Operating Systems
Introduction

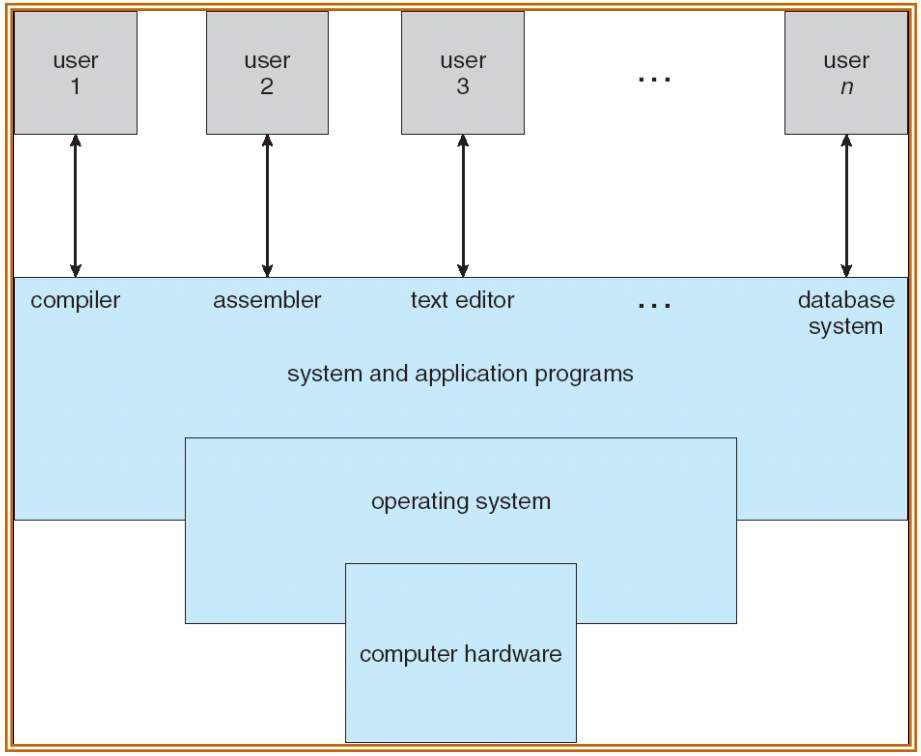
Computing Devices Everywhere

How do we tame complexity?

- Every piece of computer hardware different
  - Different CPU
    - Pentium, PowerPC, MIPS, …
  - Different amounts of memory, disk, …
  - Different types of devices
    - Mice, Keyboards, Sensors, Cameras, Fingerprint readers
  - Different networking environment
    - Cable, DSL, Wireless, Firewalls,…

- Questions:
  - Does the programmer need to write a single program that performs many independent activities?
  - Does every program have to be altered for every piece of hardware?
  - Does a faulty program crash everything?
  - Does every program have access to all hardware?

Four Components of a Computer System



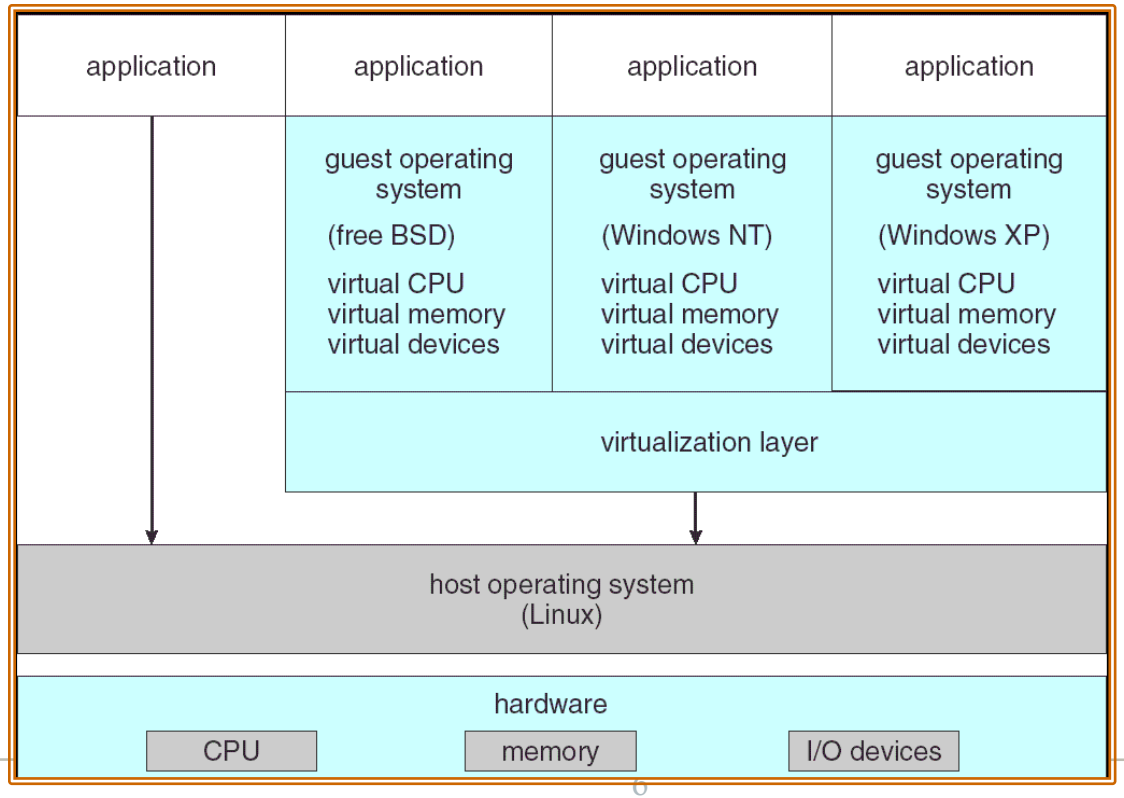
- An operating system is the interface between the user and the hardware.
- Definition: An operating system implements a virtual machine that is (hopefully) **easier and safer to program and use** than the raw hardware.

OS as Virtual Machines

- Software emulation of an abstract machine
  - Make it look like hardware has features you want
  - Programs from one hardware & OS work on another one
- Fault Isolation
  - Processes unable to directly impact other processes
  - Bugs cannot crash whole machine
- Protection and Portability
- Programming simplicity
  - Each process thinks it has all memory/CPU time
  - Each process thinks it owns all devices
  - Different Devices appear to have same interface
  - Device Interfaces more powerful than raw hardware
    - Bitmapped display ⇒ windowing system
    - Ethernet card ⇒ reliable, ordered, networking (TCP/IP)

Virtual Machines: Layers of OSs

- Useful for OS development
  - When OS crashes, restricted to one VM
  - Can aid testing programs on other OSs



Operating System: System View

- OS is a **resource allocator**
  - Manages all resources, including file system, virtual memory, networking, CPU etc.
  - 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
- The OS design goal is to make the machine convenient to use (a software engineering problem) and efficient (a system and engineering problem).

Operating System Definition

- No universally accepted definition
- "Everything a vendor ships when you order an operating system" is good approximation
  - But varies wildly
- "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**

System Boot

- Operating system must be made available to hardware so hardware can start it
  - When power initialized on system, execution starts at a fixed memory location
  - Small piece of code – **bootstrap loader**, stored in Firmware **ROM** locates the kernel, loads it into memory, and starts it
  - Sometimes two-step process where **boot block** at fixed location loaded by ROM code, which loads bootstrap program from disk
    - Common bootstrap loader, **GRUB**, allows selection of kernel from multiple disks, versions, kernel options
- Kernel loads and system is then **running**

Why Study Operating Systems?

- The operating system makes the computer work - it is a key abstraction layer for applications
  - How to get the OS to give users an illusion of infinite memory, CPUs, resources, world wide computing, etc.
- System Design: How to make tradeoffs between
  - performance and the convenience of OS abstractions,
  - performance and the simplicity of OS design, and
  - putting functionality in hardware or software.
- As systems change the OS must adapt (e.g., new hardware, software).

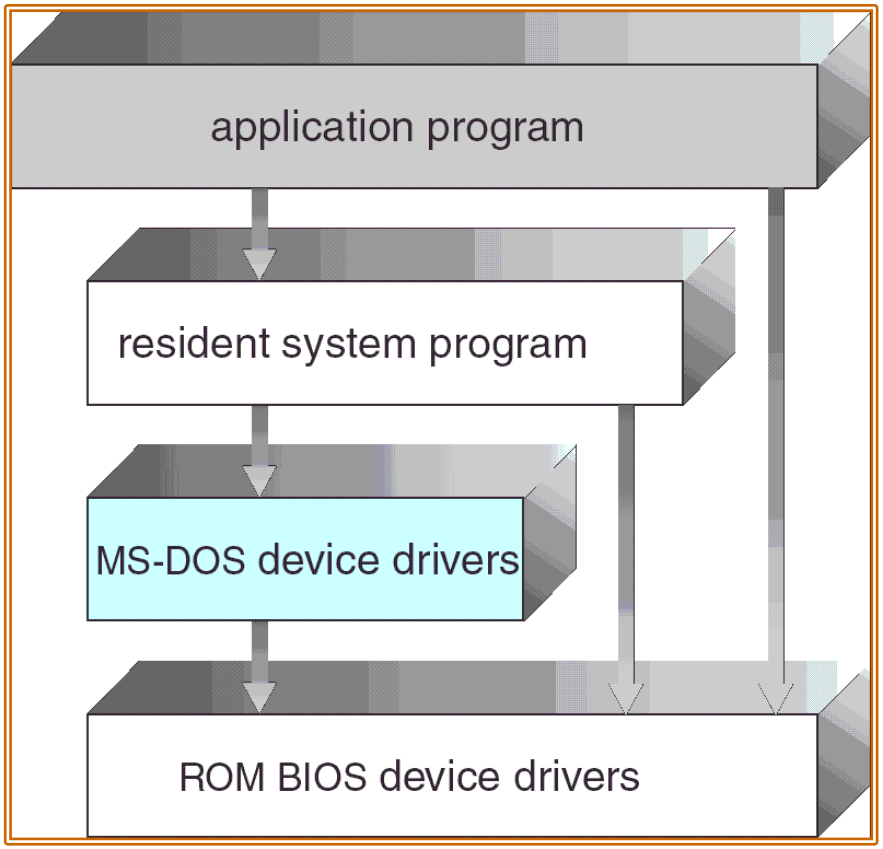
What if we didn't have an Operating System?

- Source Code ⇒ Compiler ⇒ Object Code ⇒ Hardware
- How do you get object code onto the hardware?
- How do you print out the answer?
- Once upon a time, had to toggle in program in binary and read out answer from LED's!

Simple OS: What if only one application?

- Examples:
  - Very early computers
  - Early PCs
  - Embedded controllers (elevators, cars, etc)
- OS becomes just a library of standard services
  - Standard device drivers
  - Interrupt handlers
  - Math libraries

MS-DOS Layer Structure



More complex OS: Multiple Apps

- Full Coordination and Protection
  - Manage interactions between different users
  - Multiple programs running simultaneously
  - Multiplex and protect Hardware Resources
    - CPU, Memory, I/O devices like disks, printers, etc
- Facilitator
  - Still provides Standard libraries, facilities
- Next we look a little at the history of OS

Moore's Law Change Drives OS Change

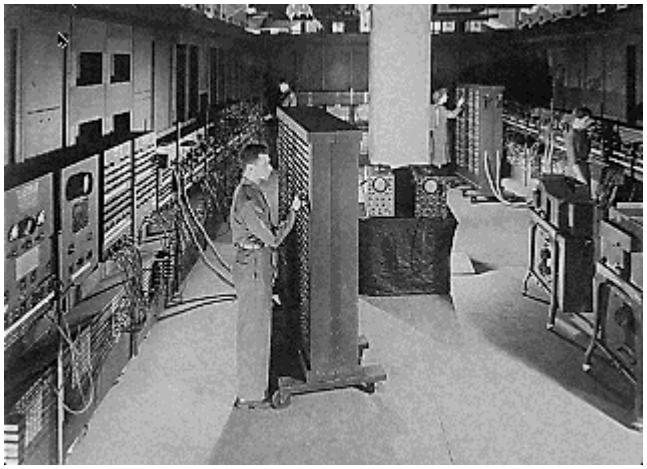
	1981	2006	Factor
CPU MHz, Cycles/inst	10 3—10	3200x4 0.25—0.5	1,280 6—40
DRAM capacity	128KB	4GB	32,768
Disk capacity	10MB	1TB	100,000
Net bandwidth	9600 b/s	1 Gb/s	110,000
# addr bits	16	32	2
#users/machine	10s	≤ 1	≤ 0.1
Price	$25,000	$4,000	0.2

Typical academic computer 1981 vs. 2006

Moore's law effects

- Nothing like this in any other area of business
- Transportation in over 200 years:
  - 2 orders of magnitude from horseback @10mph to Concorde @1000mph
  - Computers do this every decade!
- What does this mean for us?
  - Techniques have to vary over time to adapt to changing tradeoffs
- I place a lot more emphasis on principles
  - The key concepts underlying computer systems
  - Less emphasis on facts that are likely to change over the next few years…
- Let's examine the way changes in $/MIPS has radically changed how OS's work

Dawn of time
ENIAC: (1945—1955)



- "The machine designed by Drs. Eckert and Mauchly was a monstrosity. When it was finished, the ENIAC filled an entire room, weighed thirty tons, and consumed two hundred kilowatts of power."

- http://ei.cs.vt.edu/~history/ENIAC.Richey.HTML

History Phase 1 (1948—1970) Hardware Expensive, Humans Cheap

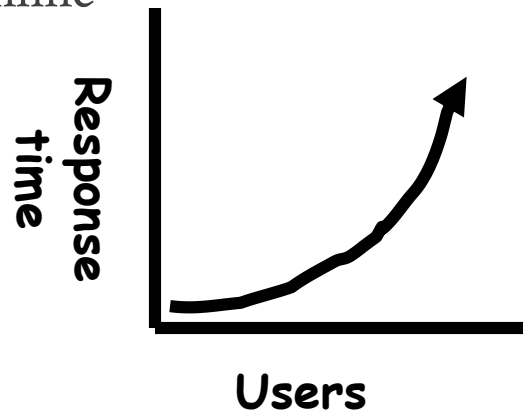
- When computers cost millions of $'s, optimize for more efficient use of the hardware!
  - Lack of interaction between user and computer
- User at console: one user at a time
- Batch monitor: load program, run, print
- Optimize to better use hardware
  - When user thinking at console, computer idle ⇒ BAD!
  - Feed computer batches and make users wait
- *No protection:* what if batch program has bugs?

History Phase 1½ (late 60s/early 70s)

- Data channels, Interrupts: overlap I/O and compute
  - DMA – Direct Memory Access for I/O devices
  - I/O can be completed asynchronously
- Multiprogramming: several programs run simultaneously
  - Small jobs not delayed by large jobs
  - More overlap between I/O and CPU
  - Need memory protection between programs and/or OS
- Complexity gets out of hand:
  - Multics: announced in 1963, ran in 1969
    - 1777 people "contributed to Multics" (30-40 core dev)
    - Turing award lecture from Fernando Corbató (key researcher): "On building systems that will fail"
  - OS 360: released with 1000 known bugs (APARs)
    - "Anomalous Program Activity Report"
- OS finally becomes an important science:
  - How to deal with complexity???
  - UNIX based on Multics, but vastly simplified

History Phase 2 (1970 – 1985) Hardware Cheaper, Humans Expensive

- Computers available for tens of thousands of dollars instead of millions
- OS Technology maturing/stabilizing
- *Interactive* timesharing:
  - Use cheap terminals (~$1000) to let multiple users interact with the system at the same time
  - Sacrifice CPU time to get better response time
  - Users do debugging, editing, and email online
- Problem: Thrashing
  - Performance very non-linear response with load
  - Thrashing caused by many factors including
    - Swapping, queuing



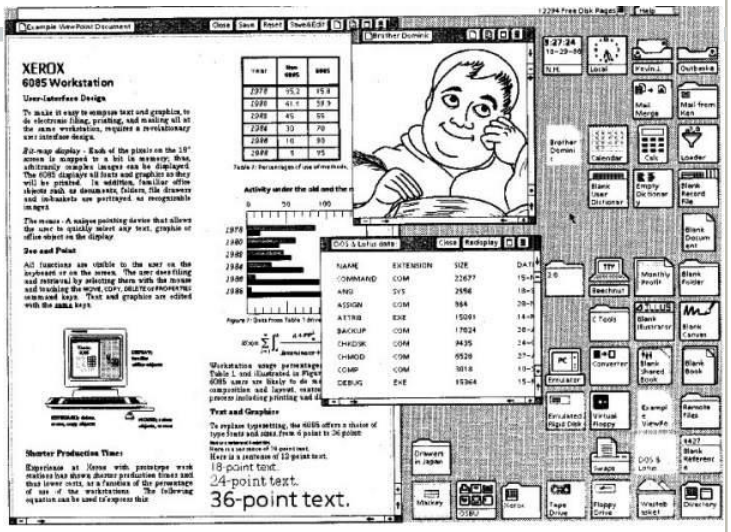
History Phase 3 (1981—)
Hardware Very Cheap, Humans Very Expensive

- Computer costs $1K, Programmer costs $100K/year
  - If you can make someone 1% more efficient by giving them a computer, it's worth it!
  - Use computers to make people more efficient
- Personal computing:
  - Computers cheap, so give everyone a PC
- Limited Hardware Resources Initially:
  - OS becomes a subroutine library
  - One application at a time (MSDOS, CP/M, …)
- Eventually PCs become powerful:
  - OS regains all the complexity of a "big" OS
  - multiprogramming, memory protection, etc (Windows NT,OS/2)
- Question: As hardware gets cheaper does need for OS go away?

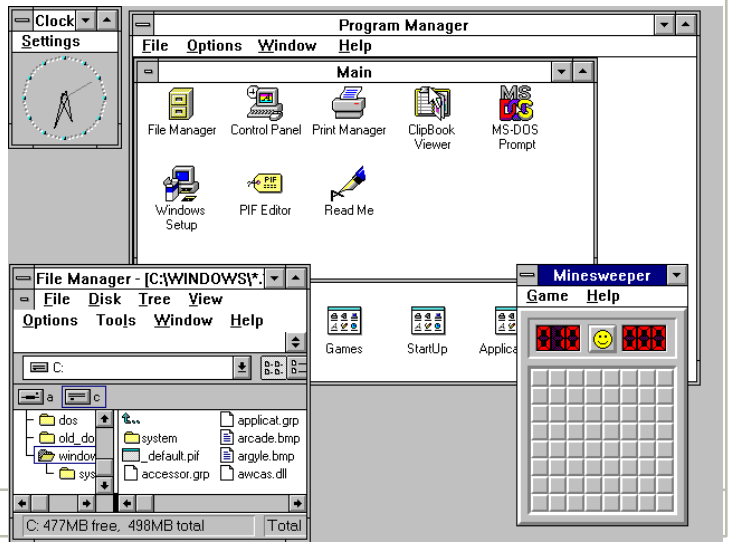
History Phase 3 (con't)
Graphical User Interfaces

- Xerox Star: 1981
  - Originally a research project (Alto)
  - First "mice", "windows"
- Apple Lisa/Machintosh: 1984
  - "Look and Feel" suit 1988
- Microsoft Windows:
  - Win 1.0 (1985)
  - Win 3.1 (1990)
  - Win 95 (1995)
  - Win NT (1993)
  - Win 2000 (2000)
  - Win XP (2001)
  - Win Vista (2007)
  - Win 7 (2009)

Xerox Star

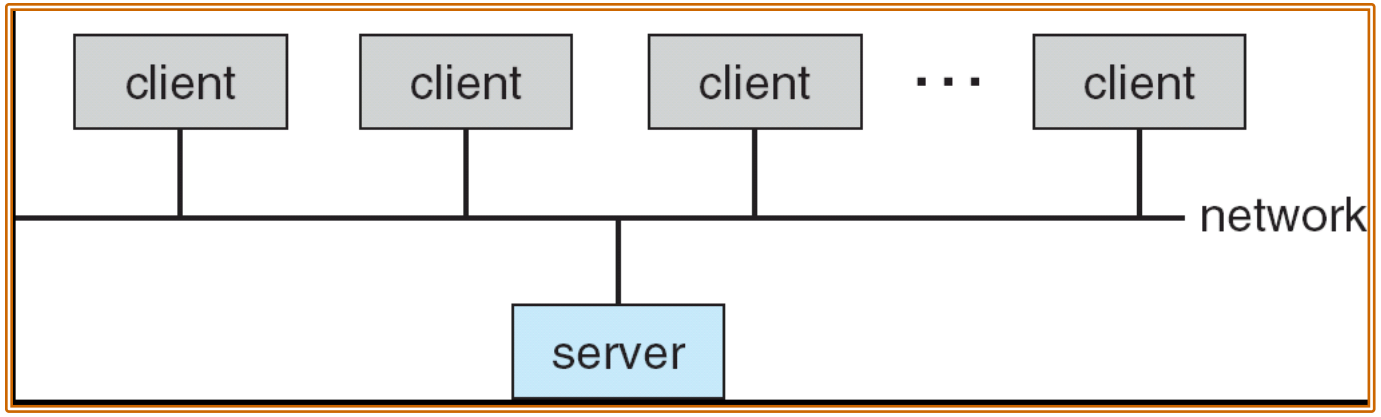


Windows 3.1



History Phase 4 (1989—): Distributed Systems

- Networking (Local Area Networking)
  - Different machines share resources
  - Printers, File Servers, Web Servers
  - Client – Server Model
- Services
  - Computing
  - File Storage



History Phase 5 (1995—): Mobile Systems

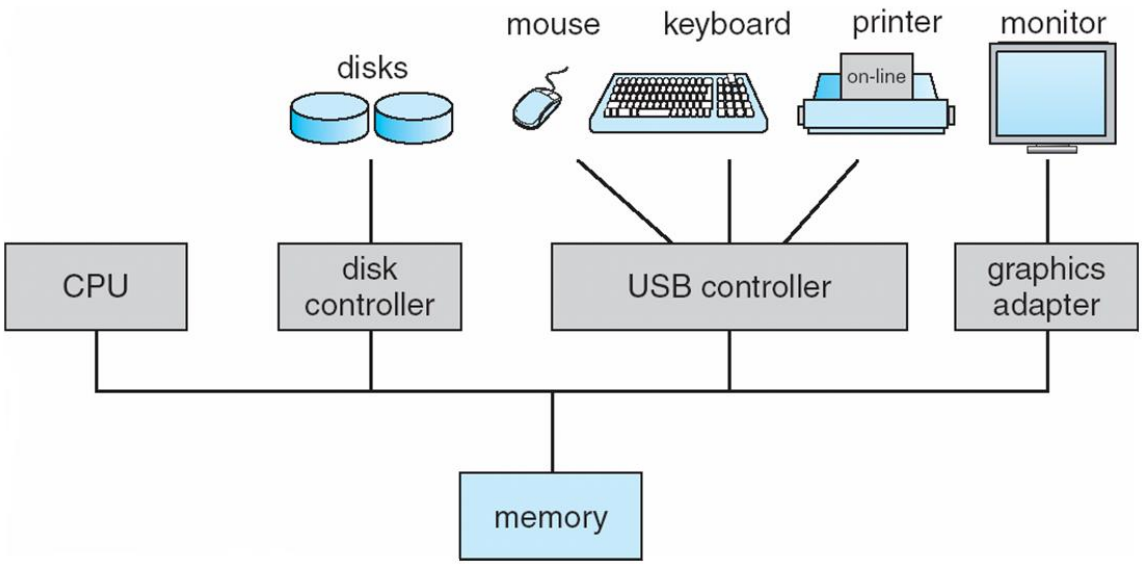
- Ubiquitous Mobile Devices
  - Laptops, PDAs, smart phones
  - Small, portable, and inexpensive
    - Recently twice as many smart phones as PDAs
    - Many computers/person!
  - Limited capabilities (memory, CPU, power, etc…)
- Wireless/Wide Area Networking
  - Leveraging the infrastructure
  - Huge distributed pool of resources extend devices
  - Traditional computers split into pieces. Wireless keyboards/mice, CPU distributed, storage remote
- Peer-to-peer systems
  - Many devices with equal responsibilities work together
  - Components of "Operating System" spread across globe

History of OS: Summary

- Change is continuous and OSs should adapt
  - Not: look how stupid batch processing was
  - But: Made sense at the time
- Situation today is much like the late 60s [poll]
  - Small OS: 100K lines
  - Large OS: >10M lines (5M for the browser!)
    - 100-1000 people-years
- Complexity still reigns
  - NT under development from early 90's to late 90's
    - Never worked very well
  - Windows XP has 45M lines of code,
  - Windows Vista delayed many times
    - Latest release date of 2005, 2006, 2007

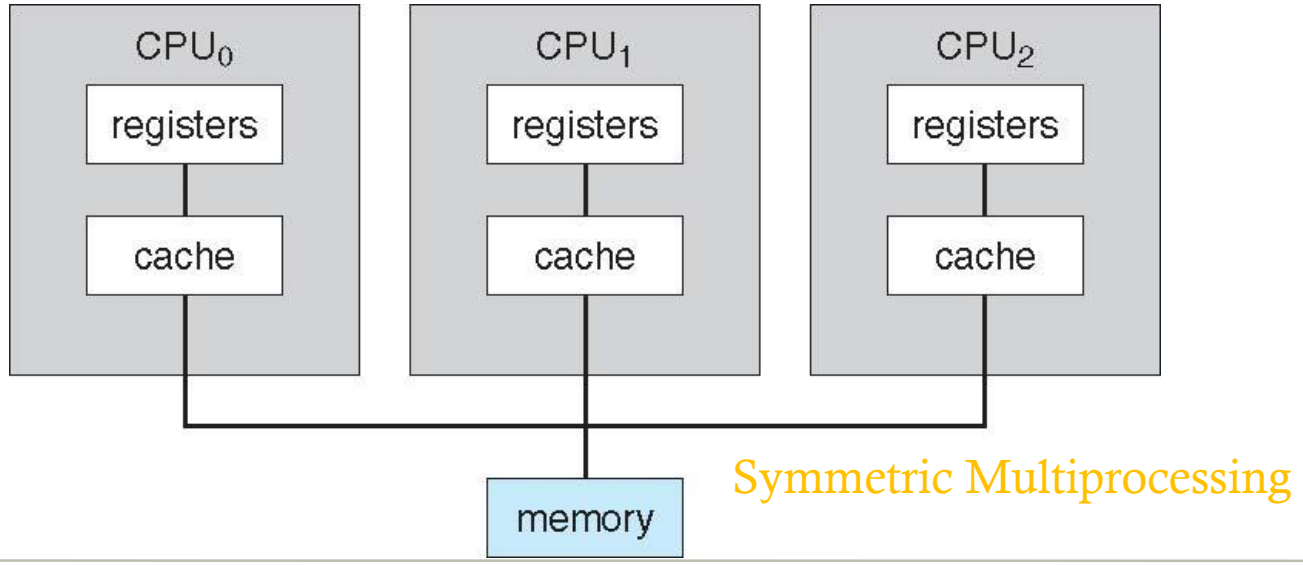
Computer System Organization

- 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



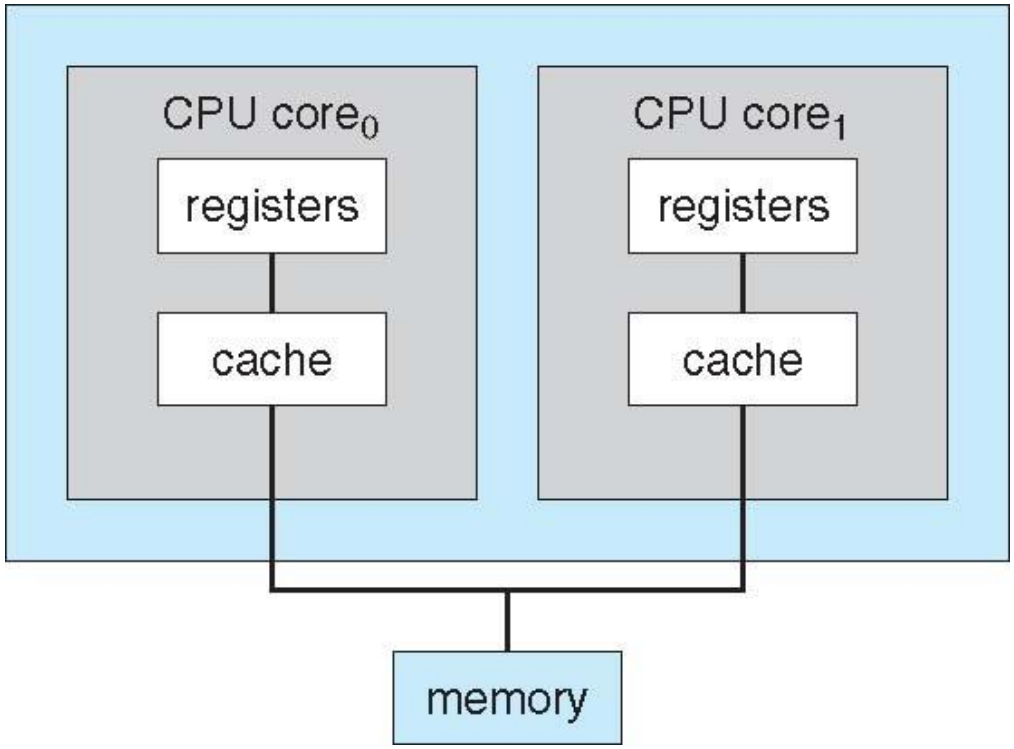
Computer-System Architecture

- Many systems use a single general-purpose processor (PDAs through mainframes)
  - Most systems have special-purpose processors as well.
- Multiprocessors systems growing in use and importance
  1. Asymmetric Multiprocessing
  2. Symmetric Multiprocessing



Symmetric Multiprocessing

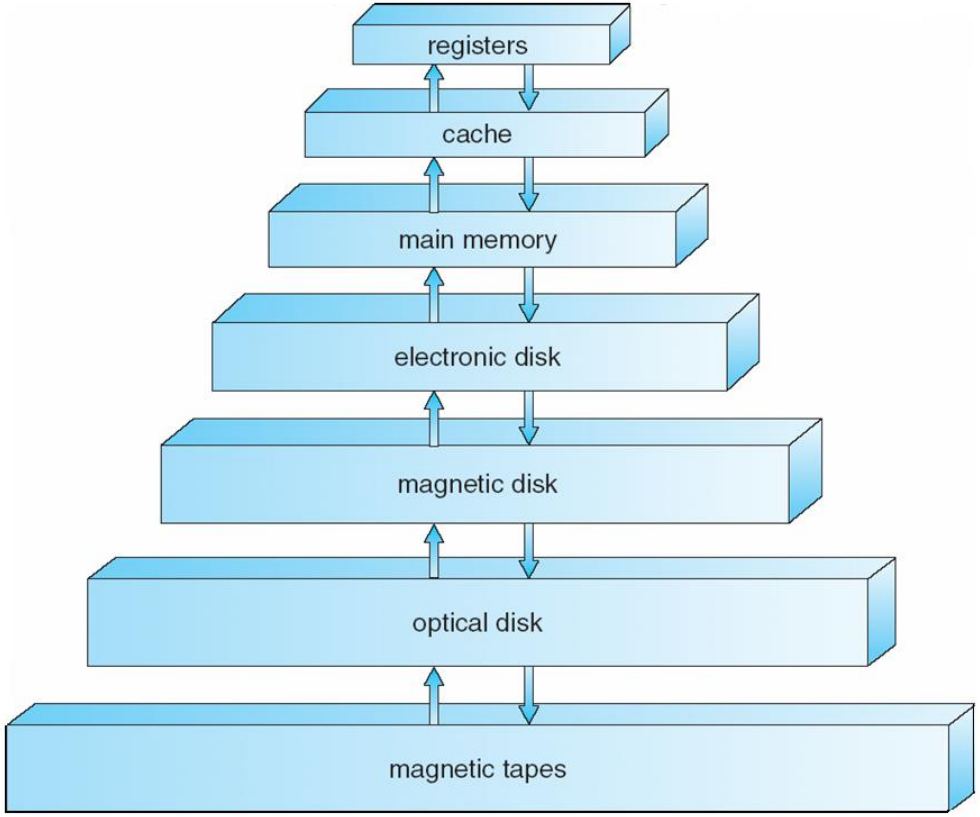
A Dual-Core Design



Storage Hierarchy

- Storage systems organized in hierarchy
  - Speed
  - Cost
  - Volatility
- Main memory – only large storage media that the CPU can access directly
- Magnetic (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
- **Caching** – copying information into faster storage system; main memory can be viewed as a last *cache* for secondary storage

Storage-Device Hierarchy



Performance of Various Levels of Storage

- Movement between levels of storage hierarchy can be explicit or implicit

Level	1	2	3	4
Name	registers	cache	main memory	disk storage
Typical size	< 1 KB	> 16 MB	> 16 GB	> 100 GB
Implementation technology	custom memory with multiple ports, CMOS	on-chip or off-chip CMOS SRAM	CMOS DRAM	magnetic disk
Access time (ns)	0.25 – 0.5	0.5 – 25	80 – 250	5,000.000
Bandwidth (MB/sec)	20,000 – 100,000	5000 – 10,000	1000 – 5000	20 – 150
Managed by	compiler	hardware	operating system	operating system
Backed by	cache	main memory	disk	CD or tape

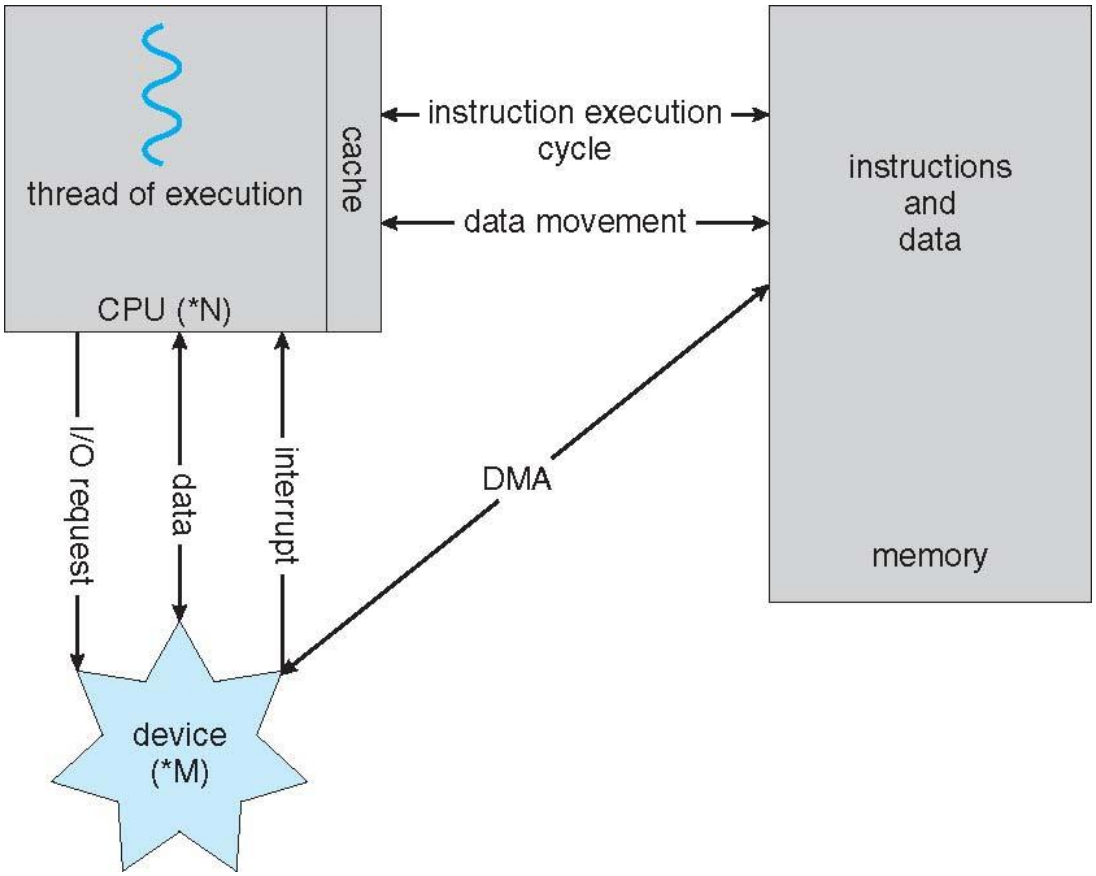
Caching

- Important principle, performed at many levels in a computer (in hardware, operating system, software)
- Information in use copied from slower to faster storage temporarily
- 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 important design problem
  - Cache size and replacement policy

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*

How a Modern Computer Works



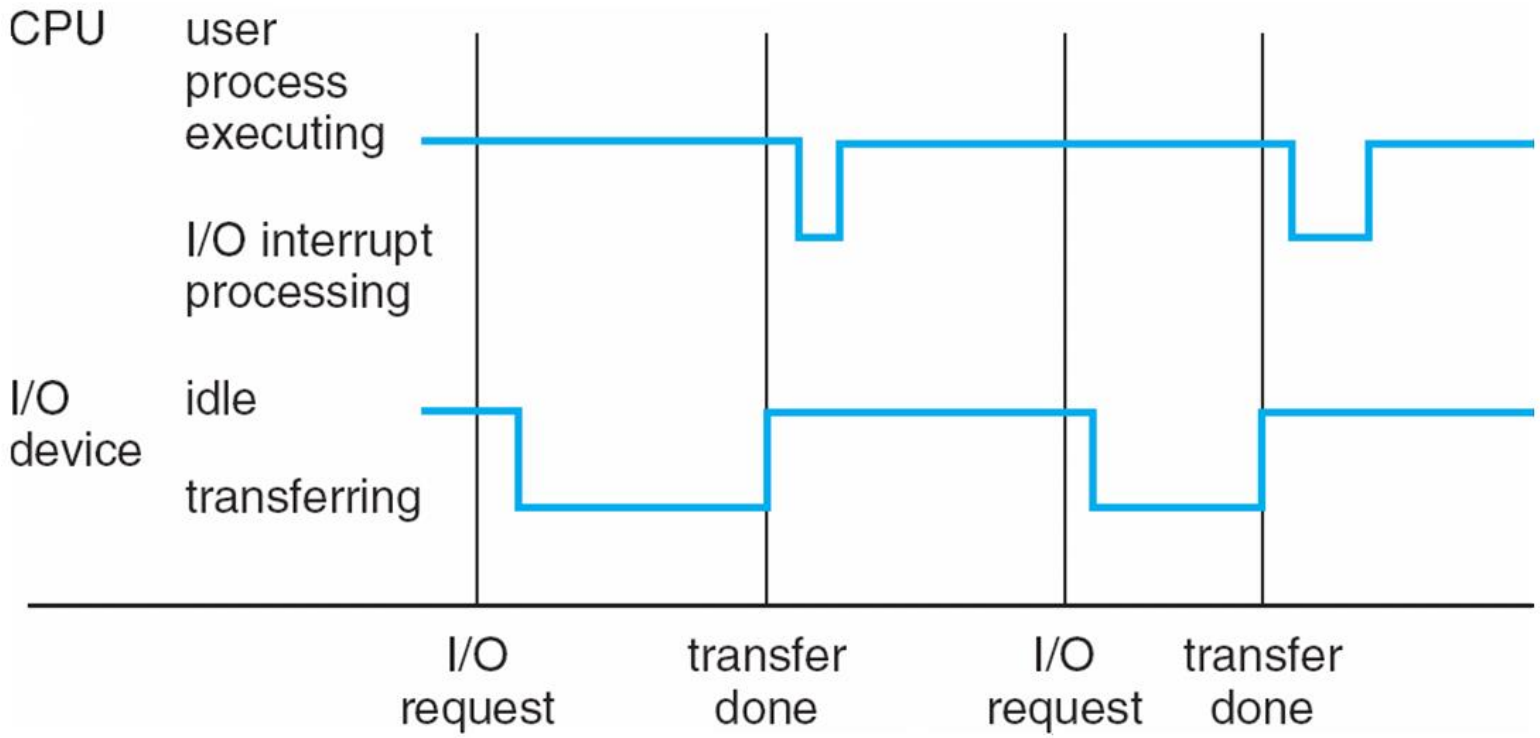
Interrupt Driven O.S.

- Most modern operating systems are interrupt driven.
  - Start-up: 1) Load O.S. (kernel) and start it running.
  - 2) O.S. waits for an event (an interrupt).
- Definition of interrupt:
  - An **interrupt** is a method to ensure that the CPU takes note of an event.
- Types of interrupt:
  - Hardware interrupts (e.g. from an I/O device)
  - Software interrupt (from an executing process)
    - Also called a **trap** or an **exception**
    - Generated to signal error (e.g. divide by zero)
    - or to request service from OS (system calls for I/O requests).

Interrupt Handling

- The operating system preserves the state of the CPU by storing registers and the program counter.
- The O.S. determines which type of interrupt has occurred.
  - Separate segments of code determine what action should be taken for each type of interrupt.
- The OS executes the sequence of commands associated with the given interrupt.
- The OS recovers the stored information from the original process and continues execution.

Interrupt Timeline



Summary

- Operating systems provide a virtual machine abstraction to handle diverse hardware
- Rapid Change in Hardware Leads to changing OS
  - Batch ⇒ Multiprogramming ⇒ Timeshare ⇒ Graphical UI ⇒ Ubiquitous Devices ⇒ Cyberspace/Metaverse/??
- OS features migrated from mainframes ⇒ PCs ⇒ Mobile

Summary of Operating System Principles

- OS as juggler: providing the illusion of a dedicated machine with infinite memory and CPU.
- OS as government: protecting users from each other, allocating resources efficiently and fairly, and providing secure and safe communication.
- OS as complex system: keeping OS design and implementation as simple as possible is the key to getting the OS to work.
- OS as history teacher: learning from past to predict the future, i.e., OS design tradeoffs change with technology.