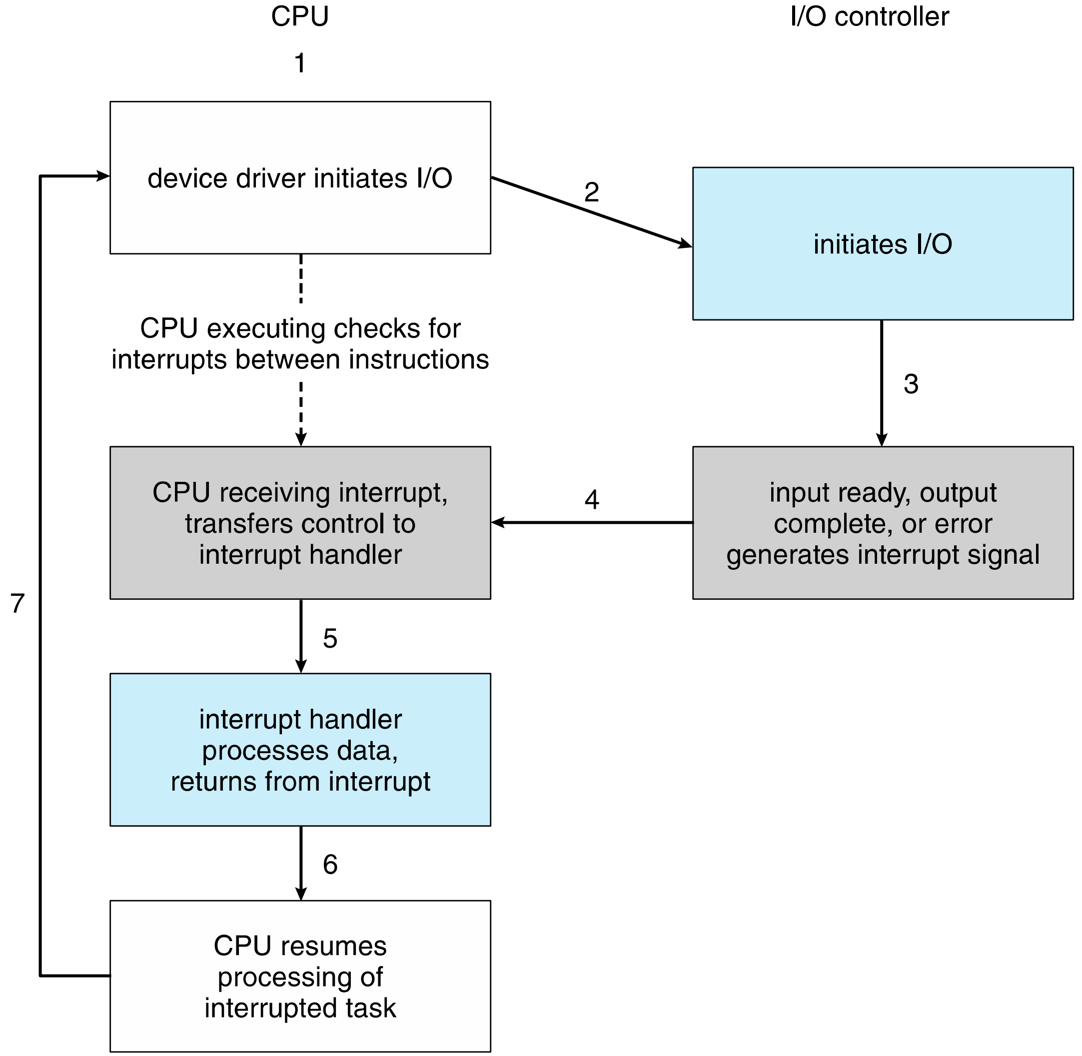
Interrupts

- I/O devices occasionally need to be serviced by the CPU
  - e.g. Inform CPU that a key has been pressed
- These events are asynchronous i.e. we cannot predict when they will happen.
- Need a way for the CPU to determine when a device needs attention
- The CPU is made aware of I/O devices activities through **interrupts** which are signals sent on system bus

Interrupts

- A device controller ***raises*** an interrupt by asserting a signal on the interrupt request line (one of the lines on the system bus)
- After executing each hardware instruction, the CPU checks the interrupt request line if an interrupt has been raised
- Interrupts tell the CPU to stop its current activities and execute the appropriate part of the OS that handles interrupts for a particular device
- An operating system is essentially an **interrupt driven** system, servicing user's tasks until it get interrupted by an external event such as keyboard stroke

Interrupt-drive I/O Cycle



Handling Interrupts

- Different routines handle different interrupts – called Interrupt Service Routines (ISR) (device drivers)
- When CPU is interrupted, it stops what it was doing, a generic routine called Interrupt Handling Routine (IHR) is run
- This code examines the nature of interrupt, through the **interrupt vector**, which contains the addresses of all the service routines
- Which then calls the corresponding Interrupt Service Routine (ISR) (for example code inside a device driver)
- After servicing the interrupt, the CPU resumes the interrupted computation

End of Introduction