

CO2017

Operating Systems





Course Syllabus

Credits	3			Code	CO2017
Credits Hours	Total: 64	Lecture: 30		Lab: 22	Assignment: 12
Evaluation	Exercise:	Lab: <i>20%</i>	Midterm:	Assignment: <i>30%</i>	Final exam: <i>50%</i>
Assessment method	Final exam: <i>Multiple choice questions</i> , ~ 90 minutes				
Prerequisites					
Co-requisites					
Undergraduate Programs	<i>Computer Science and Computer Engineering</i>				
Website	http://lms.hcmut.edu.vn/				
Phone	Phone: (84)(8) 38647256 (Ext. 5840)				
Email:	nqhung@hcmut.edu.vn				



Course Outcomes

L.O.1	Describe on how to apply fundamental knowledge of computing and mathematics in an operating system
L.O.1.1	Define the functionality and structures that a modern operating system must deliver to meet a particular need.
L.O.1.2	Explain virtual memory and its realization in hardware and software.
L.O.2	Able to report the tradeoffs between the performance and the resource and technology constraints in a design of an operating system.
L.O.2.1	Compare and contrast common algorithms used for both preemptive and non-preemptive scheduling of tasks in operating systems
L.O.2.2	Compare and contrast different approaches to file organization, recognizing the strengths and weaknesses of each.
L.O.3	Describe main operating system concepts and their aspects that are useful to realize concurrent systems and describe the benefits of each.
L.O.3.1	Compare and contrast different methods for process synchronization.



Course Outline

- Introduction to Operating systems
- Processes/Threads management
 - Process and Threads
 - CPU scheduling
 - Synchronization
- Memory management
 - Main memory
 - Virtual memory
- Storage management
 - File systems
- Advanced topics
- Summary

VIRTUALIZATION

CONCURRENCY

PERSISTENCE

Textbook and References

- [1] “Operating System Concepts”, [Abraham Silberschatz](#), [Greg Gagne](#), [Peter B. Galvin](#), 10th Edition, John Wiley & Sons, 2018. ISBN1119439256, 9781119439257.
- [2] “Operating Systems: Three Easy Pieces”, [Remzi H. Arpaci-Dusseau](#), [Andrea C. Arpaci-Dusseau](#), CreateSpace Independent Publishing Platform, 2018. ISBN198508659X, 9781985086593.
- [3] “Modern Operating System”, Andrew S. Tanenbaum and Herbert Bos, Pearson Education © 2023, eBook ISBN 13: 978-1-292-72789-9



Acknowledgement

- Acknowledgement: The lecture uses mostly slides from “Operating system concepts” copyrighted by Silberschatz, Galvin and Gagne, 2018.

Introduction





Chapter 1: Introduction

- What Operating Systems Do
- Computer-System Organization
- Computer-System Architecture
- Operating-System Operations
- Resource Management
- Security and Protection
- Virtualization
- Distributed Systems
- Kernel Data Structures
- Computing Environments
- Free/Libre and Open-Source Operating Systems

Objectives

- Describe the general organization of a computer system and the role of interrupts
- Describe the components in a modern, multiprocessor computer system
- Illustrate the transition from user mode to kernel mode
- Discuss how operating systems are used in various computing environments
- Provide examples of free and open-source operating systems



What Does the Term Operating System Mean?

- An operating system is “.....”
- What about:
 - Car
 - Airplane
 - Printer
 - Washing Machine
 - Toaster
 - Compiler
 - Etc.



What is an Operating System?

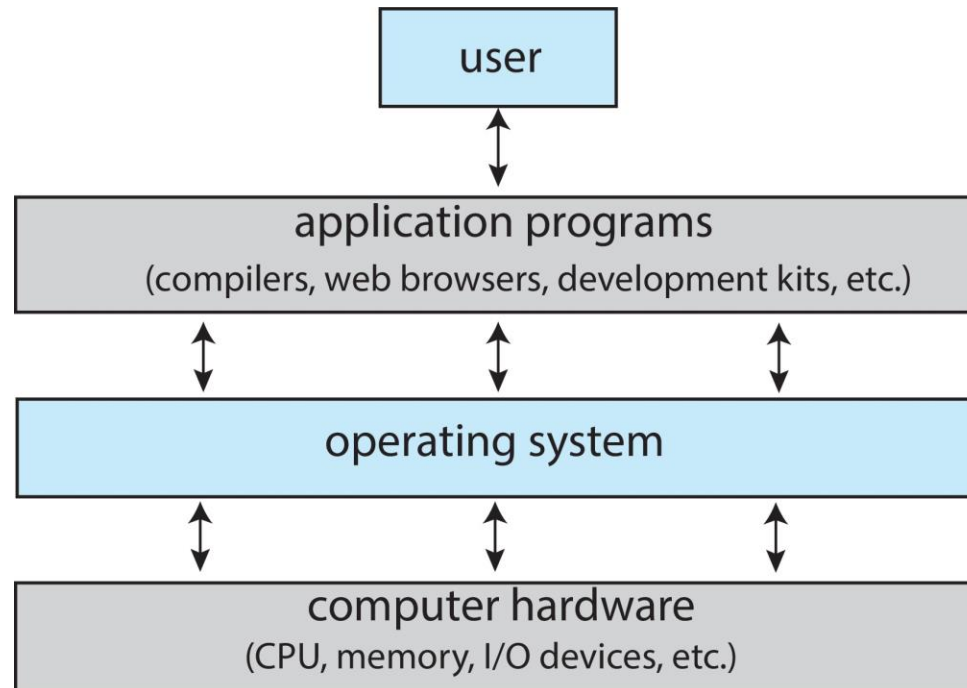
- A program that acts as an intermediary between a user of a computer and the computer hardware
- Operating system goals:
 - Execute user programs and make solving user problems easier
 - Make the computer system convenient to use
 - Use the computer hardware in an efficient manner

Computer-System Structure

- Computer system can be divided into **4** components:
 - **Hardware (HW)** – provides basic computing resources
 - 4 E.g., **Central Processing Unit (CPU)**, memory, **Input/Output (I/O)** devices
 - **Operating system (OS)** – controls and coordinates use of hardware among various applications and users
 - 4 E.g., Microsoft Windows, Unix, Linux, Apple MacOS
 - **Application programs** – define the ways in which the system resources are used to solve the computing problems of the users
 - 4 E.g., Compilers, web browsers, development kits, word processors, database systems, video games, multimedia players
 - **Users**
 - 4 E.g., People, machines, other computers



Abstract View of Components of Computer





What Operating Systems Do

- **Users** want convenience, ease of use and good performance
 - 4 But, don't care about resource utilization
- **Shared computers** (e.g, *mainframe* or *minicomputer*) keep all users happy
 - 4 Operating system is a resource allocator and control program making efficient use of HW and managing execution of user programs
- **Dedicated systems** (e.g, *workstations*) have dedicated resources but users frequently utilize shared resources from *servers*
- **Mobile devices** (e.g., *smartphones* and *tablets*) are resource poor, have to be optimized for battery life and usability using user interfaces such as touch screens.
- Some computers have little or no user interface, such as **embedded computers** in devices and automobiles
 - 4 Run primarily without user intervention



Operating System Definition (Cont.)

- “*Everything a vendor ships when you order an operating system*” is a good approximation (but varies wildly)
- “*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



OS History

- Evolution of MacOS ()

<https://www.youtube.com/watch?v=HkDiPDCoGkk>

- Evolution of Microsoft Windows:

<https://www.youtube.com/watch?v=pYxNs5kYJwk&t=135s>

[History of Windows](#)

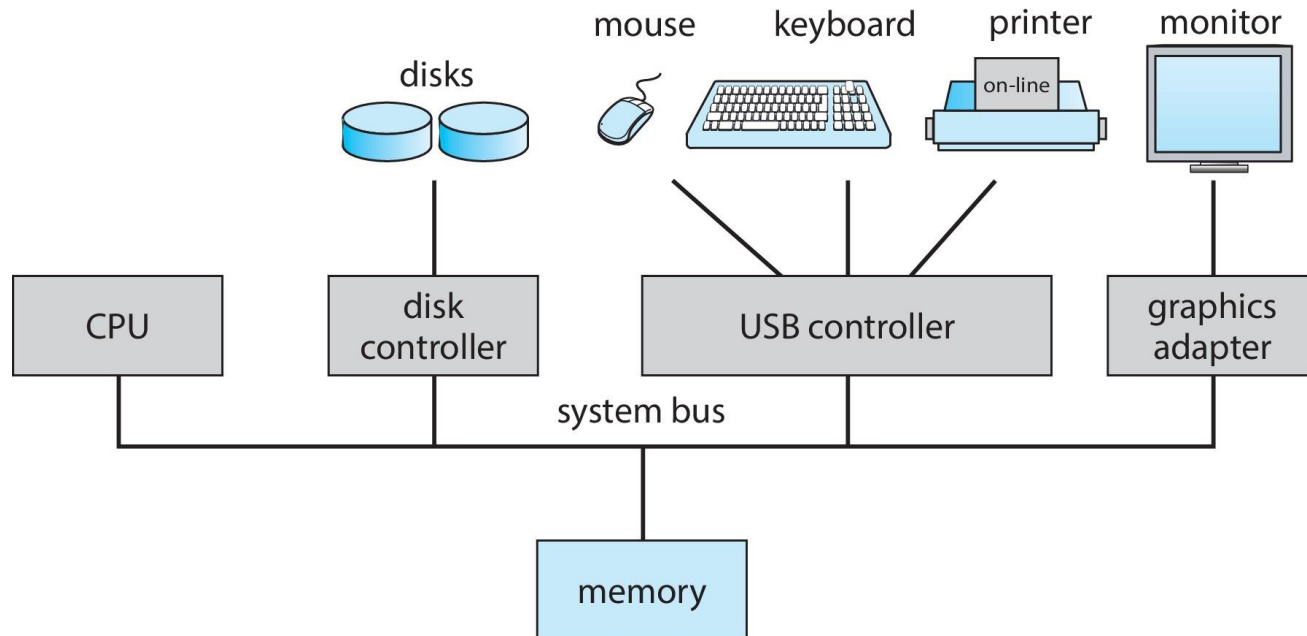
[Evolution of Mac OS](#)

[Windows & Apple's OS](#)

Overview of Computer System Structure

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

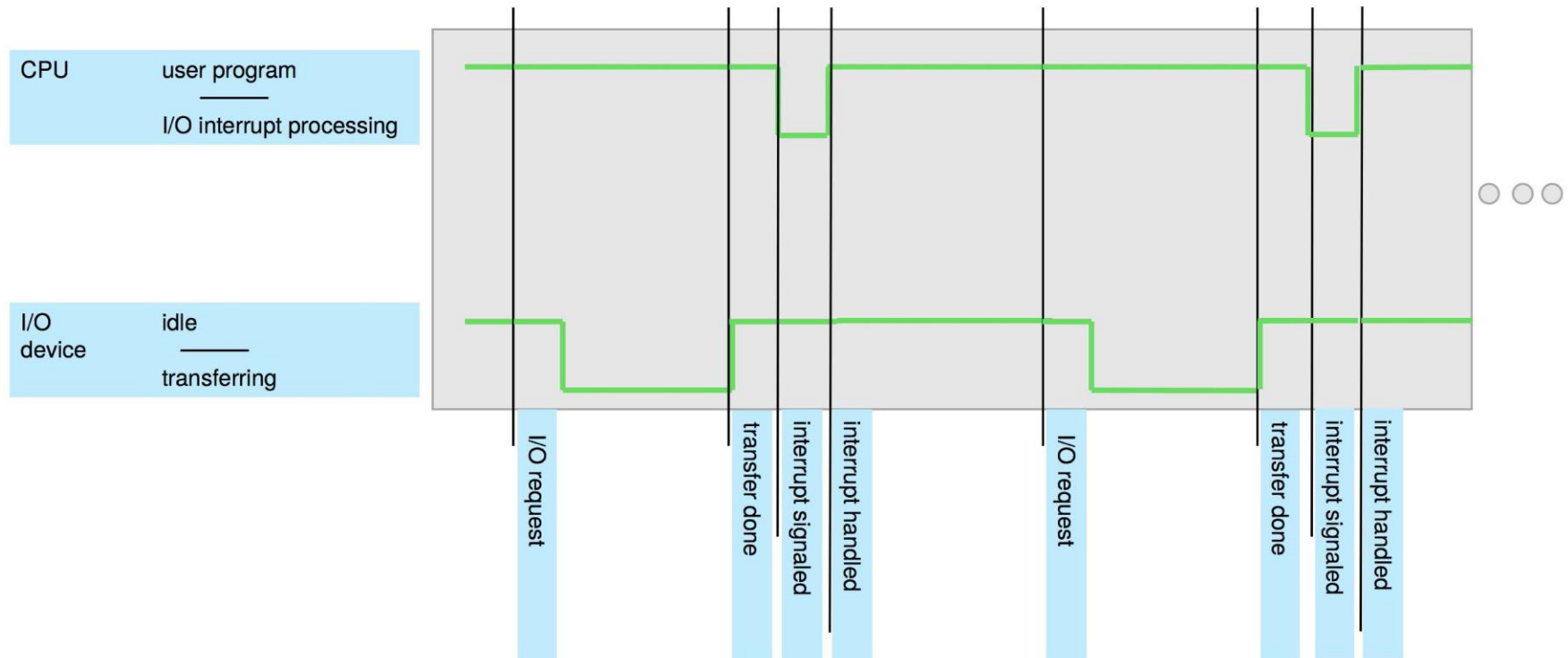
- **I/O devices** and the **CPUs** can execute *concurrently*
- The **device controller** (on device) determines the logical interaction between the device and the computer
 - Each device controller is in charge of a particular device type
 - Each device controller has a *local buffer*
- CPU moves data from/to *main memory* to/from *local buffers*, I/O device does from the *device* to *local buffer* of controller
- Each device controller type has a **device driver** (installed inside an operating system) to manage I/O operation
 - Provides uniform interface between controller and kernel



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**

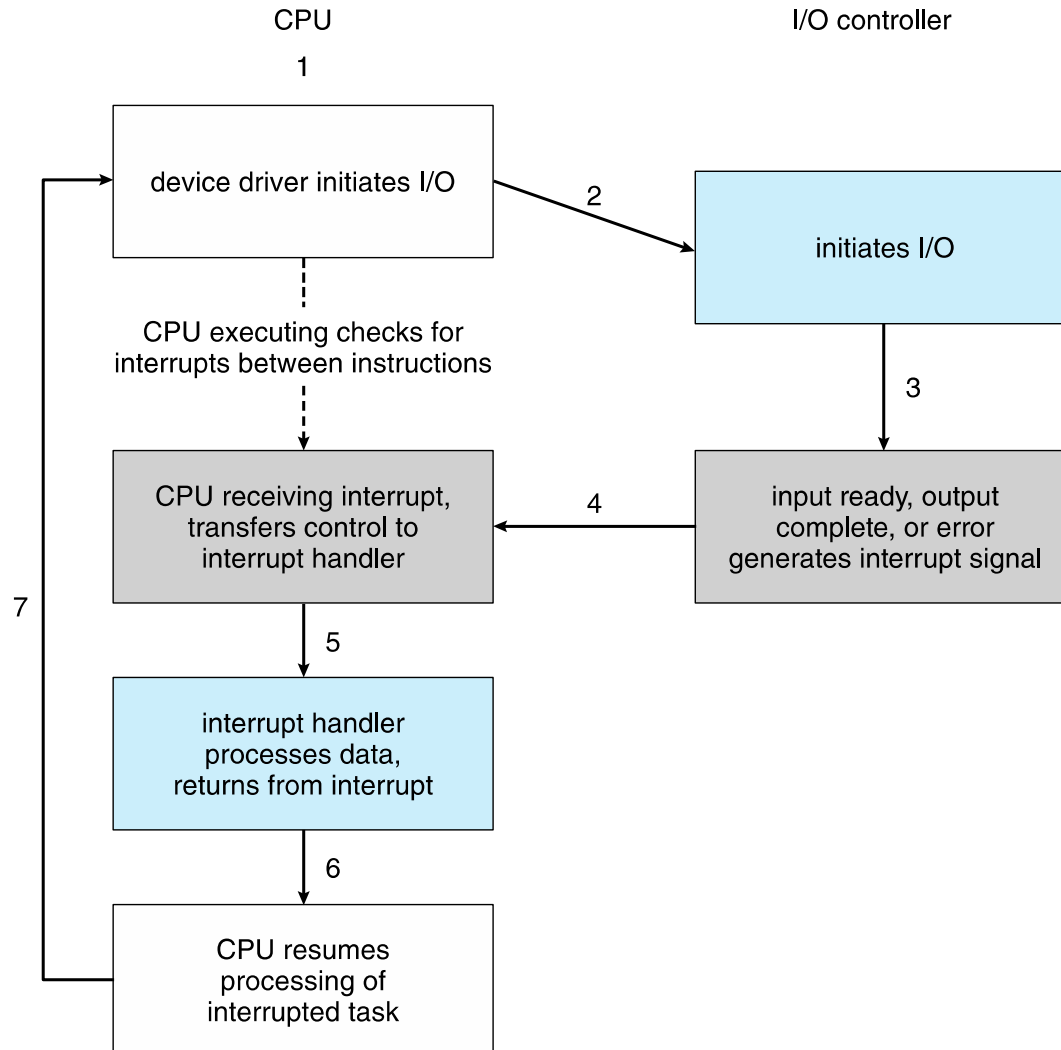
Interrupt Timeline



Interrupt Handling

- The operating system preserves the state of the CPU by storing **registers** and the **program counter (PC)**
- Determines which type of interrupt has occurred:
 - *Vectored interrupt system* used to handle asynchronous events and to trap to supervisor-mode routines in the kernel
 - Separate segments of code determine what action should be taken for each type of interrupt
- Some device drivers use *interrupts* when the I/O rate is low and switch to *polling* when the rate increases to the point where polling is faster and more efficient.

Interrupt-drive I/O Cycle





I/O Structure

- Two methods for handling I/O
 - After I/O starts, control returns to user program only upon I/O completion
 - After I/O starts, control returns to user program without waiting for I/O completion

- 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 (e.g., 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*
 - 4 OS indexes into **I/O device table** to determine device status and to modify table entry to include interrupt

Storage Structure

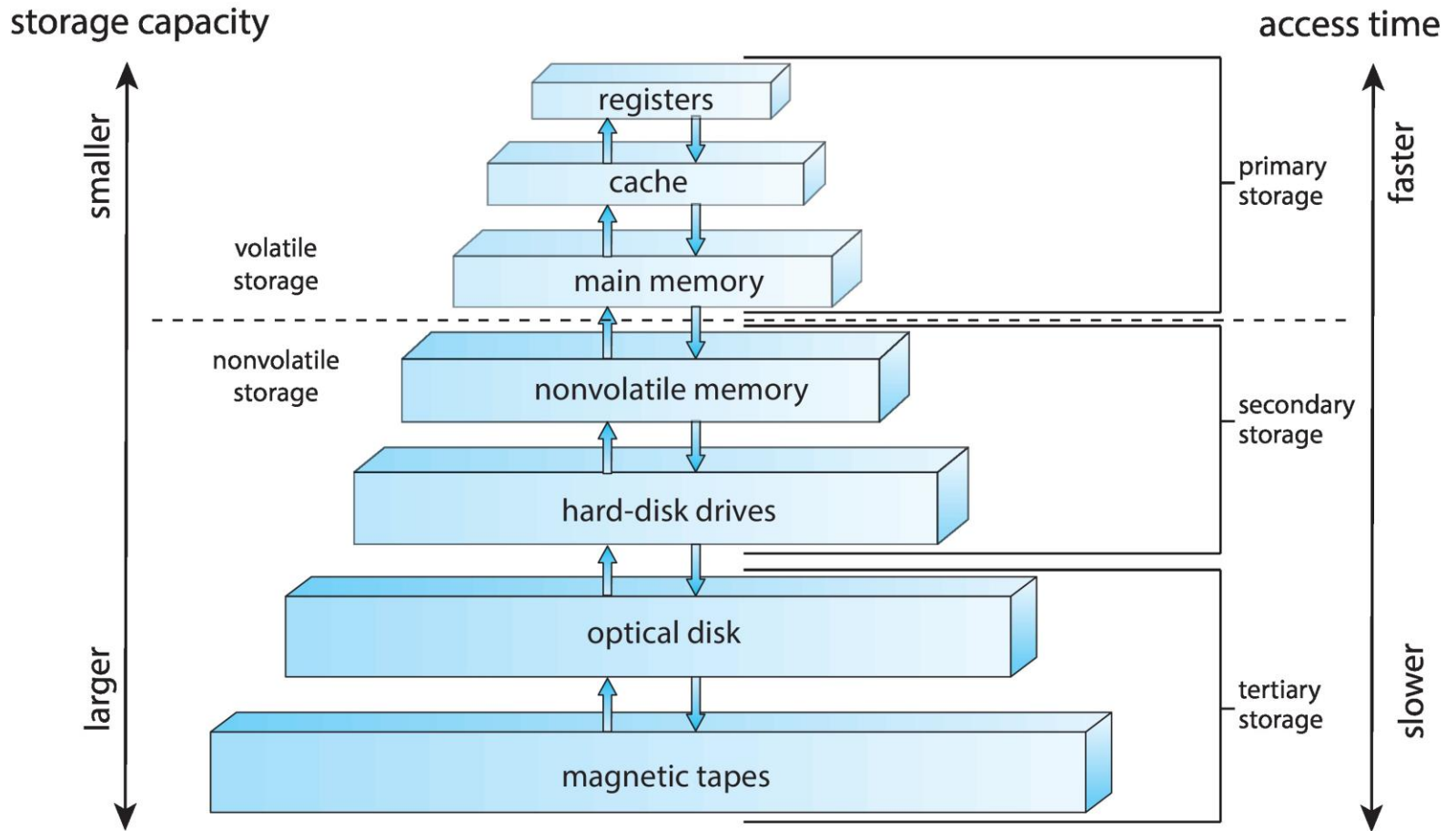
Storage Structure

- **Main memory** – only storage media that the CPU can access directly
 - *Random access*, typically in the form of **Dynamic Random-Access Memory (DRAM)**
 - Typically *volatile*
- **Secondary storage** – extension of main memory that provides large *nonvolatile* storage capacity
 - **Hard Disk Drives (HDD)** – rigid metal or glass platters covered with magnetic recording material
 - 4 Disk surface is logically divided into *tracks*, which are subdivided into *sectors*
 - **Non-volatile memory (NVM)** devices– *faster* than hard disks, *nonvolatile*
 - 4 Becoming more popular as capacity and performance increases, price drops
 - 4 Various technologies

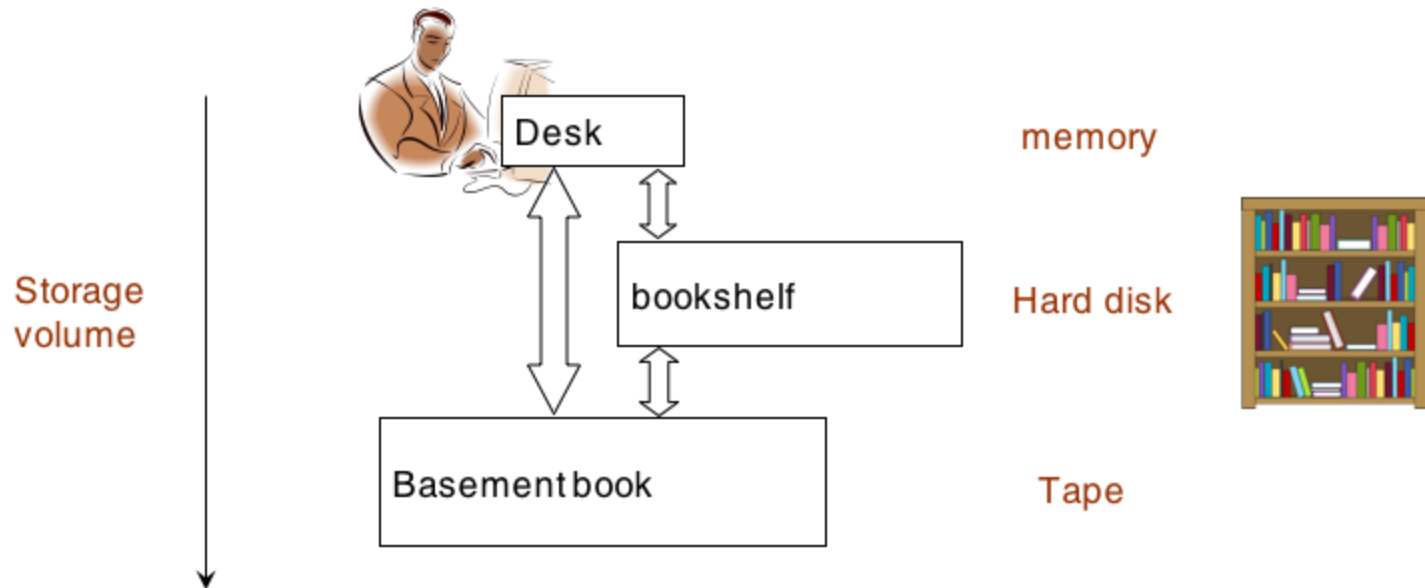
Storage Hierarchy

- Storage systems are organized in hierarchy according to
 - *Speed* (or *access time*)
 - *Capacity*
 - *Volatility*
 - *Cost*
- **Caching** – mechanism copying data into faster storage system
 - Main memory can be viewed as a cache for secondary storage

Storage-Device Hierarchy



An example of storage hierarchy

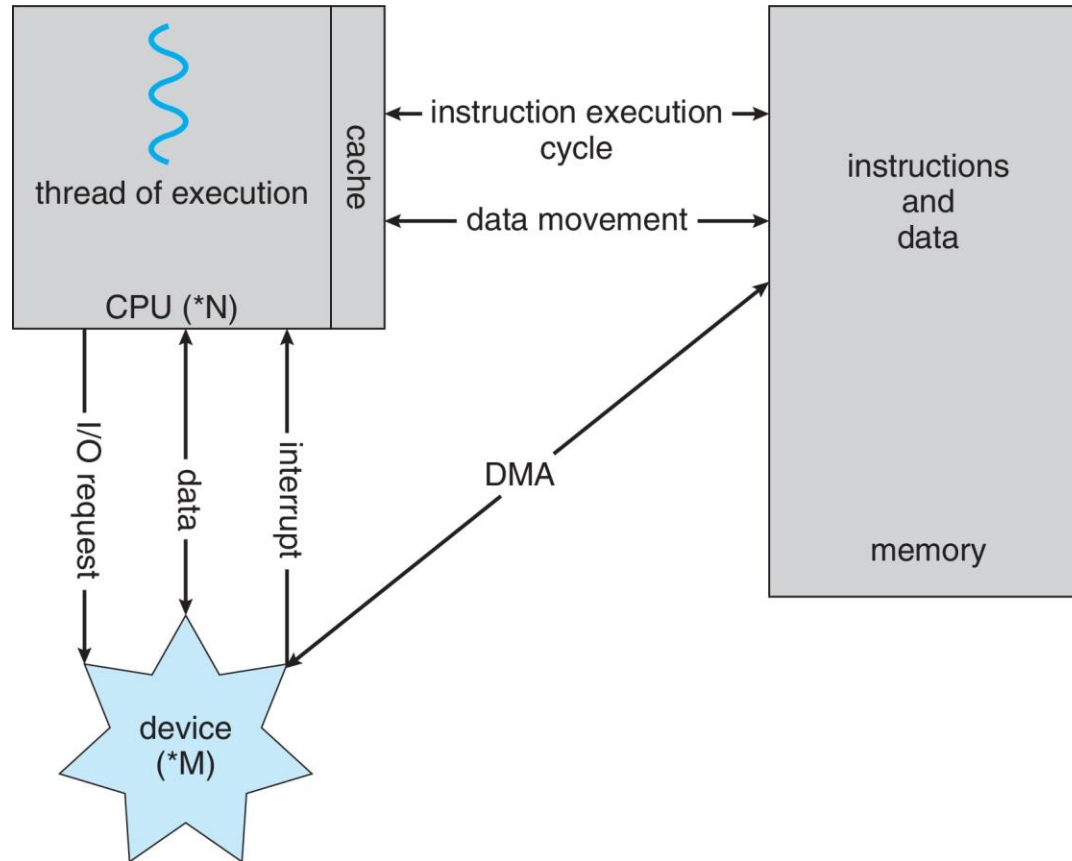




Direct Memory Access Structure

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

How a Modern Computer Works



A von Neumann architecture



Operating-System Operations

- Bootstrap program – simple code to initialize the system, load the kernel
- Kernel loads
- Starts **system daemons** (services provided outside of the kernel)
- Kernel **interrupt driven** (hardware and software)
 - Hardware interrupt by one of the devices
 - Software interrupt (**exception** or **trap**):
 - 4 Software error (e.g., division by zero)
 - 4 Request for operating system service – **system call**
 - 4 Other process problems include infinite loop, processes modifying each other or the operating system



Multiprogramming (Batch system)

- Single user cannot always keep CPU and I/O devices busy
- Multiprogramming organizes jobs (code and data) so CPU always has one to execute
- A subset of total jobs in system is kept in memory
- One job selected and run via **job scheduling**
- When job has to wait (for I/O for example), OS switches to another job

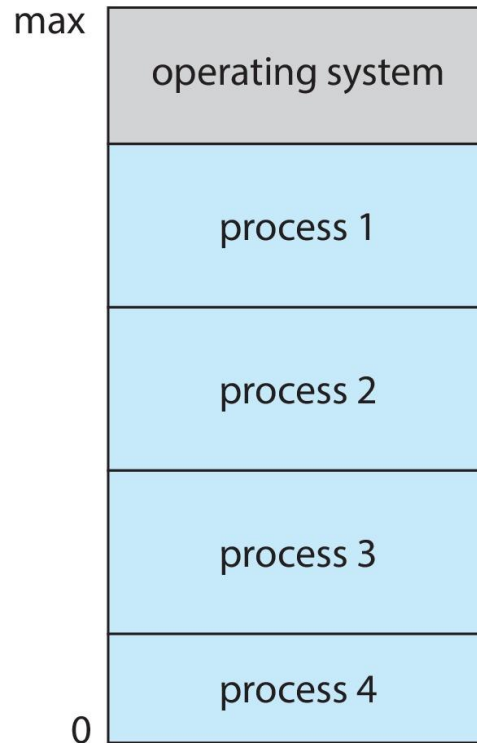


Multitasking (Timesharing)

- A logical extension of Batch systems– the CPU switches jobs so frequently that users can interact with each job while it is running, creating **interactive** computing
 - **Response time** should be < 1 second
 - Each user has at least one program executing in memory
⇒ **process**
 - If several jobs ready to run at the same time ⇒ **CPU scheduling**
 - If processes don't fit in memory, **swapping** moves them in and out to run
 - **Virtual memory** allows execution of processes not completely in memory



Memory Layout for Multiprogrammed System

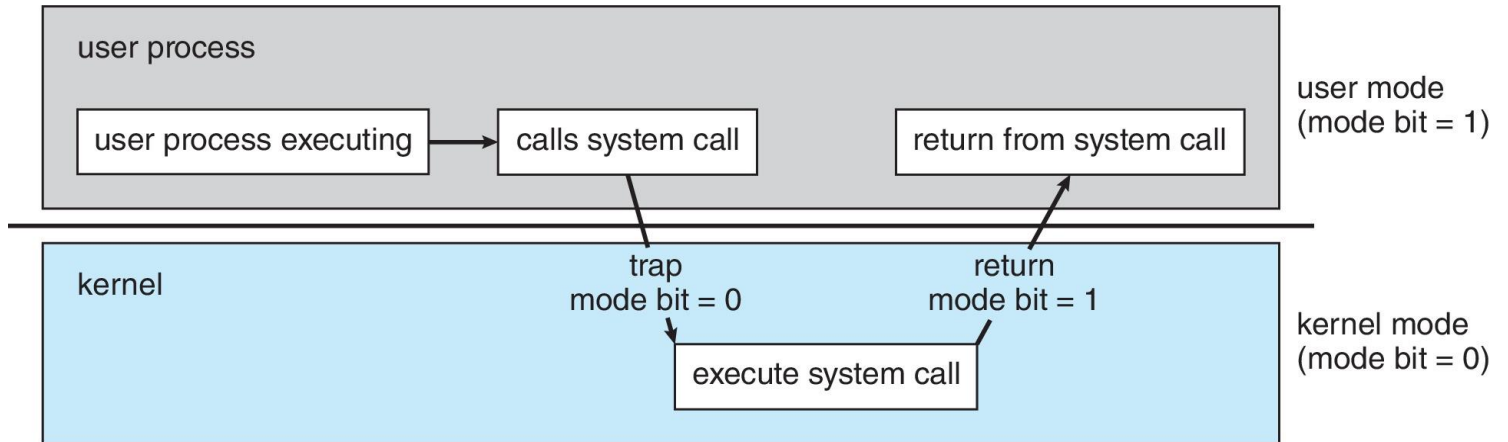




Dual-mode Operation

- **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

Transition from User to Kernel Mode



- Timer to prevent infinite loop (or process hogging resources)
 - Timer is set to interrupt the computer after some time period
 - Keep a counter that is decremented by the physical clock
 - Operating system set the counter (privileged instruction)
 - When counter zero generate an interrupt
 - Set up before scheduling process to regain control or terminate program that exceeds allotted time

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
 - 4 CPU, memory, I/O, files, initialization data
 - 4 Process termination requires reclaim of any reusable resources
 - **Single-threaded process** has *one program counter* specifying location of next instruction to execute
 - 4 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 processes) running concurrently on one or more CPUs
 - **Concurrency** by multiplexing the CPUs among the processes / threads



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*
 - Providing mechanisms for *deadlock handling*



Memory Management

- To execute a program,
 - All (or part) of the **instructions** must be in memory
 - All (or part) of the **data** needed by the program must be in memory
- Memory management determines what is in memory and when
 - Optimizing CPU utilization and computer response to users
- OS activities
 - *Keeping track* of which parts of memory are currently being used and by whom
 - *Deciding* which processes (or parts thereof) and data to move into and out of memory
 - *Allocating* and *deallocating* memory space as needed

Filesystem Management

- OS provides uniform, logical view of *data storage*
 - Abstracts physical properties to logical storage unit - **file**
 - Each medium is controlled by device (i.e., disk drive, tape drive)
 - 4 Varying properties include *access speed, capacity, data-transfer rate, access method (sequential or random)*
- Filesystem management
 - **Files** usually organized into **directories**
 - **Access control** on most systems to determine who can access what
- OS activities
 - *Creating* and *deleting* files / directories, primitives to *manipulate* files / directories, to *backup* files onto stable (non-volatile) storage media
 - *Mapping* files onto secondary storage



Mass-Storage Management

- Usually **disks** used to store **programs** and **data** that do not fit in **main memory** or that must be kept for a “long” period of time
- Proper management is of central importance
- *Entire speed of computer operation hinges on disk subsystem and its algorithms*
- OS activities
 - *Mounting* and *unmounting*
 - *Free-space* management
 - Storage *allocation*
 - Disk *scheduling*
 - *Partitioning*
 - *Protection*

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 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 is smaller than storage being cached
 - Cache management is an important design problem
 - *Cache size and replacement policy*

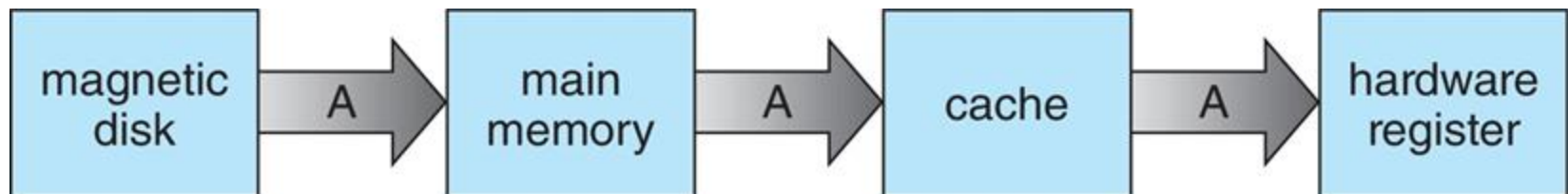
Various Types of Storage

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

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

Migration of Data from Disk to Register

- **Multitasking environment** must be careful to use most recent value, no matter where it is stored in the storage hierarchy
- **Multiprocessor environment** must provide cache coherency in hardware such that all CPUs have the most recent value in their cache
- In **distributed environment**, the situation is even more complex
 - Several copies of a datum can exist
 - Various solutions



I/O Subsystem

- One purpose of OS is to hide peculiarities of hardware devices from the user
- **I/O subsystem** is responsible for
 - Memory management of I/O including *buffering* (storing data temporarily while it is being transferred), *caching* (storing parts of data in faster storage for performance), *spooling* (the overlapping of output of one job with input of other jobs)
- I/O subsystem includes
 - *General device-driver interface*
 - *Drivers* for specific hardware devices

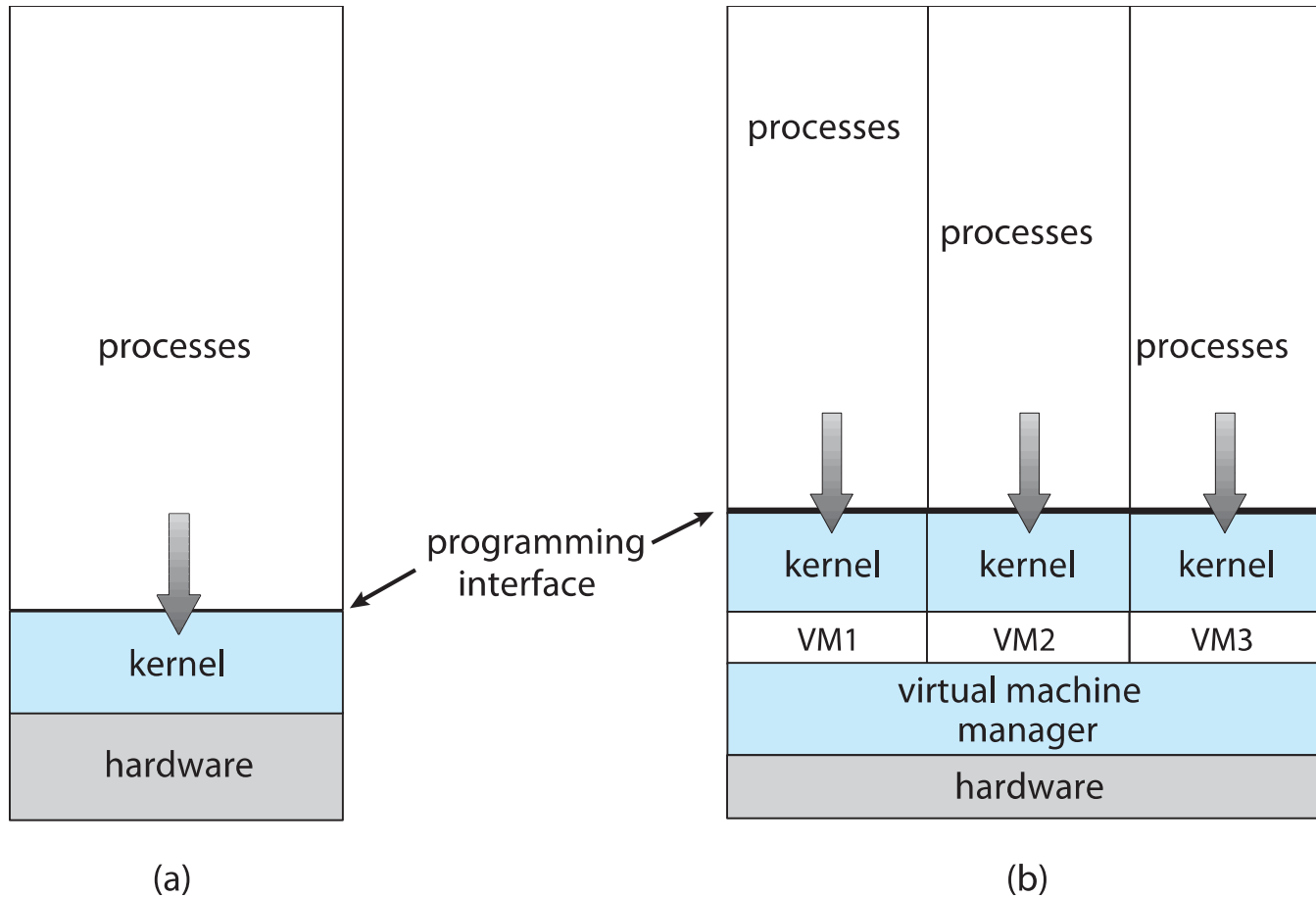
Protection and Security

- **Protection** – any mechanism for controlling access of processes or users to resources defined by the OS
- **Security** – defense of the system against internal and external attacks
 - Huge range, including *denial-of-service*, *worms*, *viruses*, *identity theft*, *theft of service*
- Systems generally first distinguish among users, to determine who can do what
 - 4 **User identity** (**UID**, or security ID) includes name and an associated number. User ID is then associated with all files, processes of that user to determine access control
 - 4 **Group identifier** (**GID**) allows set of users to be defined for access control, then also associated with each process or file
 - 4 **Privilege escalation** allows user to change to *effective ID* with more rights

- *Allows operating systems to run applications within other OSes*
 - Vast and growing industry
- **Emulation** used when source CPU type different from target CPU type (e.g., PowerPC to Intel x86)
 - Generally slowest method
 - When computer language not compiled to native code – *Interpretation*
- **Virtualization** – OS natively compiled for CPU, running **guest** OSes also natively compiled
 - E.g., Consider VMware running WinXP **guests**, each running applications, all on native WinXP **host** OS
 - **Virtual Machine Manager (VMM)** provides virtualization services

Virtualization (cont.)

- Use cases involve laptops and desktops running multiple OSES for exploration or compatibility
 - E.g.,
 - 4 Apple laptop running **Mac OS X** host, **Windows** as a guest
 - 4 Developing apps for multiple OSES without having multiple systems
 - 4 Q&A testing applications without having multiple systems
 - 4 Executing and managing computing environments within data centers
- VMM can run natively, in which case they are also the host
 - There is no general purpose host then (e.g., **VMware ESX** and **Citrix XenServer**)

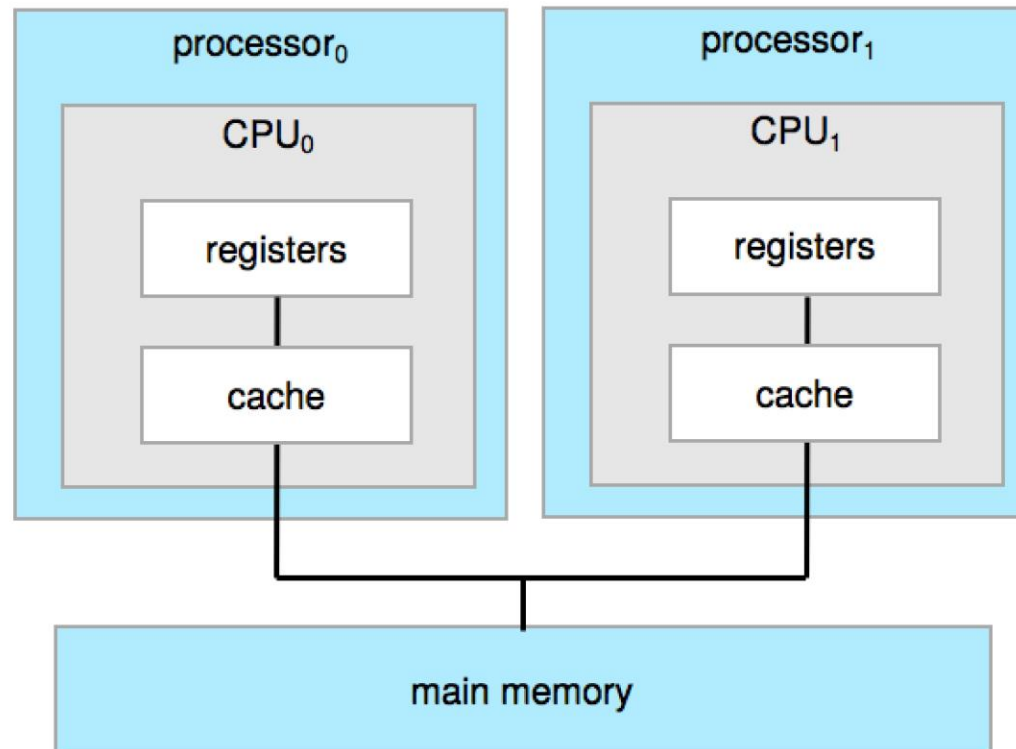


- **Distributed computing**
 - Collection of separate (possibly heterogeneous) systems networked together
 - 4 **Network** is communications paths (**TCP/IP** is most common protocol stack)
 - **Local Area Network (LAN)**
 - **Wide Area Network (WAN)**
 - **Metropolitan Area Network (MAN)**
 - **Personal Area Network (PAN)**
 - **Network Operating System (NOS)** provides features between systems across network
 - 4 Communication scheme allows systems to exchange messages
 - 4 Illusion of a single system

Computer System Architecture

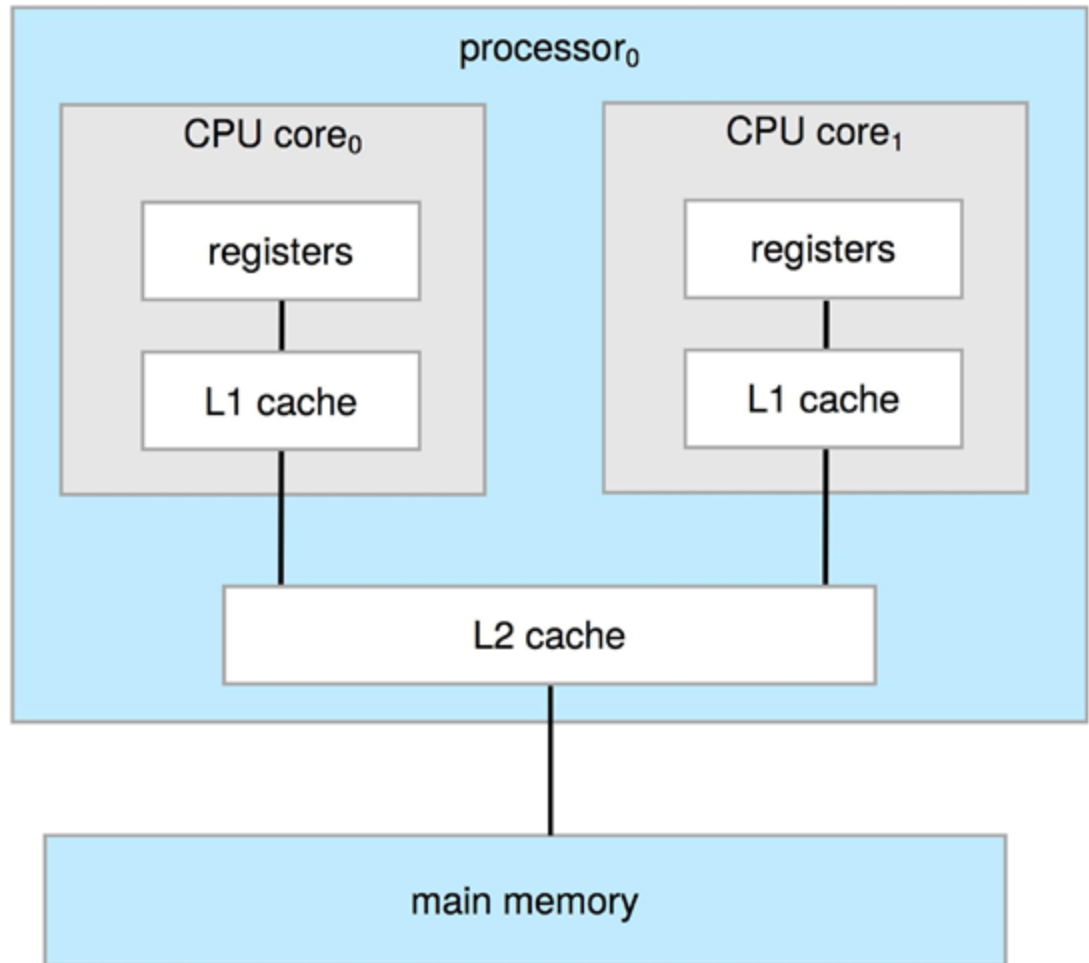
- Most older systems use a single *general-purpose processor*
 - Most systems have *special-purpose processors* as well
- **Multiprocessors** systems growing in use and importance
 - Also known as **parallel systems**, **tightly-coupled systems**
 - Advantages include:
 - 4 *Increased throughput*
 - 4 *Economy of scale, increased reliability* – graceful degradation or fault tolerance
 - Two types:
 - 4 **Asymmetric Multiprocessing** – each processor is assigned a special task.
 - 4 **Symmetric Multiprocessing** – each processor performs all tasks

Symmetric Multiprocessing Architecture

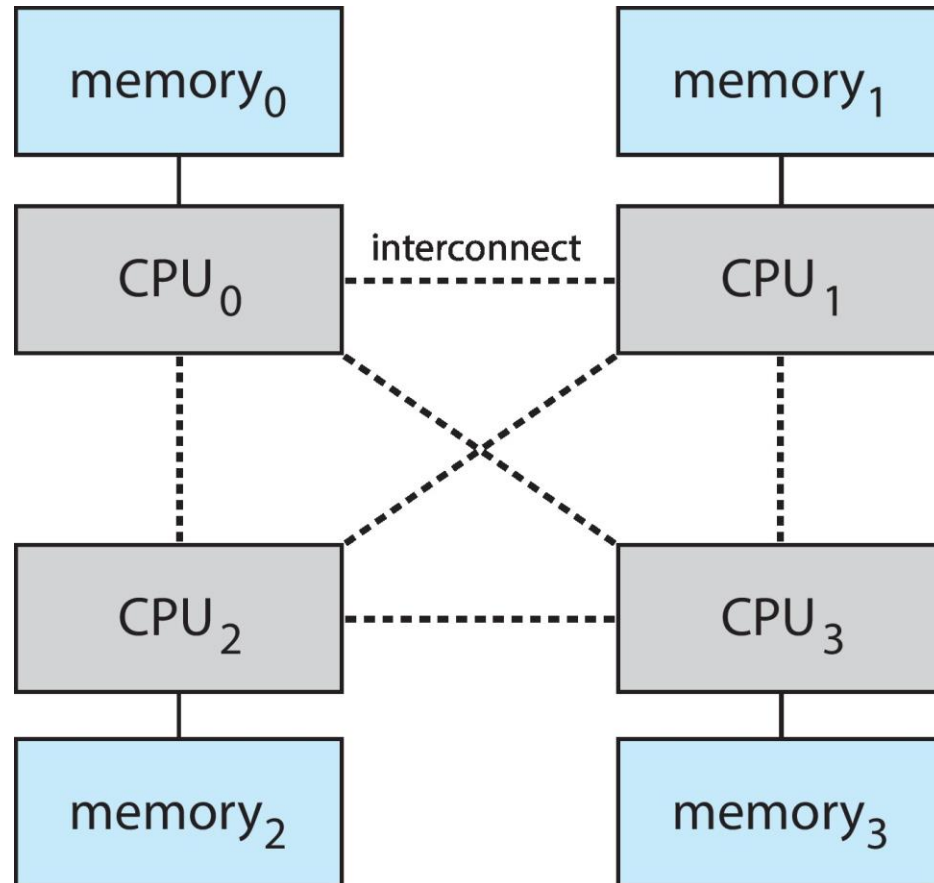


A Dual-Core Design

- **Multi-chip** and **multicore**
- **Systems** containing all chips
- **Chassis** containing multiple separate systems

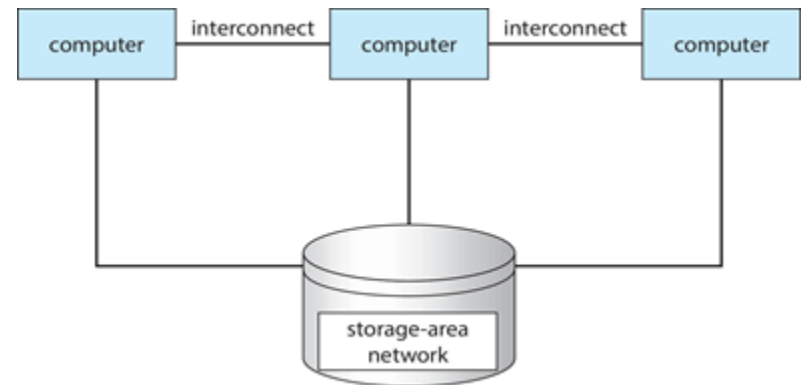


Non-Uniform Memory Access System



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
 - 4 *Asymmetric clustering* has one machine in hot-standby mode
 - 4 *Symmetric clustering* has multiple nodes running applications
- Some clusters are used for **High-Performance Computing (HPC)**
 - 4 Applications must be written to use *parallelization*
- Some clusters have **Distributed Lock Manager (DLM)** to avoid conflicting operations



Computer System Environments



Computing Environments

- Traditional
- Mobile
- Client Server
- Pear-to-Pear
- Cloud computing
- Real-time Embedded



Computing Environments - Traditional

- *Stand-alone general purpose machines*
- But blurred as most systems interconnect with others (i.e., the Internet)
- **Portals** provide web access to internal systems
- **Network computers** (or *thin clients*) are like Web terminals
- **Mobile computers** interconnect via *wireless networks*
- Networking becoming ubiquitous – even home systems use **firewalls** to protect home computers from Internet attacks

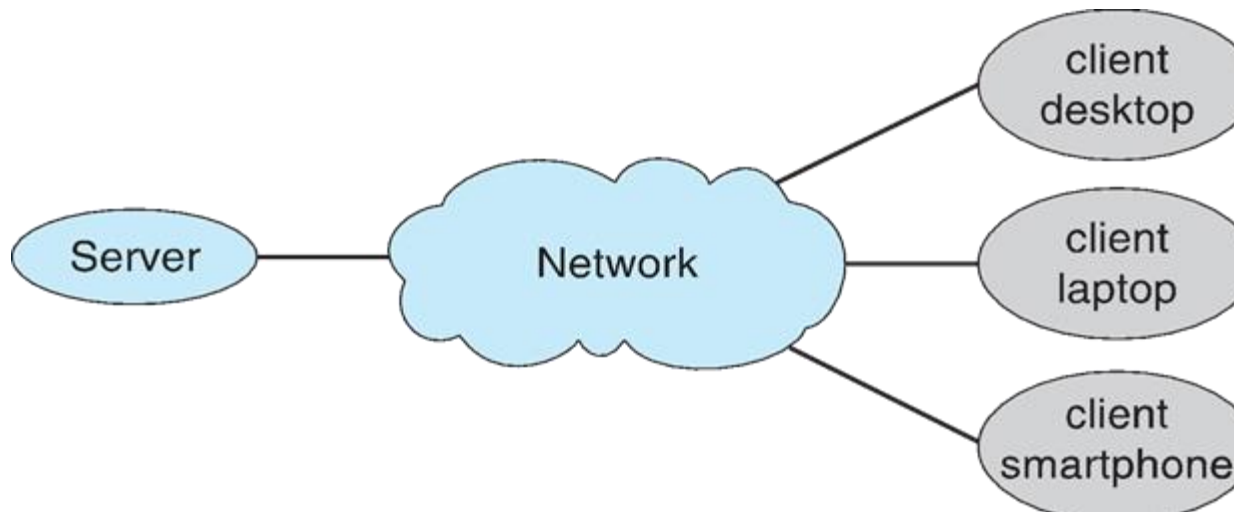


Computing Environments - Mobile

- Such as *handheld smartphones, tablets, etc.*
- What is the functional difference between them and a “traditional” laptop?
 - Extra feature – *more OS features* (e.g., GPS, gyroscope)
 - Allows *new types of apps* like **Augmented Reality (AR)**
 - Use IEEE 802.11 wireless, or cellular data networks for *connectivity*
- Leaders are **Apple iOS** and **Google Android**

- **Client-Server Computing**

- Dumb terminals supplanted by *smart PCs*
- Many systems now **servers**, responding to requests generated by **clients**
 - 4 **Compute-server system** provides an interface to client to request services (i.e., database)
 - 4 **File-server system** provides interface for clients to store and retrieve files



Computing Environments - Peer-to-Peer

- Another model of **distributed system**
- **P2P** does not distinguish clients and servers
 - Instead all nodes are considered **peers**
 - May each act as client, server or both
 - Node must join **P2P network**
 - 4 Registers its service with *central lookup service* on network, or
 - 4 Broadcast request for service and respond to requests for service via *discovery protocol*
- Examples include **Napster** and **Gnutella**, **Voice over IP (VoIP)** such as **Skype**

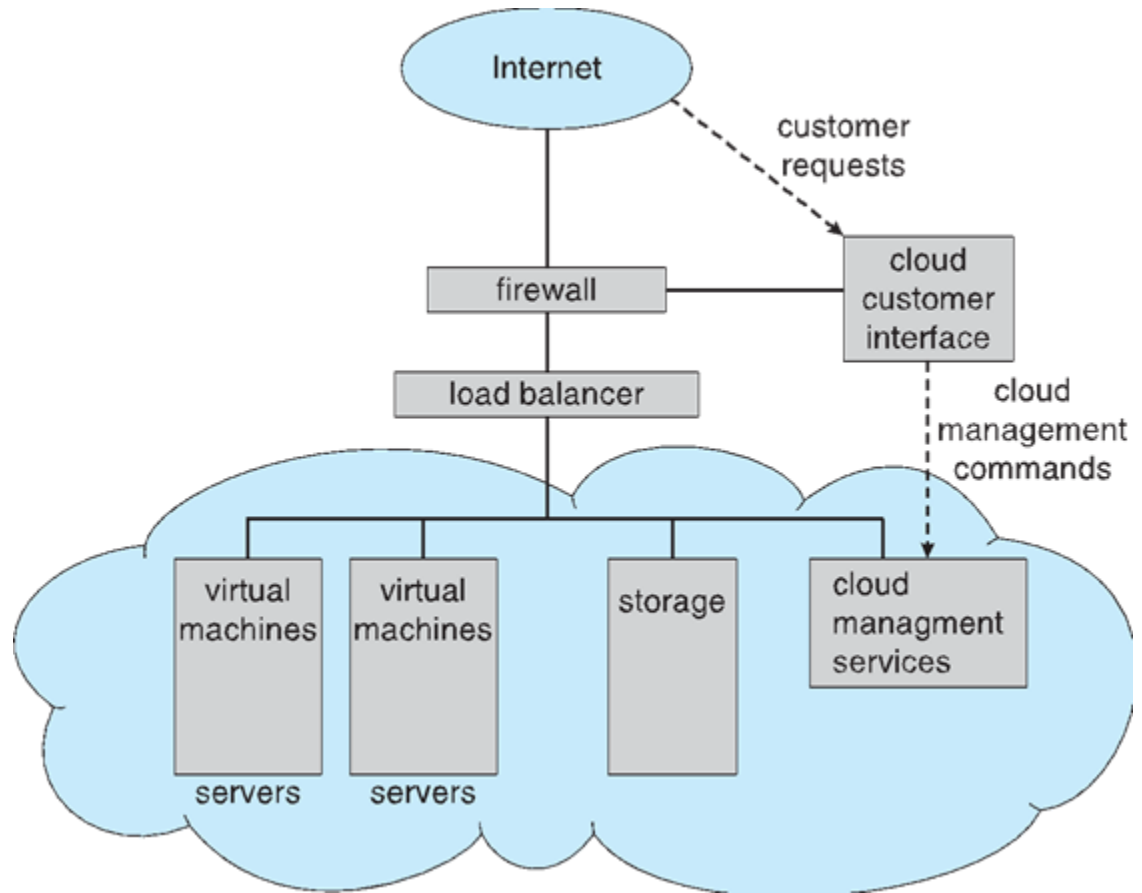


Computing Environments – Cloud Computing

- Delivers *computing*, *storage*, *apps* as a service across a network
- Logical extension of *virtualization* because it uses virtualization as the base for its functionality.
 - E.g., **Amazon EC2** has thousands of servers, millions of virtual machines, petabytes of storage available across the Internet
- Many types of **services**
 - **Software as a Service (SaaS)** – one or more applications available via the Internet
 - **Platform as a Service (PaaS)** – software stack ready for application use via the Internet
 - **Infrastructure as a Service (IaaS)** – servers or storage available over Internet (i.e., storage available for backup use)
- Many types of **structure**
 - **Public cloud** – available via Internet to anyone willing to pay
 - **Private cloud** – run by a company for the company's own use
 - **Hybrid cloud** – includes both public and private cloud components

Computing Environments – Cloud Computing

- Cloud computing environments composed of **traditional OSes**, plus **VMMs**, plus **cloud management tools**
 - Internet connectivity requires security like *firewalls*
 - *Load balancers* spread traffic across multiple applications



Real-Time Embedded Systems

- Real-time embedded systems most prevalent form of computers
 - Vary considerable, special purpose, limited purpose OS, **real-time OS**
- Many other special computing environments as well
 - Some have OSes, some perform tasks without an OS
- Real-time OS has well-defined fixed time constraints
 - Processing ***must*** be done within constraint
 - Correct operation only if constraints met



Free and Open-Source Operating Systems

- Operating systems made available in source-code format rather than just binary *closed-source* and *proprietary*
 - Counter to the *copy protection* and **Digital Rights Management (DRM)** movement
 - Started by **Free Software Foundation (FSF)**, which has “copyleft” **GNU Public License (GPL)** or **Lesser GPL (LGPL)**
 - Free software and open-source software are two different ideas championed by different groups of people
- 4 <http://gnu.org/philosophy/open-source-misses-the-point.html/>
- E.g., **GNU/Linux** and **BSD UNIX** (including **Darwin**, core of **Mac OS X**)
 - Use VMM like **VMware Player** (Free on Windows), **VirtualBox**
 - Use to run guest operating systems for exploration



The Study of Operating Systems

There has never been a more interesting time to study operating systems, and it has never been easier. The open-source movement has overtaken operating systems, causing many of them to be made available in both source and binary (executable) format. The list of operating systems available in both formats includes Linux, BSD UNIX, Solaris, and part of macOS. The availability of source code allows us to study operating systems from the inside out. Questions that we could once answer only by looking at documentation or the behavior of an operating system we can now answer by examining the code itself.

Operating systems that are no longer commercially viable have been open-sourced as well, enabling us to study how systems operated in a time of fewer CPU, memory, and storage resources. An extensive but incomplete list of open-source operating-system projects is available from https://curlie.org/Computers/Software/Operating_Systems/Open_Source/

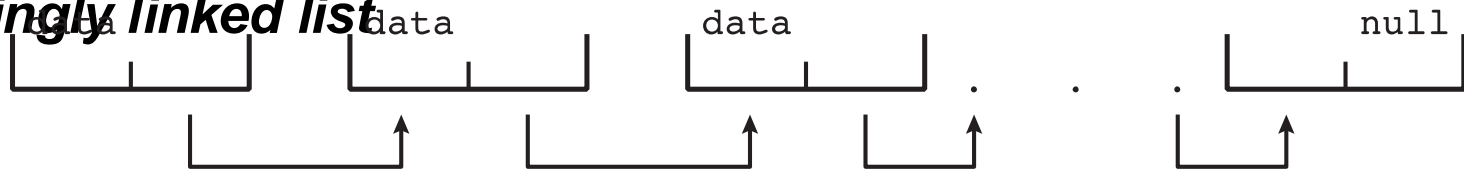
In addition, the rise of virtualization as a mainstream (and frequently free) computer function makes it possible to run many operating systems on top of one core system. For example, VMware (<http://www.vmware.com>) provides a free “player” for Windows on which hundreds of free “virtual appliances” can run. Virtualbox (<http://www.virtualbox.com>) provides a free, open-source virtual machine manager on many operating systems. Using such tools, students can try out hundreds of operating systems without dedicated hardware.

The advent of open-source operating systems has also made it easier to make the move from student to operating-system developer. With some knowledge, some effort, and an Internet connection, a student can even create a new operating-system distribution. Just a few years ago, it was difficult or impossible to get access to source code. Now, such access is limited only by how much interest, time, and disk space a student has.

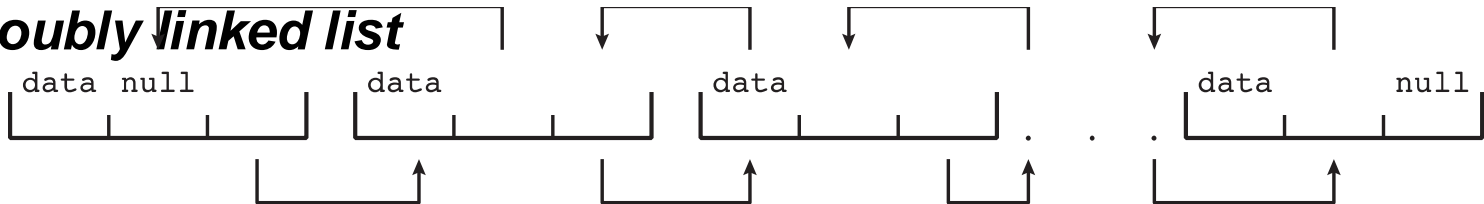
Kernel Data Structures

n Many similar to standard programming data structures

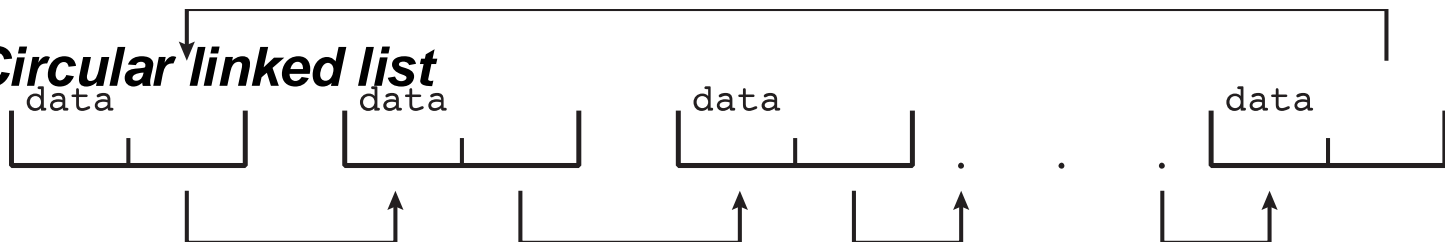
n ***Singly linked list***



n ***Doubly linked list***



n ***Circular linked list***

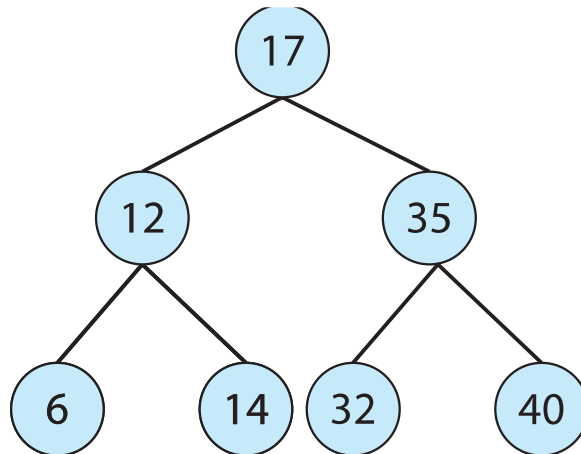


Kernel Data Structures

- **Binary search tree**

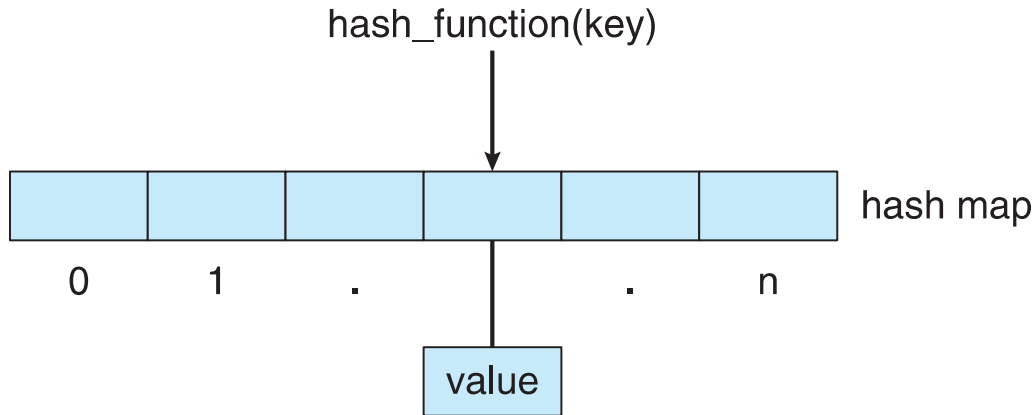
left \leq right

- Search performance is $O(n)$
- **Balanced binary search tree** is $O(\lg n)$



Kernel Data Structures

- **Hash function** can create a **hash map**



- **Bitmap** – string of n binary digits representing the status of n items
- Linux data structures defined in ***include*** files `<linux/list.h>`, `<linux/kfifo.h>`, `<linux/rbtree.h>`

End of Chapter 1

