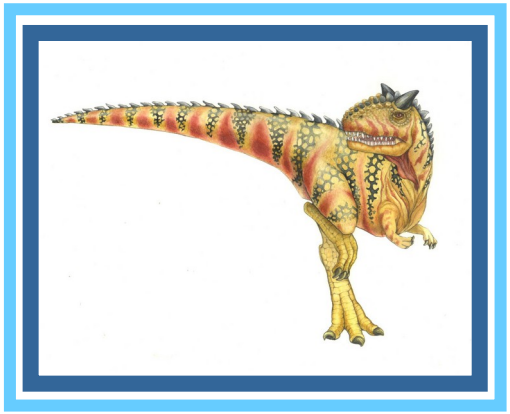
Introduction to Operating Systems



Objectives

- To outline the most common computer system architectures
- To describe operating systems structure
- To describe computer system and operating system operations in terms of interrupts and system calls
- Establish the basis for you to further explore processes and threads

Before getting serious

- Usage share of operating systems on desktop/laptop computers ?

Before getting serious

- Usage share of operating systems on desktop/laptop computers ?

Windows	OS X	Unknown	Linux	Chrome OS	FreeBSD
75.41 %	14.64 %	4.74 %	2.92 %	2.29 %	0.01 %

Source statcounter.com
January 2025

Before getting serious

- Usage share of operating systems on desktop/laptop computers ?

Windows	OS X	Unknown	Linux	Chrome OS	FreeBSD
75.41 %	14.64 %	4.74 %	2.92 %	2.29 %	0.01 %

Source statcounter.com
January 2025

- Usage share of operating systems on mobile computers ?

Before getting serious

- Usage share of operating systems on desktop/laptop computers ?

Windows	OS X	Unknown	Linux	Chrome OS	FreeBSD
75.41 %	14.64 %	4.74 %	2.92 %	2.29 %	0.01 %

Source statcounter.com
January 2025

- Usage share of operating systems on mobile computers ?

Android	iOS	Samsung	Unknown	KaiOS	Linux
73.52 %	26.01 %	0.31 %	0.1 %	0.03 %	0.01 %

Before getting serious

- Usage share of operating systems on desktop/laptop computers ?

Windows	OS X	Unknown	Linux	Chrome OS	FreeBSD
75.41 %	14.64 %	4.74 %	2.92 %	2.29 %	0.01 %

Source statcounter.com
January 2025

- Usage share of operating systems on mobile computers ?

Android	iOS	Samsung	Unknown	KaiOS	Linux
73.52 %	26.01 %	0.31 %	0.1 %	0.03 %	0.01 %

- Usage share of operating systems on web servers ?

Before getting serious

- Usage share of operating systems on desktop/laptop computers ?

Windows	OS X	Unknown	Linux	Chrome OS	FreeBSD
75.41 %	14.64 %	4.74 %	2.92 %	2.29 %	0.01 %

Source statcounter.com
January 2025

- Usage share of operating systems on mobile computers ?

Android	iOS	Samsung	Unknown	KaiOS	Linux
73.52 %	26.01 %	0.31 %	0.1 %	0.03 %	0.01 %

- Usage share of operating systems on web servers ?

Unix/Linux	Windows
88.2 %	12.1 %

Source w3techs.com
January 2023

Before getting serious

- Usage share of operating systems on desktop/laptop computers ?

Windows	OS X	Unknown	Linux	Chrome OS	FreeBSD
75.41 %	14.64 %	4.74 %	2.92 %	2.29 %	0.01 %

Source statcounter.com
January 2025

- Usage share of operating systems on mobile computers ?

Android	iOS	Samsung	Unknown	KaiOS	Linux
73.52 %	26.01 %	0.31 %	0.1 %	0.03 %	0.01 %

- Usage share of operating systems on web servers ?

Unix/Linux	Windows
88.2 %	12.1 %

Source w3techs.com
January 2025

- For a historical perspective on operating systems : https://en.wikipedia.org/wiki/Operating_system

What is an Operating System?

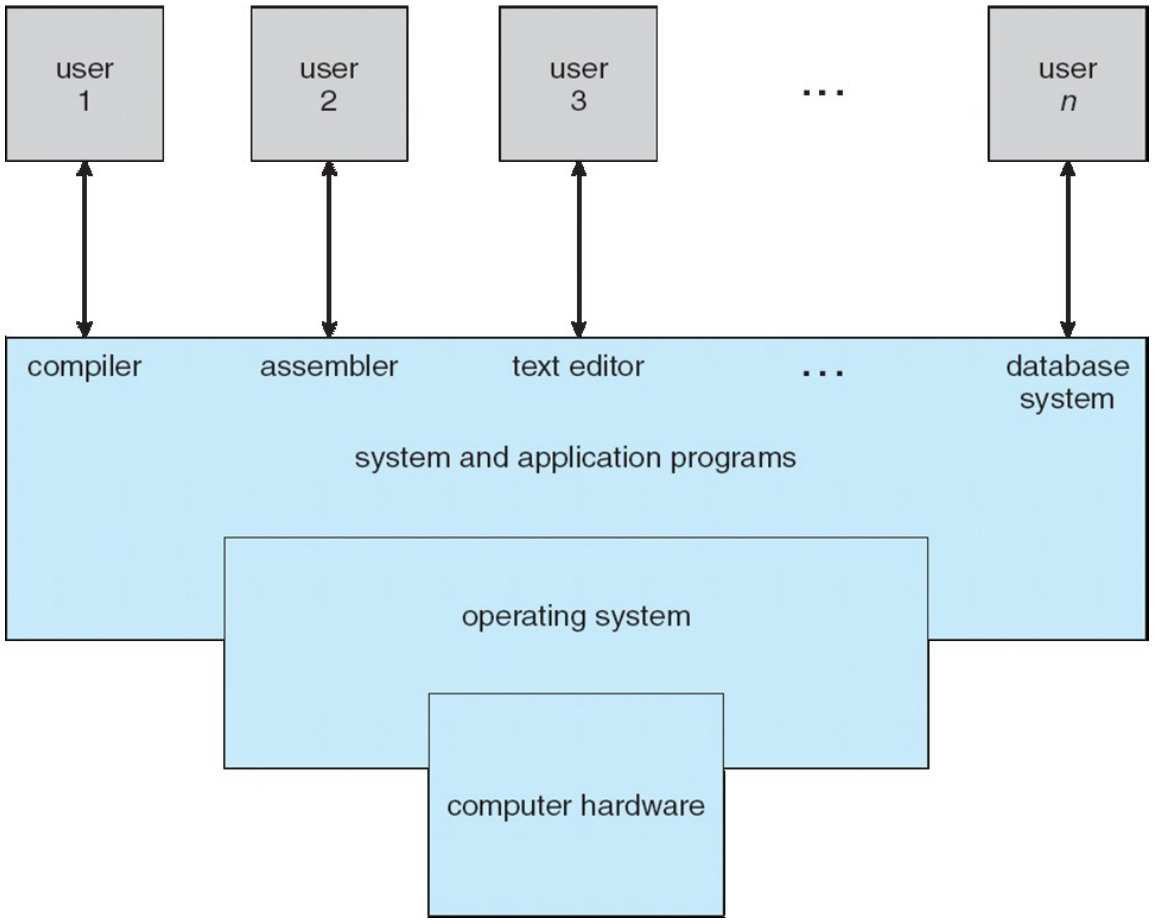
What is an Operating System?

- A program that acts as an **intermediary** between a user of a computer and the computer hardware
- Operating system goals:
  - Execute user programs and make solving user problems easier
  - Make the computer system **convenient** to use
  - Use the computer hardware in an **efficient** manner

Computer System Components

- Computer systems can be divided into four components:
  - **Hardware** – provides basic computing resources
    - CPU, memory, I/O devices
  - **Operating system**
    - Controls and coordinates use of hardware among various applications and users
  - **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
  - **Users**
    - People, machines, other computers

Four Components of a Computer System



Operating System Definition

- Depends on the point of view !
- **User view:** convenience, ease of use and good performance
  - Don’t care about **resource utilization**
- But shared computers such as data servers must keep all users happy !
- Hand-held computers (mobile phones, tablets etc.) are resource poor, optimized for usability and battery life
- Some computers have little or no user interface, such as embedded computers in devices and automobiles

Operating System Definition

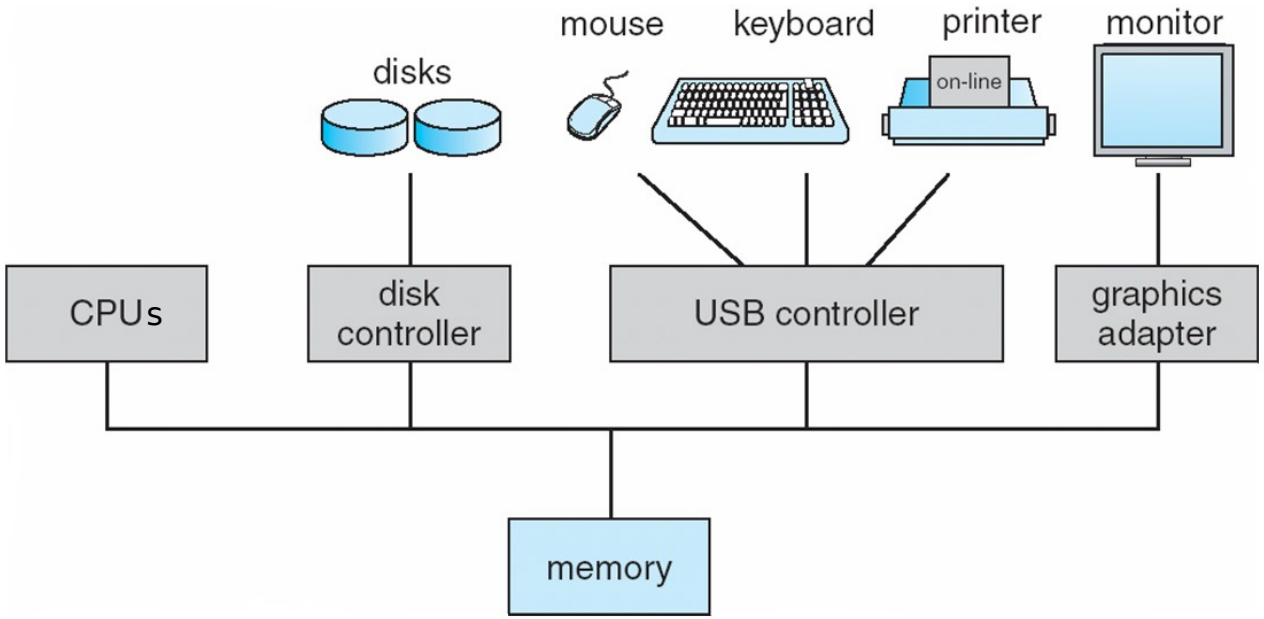
- **System view :**
- OS is a **resource allocator**
  - Manages all resources
  - Decides between conflicting requests for efficient and fair resource use
- OS is a **control program**
  - Controls execution of programs to prevent errors and improper use of the computer

Operating System Definition

- No universally accepted definition ...
- "*Everything a vendor ships when you order an operating system*" is a good approximation
- The one program running at all times on the computer is the **kernel**
- Everything else is either
  - a system program (ships with the operating system) , or
  - an application program

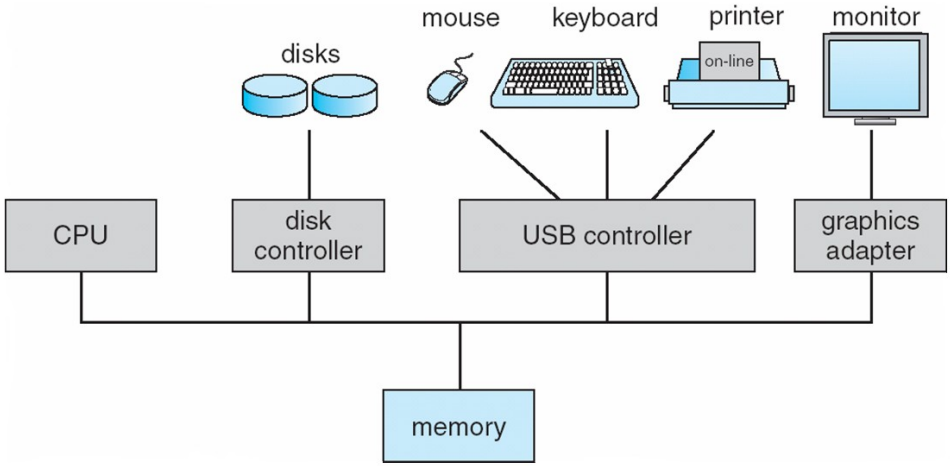
Computer System Organization

- Computer-system operation
  - 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



Computer System Operation

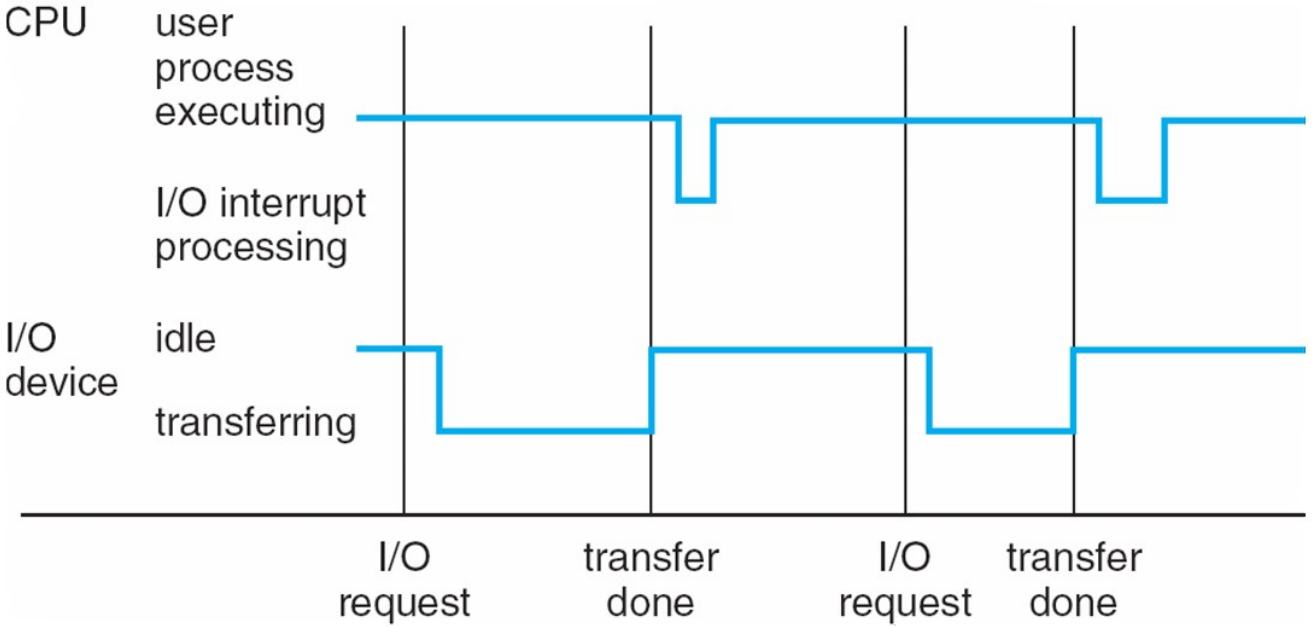
- I/O devices and the CPU can execute concurrently
- Each **device controller** is in charge of a particular device type
- Each device controller has a **local buffer**
- CPU moves data from/to main memory to/from local buffers
- I/O is from the device to local buffer of controller
- Device controller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



Interrupts

- Most hardware, therefore most operating systems are **interrupt driven**
- CPU execution of the current program is interrupted
- Interrupt architecture must save the address of the interrupted instruction and the state of the CPU
  - **Analogy :** putting a bookmark in a book you're reading when receiving a call
  - Interrupt transfers control to the interrupt service routine generally, through the interrupt vector, which contains the addresses of all the service routines
- A **trap** or **exception** is a **software-generated** interrupt caused either by an error or a user request by **system calls**

Interrupt Timeline



Timeline of a user process performing I/0

Interrupts

- What if there's no interrupts ? i.e. no possibility to interrupt a CPU ? Can you think of an alternative solution to service I/O ? What are the implications ?

I/O Structure

- **Synchronous, blocking I/O**: after I/O starts, control returns to user program **only upon I/O completion**
  - CPU idle until the next interrupt, at most one I/O request at a time
- **Asynchronous, non-blocking I/O**: after I/O starts, control returns to user program **immediately without waiting for I/O completion**
  - User program needs to check and retrieve data later
- Concepts of synchronous and asynchronous transfer also apply to inter-process communication (via message passing, sockets, etc.) and not just file or network I/O !

I/O Structure

- The OS has a **device driver** for each device controller
  - provides uniform interface between kernel and the controller
- I/O steps:
  - device driver loads controller registers
  - device controller performs I/O to/from device buffer
  - upon completion, device controller notifies driver via an interrupt
  - device driver returns control and data to the OS

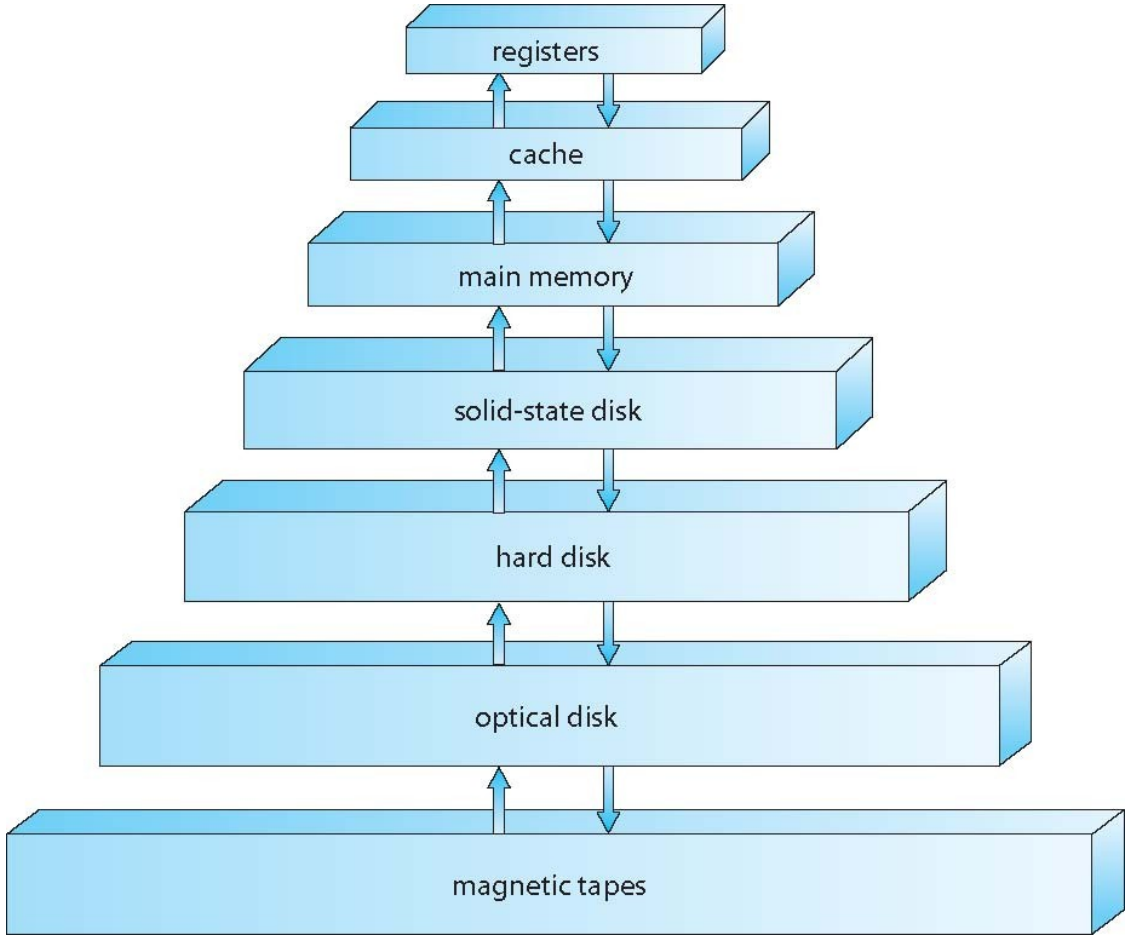
Direct Memory Access

- 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

Storage Structure

- Main memory – only large storage media that the CPU can access directly
  - **Random access**
  - Typically **volatile**
- Secondary storage – extension of main memory that provides large **nonvolatile** storage capacity
- Hard disks – rigid metal or glass platters covered with magnetic recording material
  - Disk surface is logically divided into tracks, which are subdivided into sectors
  - The disk controller determines the logical interaction between the device and the computer
- Solid-state disks – faster than hard disks, nonvolatile
  - Various technologies
  - Becoming more popular

Storage Device Hierarchy



speed – cost - volatility

Caching

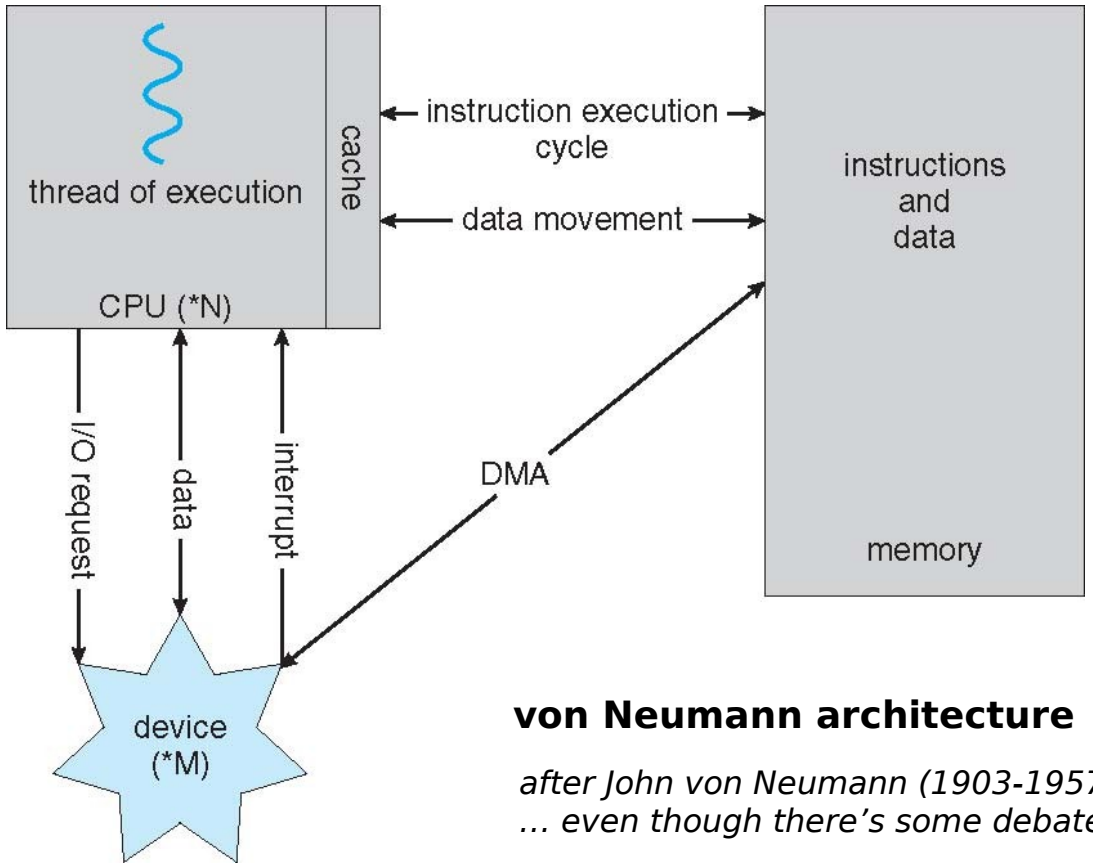
- Important principle, performed at many levels in a computer (in hardware, operating system, software)
- Information in use copied from slower to faster storage temporarily
  - Multiple copies of a datum can exist !
- Faster storage (cache) checked first to determine if information is there
  - If it is, information used directly from the cache (fast)
  - If not, data copied to cache and used there
- Cache smaller than storage being cached
  - Cache management is an important design problem
  - Cache size and replacement policy

Performance of Various Levels 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

How a Modern Computer Works



von Neumann architecture

after John von Neumann (1903-1957)
... even though there's some debate on other contributors

Computer System Architecture

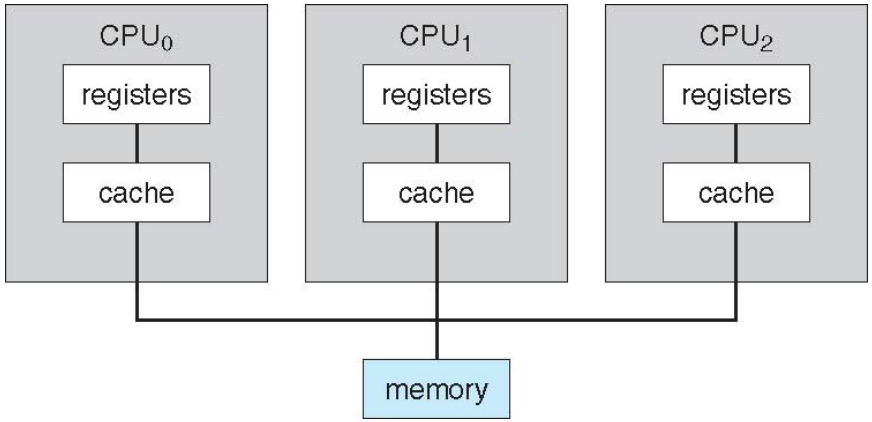
- Most systems use a single general-purpose processor
  - Not all computer systems are desktop or mobile computers !
  - Many systems have special-purpose processors as well
- **Multiprocessor** systems growing in use and importance
  - Also known as **parallel systems**
  - What are the advantages of having multiple processors ?

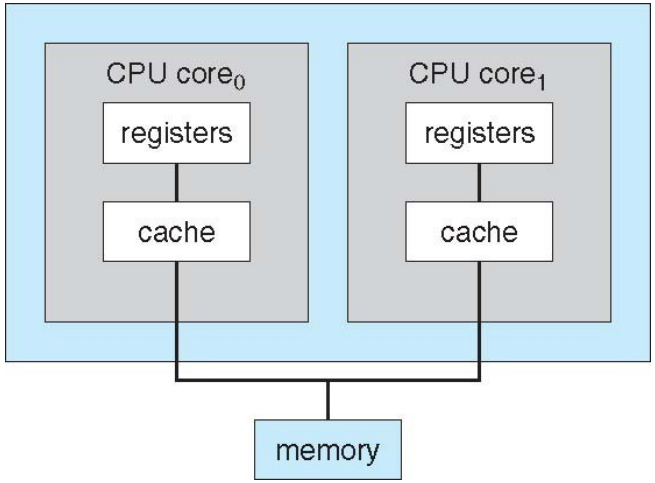
Computer System Architecture

- Most systems use a single general-purpose processor
  - Not all computer systems are desktop or mobile computers !
  - Many systems have special-purpose processors as well
- **Multiprocessor** systems growing in use and importance
  - Also known as **parallel systems**
  - Advantages include:
    1. **Increased throughput**
    2. **Economy of scale**
    3. **Increased reliability** – graceful degradation or fault tolerance
  - Two strategies :
    1. **Asymmetric Multiprocessing** – each processor is assigned a specific type of task.
    2. **Symmetric Multiprocessing** – each processor performs any type of task

Multi-Core Design

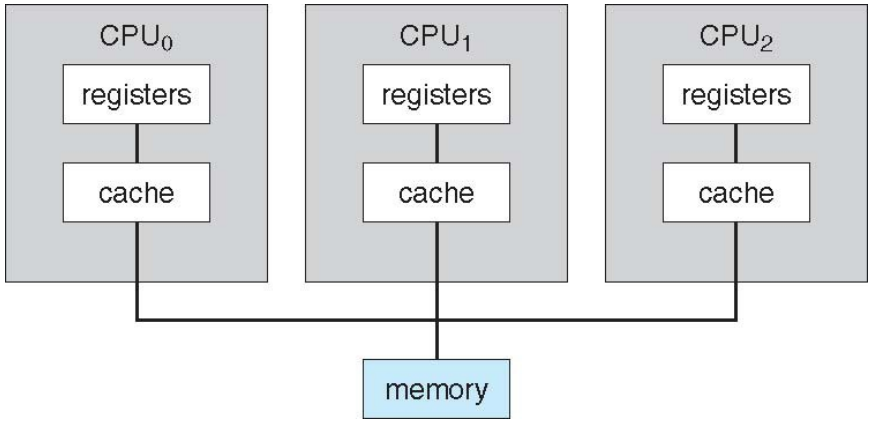
- **Multi-core** vs multi-chip (a CPU by chip)
- What are the advantages of a multi-core design ?

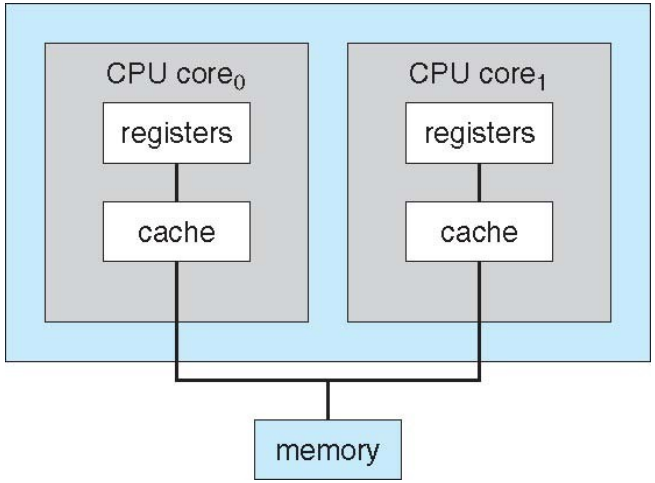




Multi-Core Design

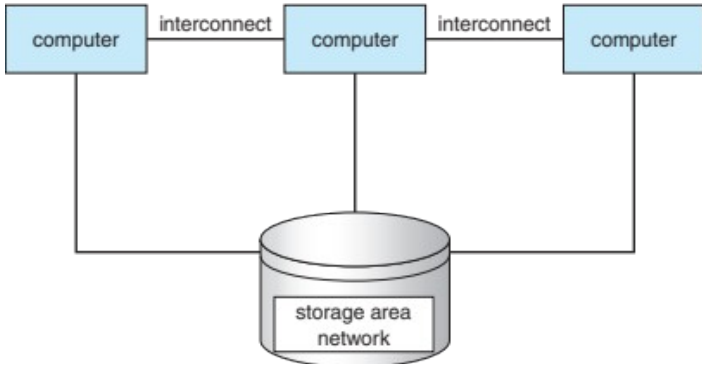
- **Multi-core** vs multi-chip (a CPU by chip)
- What are the advantages of a multi-core design ?
  - Faster CPU to CPU communication
  - Consume less power





Clustered Systems

- Like multiprocessor systems, but multiple systems working together
  - Usually sharing storage via a **storage-area network (SAN)**
  - Provides a **high-availability** service which survives failures
    - **Asymmetric clustering** has one machine in hot-standby mode
    - **Symmetric clustering** has multiple nodes running applications, monitoring each other
  - Some clusters are for **high-performance computing (HPC)**
    - Applications must be written to use **parallelization**
  - Some have **distributed lock manager** (**DLM**) to avoid conflicting operations



Operating System Structure

- A single user/program cannot hold the CPU and I/O devices all the time!
- A **job pool** keeps all programs awaiting to be run, a subset of it is loaded in memory
  - A program loaded and executing in memory is called a **process**

- **Multi-programming (batch system) :**
  - CPU always has one job to execute
  - One job selected and run via **job scheduling**
  - When it has to wait (for I/O for example), OS switches to another job

- **Time-sharing** (**multi-tasking**) :
  - CPU switches jobs frequently
  - Many jobs get to progress, users get more responsive execution
  - If several jobs ready to run at the same time, perform **scheduling** to select

Operating System Operations

- Process management ✓
- Memory management
- Storage management
- I/O system management

Process Management

- A process is a program in execution. It is a unit of work within the system. Program is a **passive entity**, process is an **active entity**
- Process needs resources to accomplish its task
  - CPU, memory, I/O, files
  - Initialization data
- Process termination requires reclaim of any reusable resources
- Single-threaded process has one **program counter** specifying location of next instruction to execute
  - Process executes instructions sequentially, one at a time, until completion
- Multi-threaded process has **one program counter per thread**
- Typically system has many processes, some user, some operating system running concurrently on one or more CPUs
  - Concurrency by multiplexing the CPUs among the processes

Process Management Activities

The operating system is responsible for the following activities in connection with process management:

- Creating and deleting both user and system processes
- Suspending and resuming processes
- Providing mechanisms for process synchronization
- Providing mechanisms for process communication
- etc.

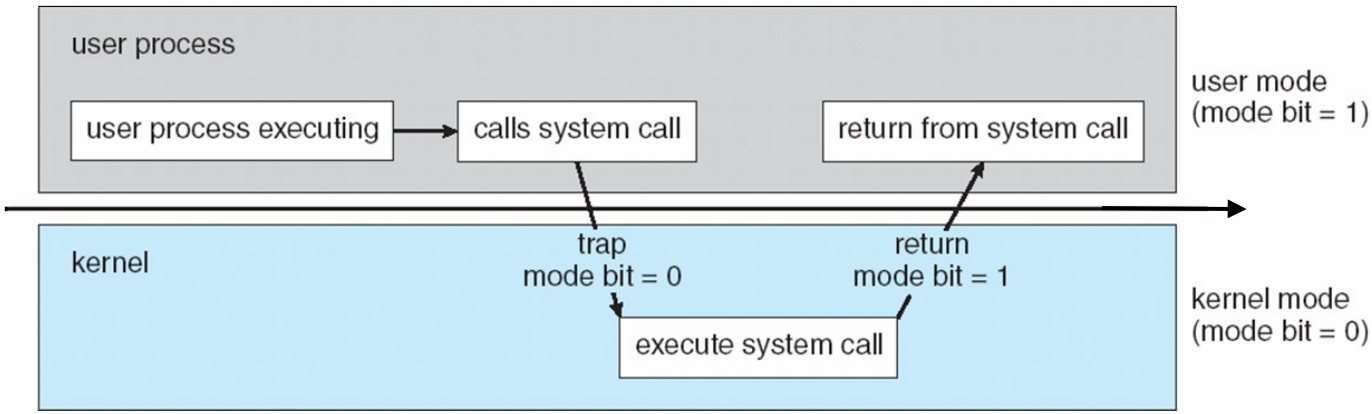
All of which involve one more more **system calls**.

Back to software interrupts

- **Interrupt driven** (hardware and software)
  - Hardware interrupt : one of device controllers
  - Software interrupt (**exception** or **trap**)
    - Software error, e.g., division by zero, unauthorized memory access etc.
    - Request for operating system service, a **system call**
- What happens during a system call ?

Back to software interrupts

- **Dual-mode** operation allows OS to protect itself and other system components
  - **User mode** and **kernel mode,** set by a **mode bit** in CPU:
    - Provides ability to distinguish when system is running user code or kernel code
    - Some instructions designated as **privileged**, only executable in kernel mode, e.g. I/O control, timer and interrupt management etc.
    - System call changes mode to kernel, return from call resets it to user
- Increasingly CPUs support multi-mode operations
  - i.e. virtual machine manager (VMM) mode for guest VMs



System Calls : Closer Look

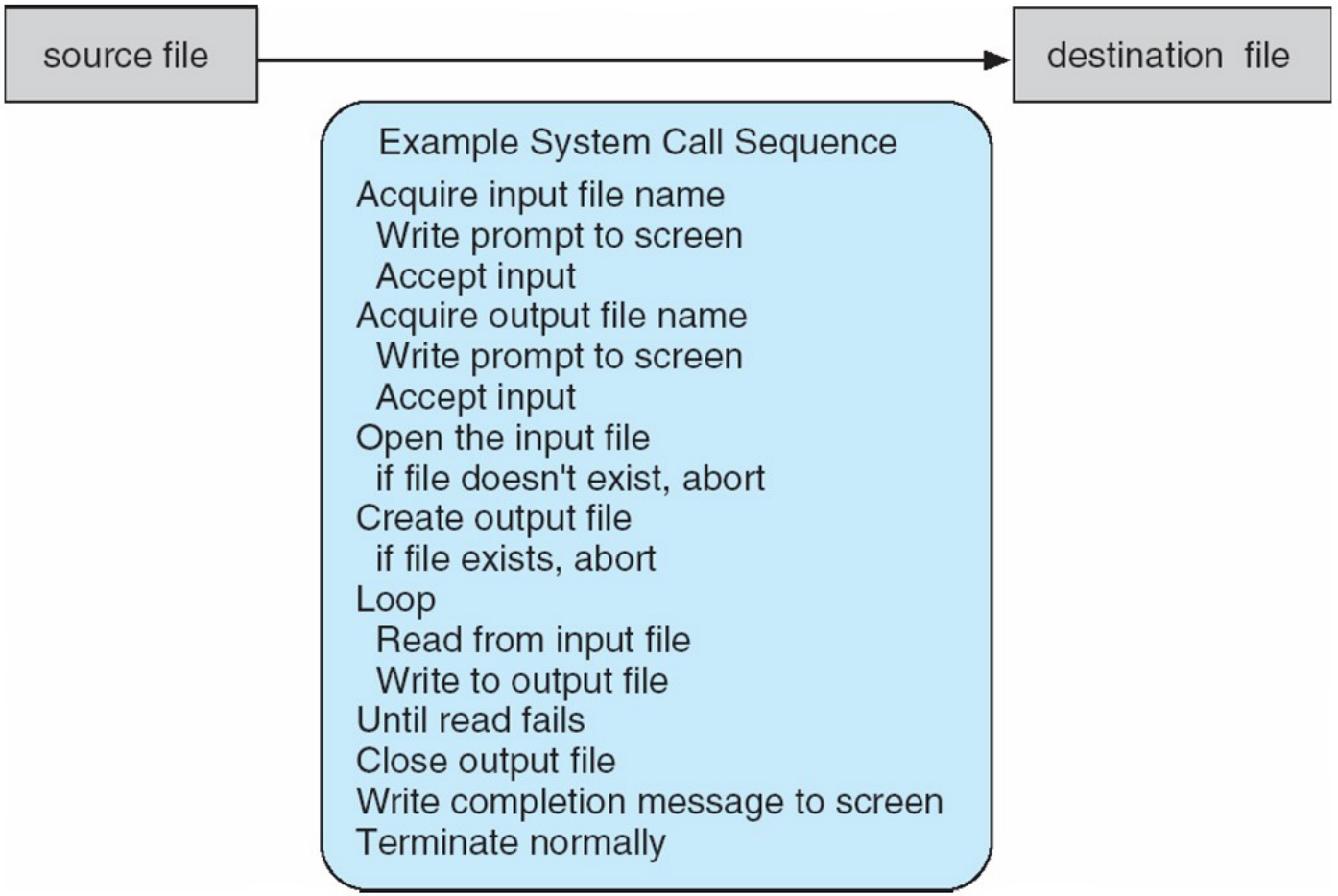
- Programming interface to the services provided by the OS
- Typically written in a systems programming language (C or C++)
- Accessed by programs via an **Application Programming Interface (API)**
- Common APIs: Win32 API for Windows, POSIX API for POSIX-based systems (including virtually all versions of UNIX, Linux, and Mac OS X), Java API for the Java virtual machine (JVM)
- Higher-level languages, like Python, build high-level APIs around native system API

Example of System Calls

- Simple program : copy the contents of one file to another file.
- Can you think of the sequence of system calls ?

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:

```
#include <unistd.h>

ssize_t    read(int fd, void *buf, size_t count)
```

return value — function name — parameters

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

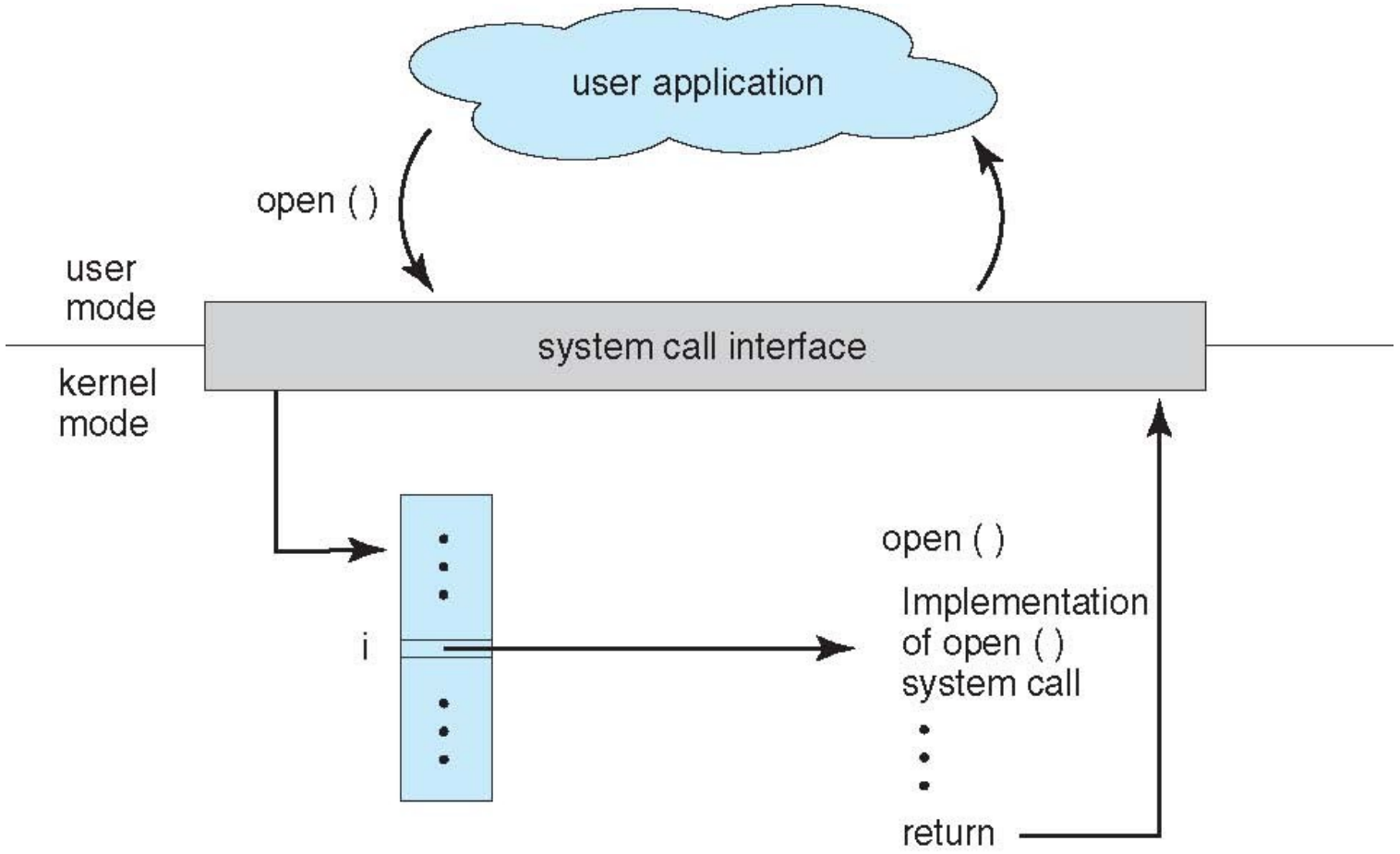
- `int fd`—the file descriptor to be read
- `void *buf`—a buffer where the data will be read into
- `size_t count`—the maximum number of bytes to be read into the buffer

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

System Call Implementation

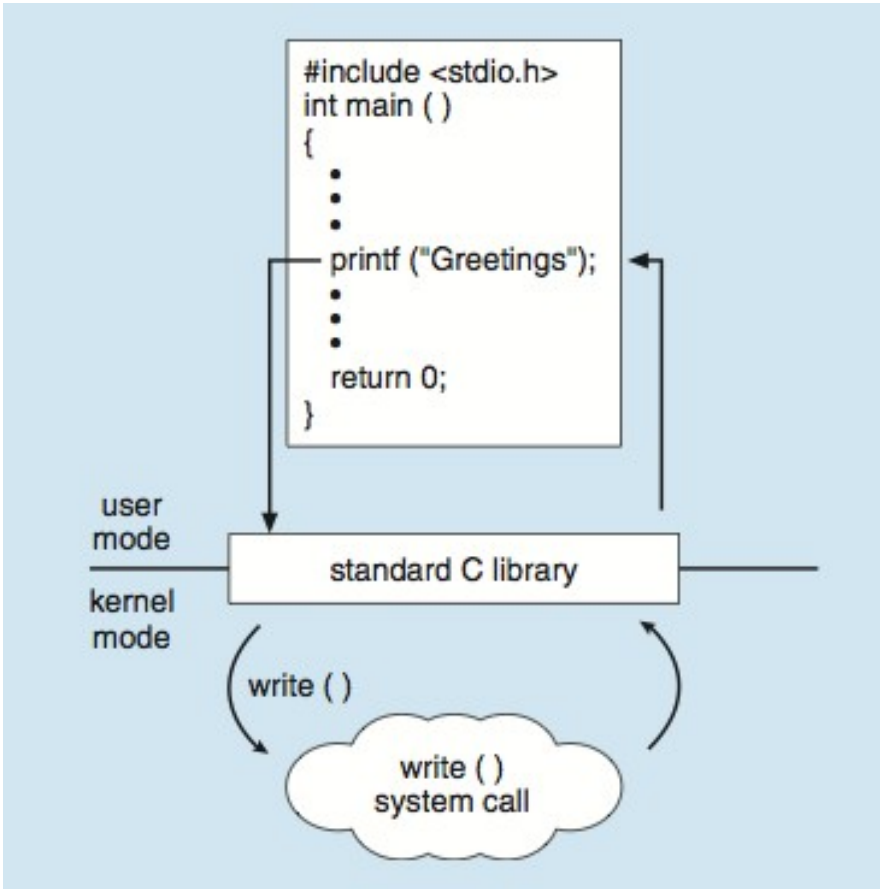
- Typically, a number associated with each system call
  - **System-call interface** maintains a table indexed according to these numbers
- The system call interface invokes the intended system call in OS kernel and returns status of the system call and any return values
- The caller need know nothing 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)

API – System Call – OS Relationship



Standard C Library Example

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



Types of System Calls

- 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
  - dump memory if error
  - debugger for determining bugs, single step execution
  - **Locks** for managing access to shared data between processes
  - etc.

Types of System Calls

- Communications
  - create, delete communication connection
  - send, receive messages if **message passing model** to host name or process name, or from client to server
  - **shared-memory model** create and gain access to memory regions, deallocate memory regions
  - etc.

Types of System Calls

- File management
- Device management
- Protection
- Etc.

Examples of Windows and Unix System Calls

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

Computing Environments at a Glance

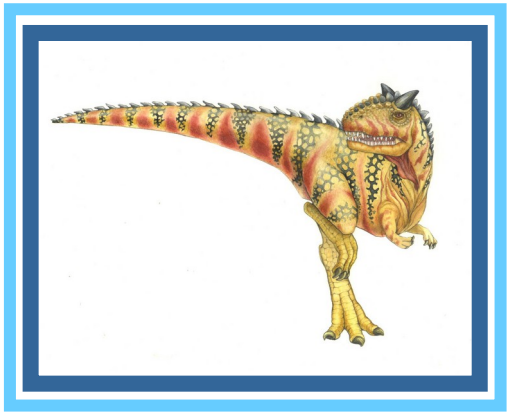
- **Traditional**: stand-alone general purpose machines traditionally
  - Not anymore, interconnections via the Internet, wireless networks, web terminals etc.
- **Mobile**: smartphones, tablets – mobile operating system
- **Distributed computing**: separate, heterogeneous systems networked together on a TCP/IP network – network operating system
- **Client-server computing**
- **Peer-to-peer computing** - protocols
- **Cloud computing**
- **Real-time computing**
- **Virtualization**

Concepts and tools encountered in this course apply to all of them !

Free and Open-Source Operating Systems

- Unix-like operating systems made available in source-code format rather than just binary closed-source
- Allows programmers to create, study, test and fix all kinds of software, including new operating systems!
- Two main families:
  - **GNU/Linux-based :** e.g. Debian, RedHat, Fedora etc.
    GNU Public License (GPL)
  - **BSD UNIX-based:** FreeBSD, OpenBSD, TrueOS etc.
    BSD License
- Have a look on DistroWatch https://distrowatch.com/
- Can use VMM like VMware Player (Free on Windows), VirtualBox (open source and free on many platforms)
  - Use to run guest operating systems for exploration

Alright !
Questions ?



Exercises

- What is the purpose of interrupts? What are the differences between a trap and an interrupt? Can traps be generated intentionally by a user program? If so, for what purpose?
- What's the purpose of system calls?
- Mention 2 advantages of multi-processor systems.
- What do you call a program that's been loaded and is executing?
- Which part of the operating system decides which job will run?
- Mention 2 activities the operating system is responsible for in connection with process management.