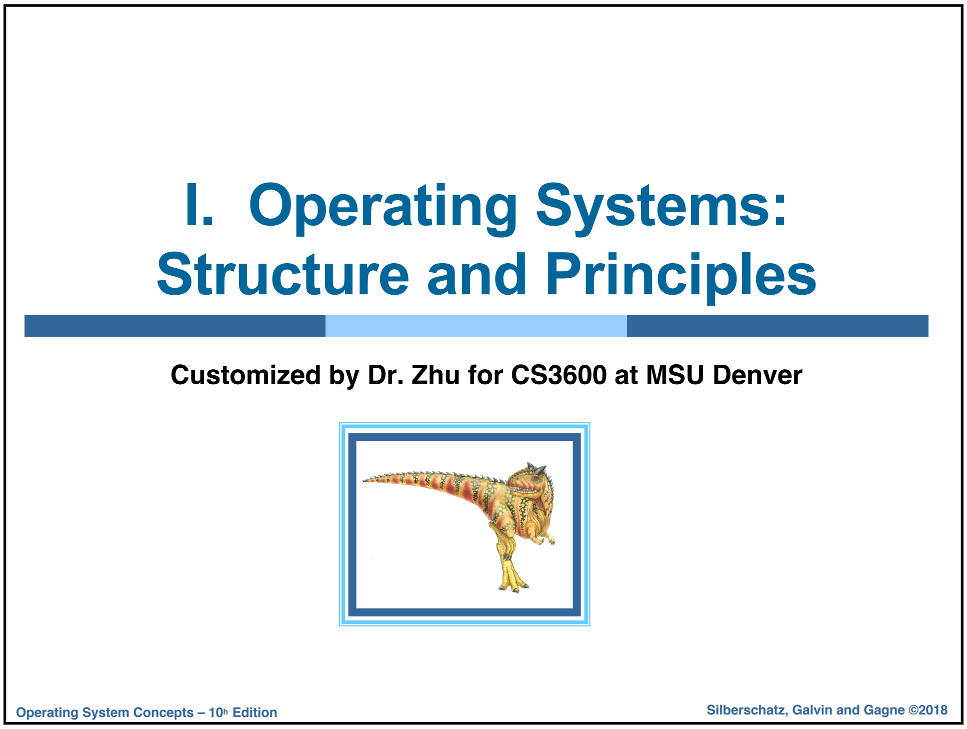
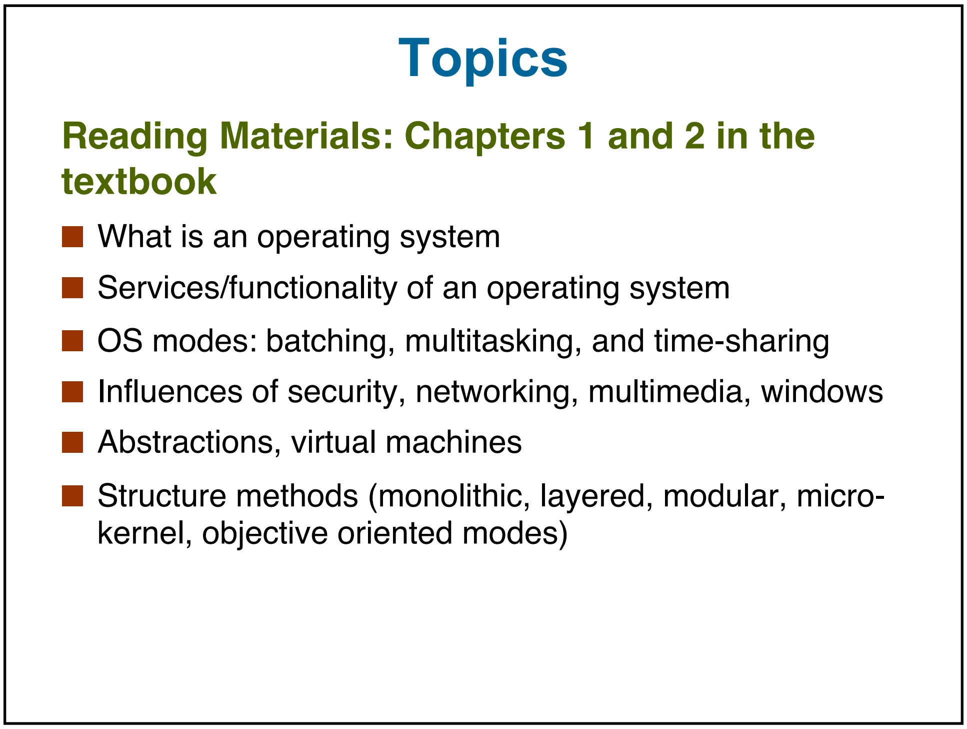


# I. Operating Systems: Structure and Principles

**Customized by Dr. Zhu for CS3600 at MSU Denver**

Operating System Concepts – 10th Edition

1



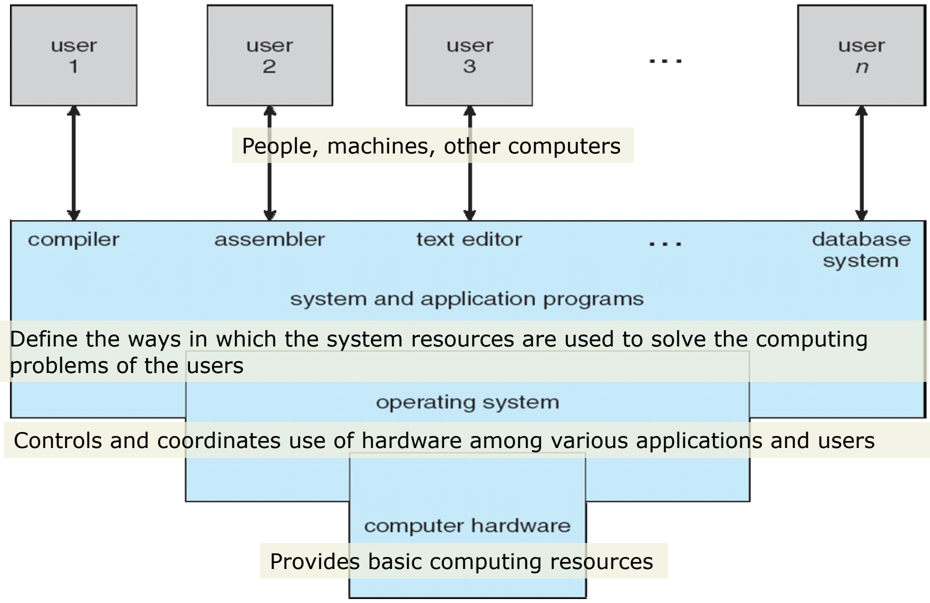
## Topics

### Reading Materials: Chapters 1 and 2 in the textbook

- What is an operating system
- Services/functionality of an operating system
- OS modes: batching, multitasking, and time-sharing
- Influences of security, networking, multimedia, windows
- Abstractions, virtual machines
- Structure methods (monolithic, layered, modular, micro-kernel, objective oriented modes)

2

## Four Components of a Computer System



People, machines, other computers

Define the ways in which the system resources are used to solve the computing problems of the users

Controls and coordinates use of hardware among various applications and users

Provides basic computing resources

3

## Operating System Definition

- 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

4

## Operating System Definition (Cont.)

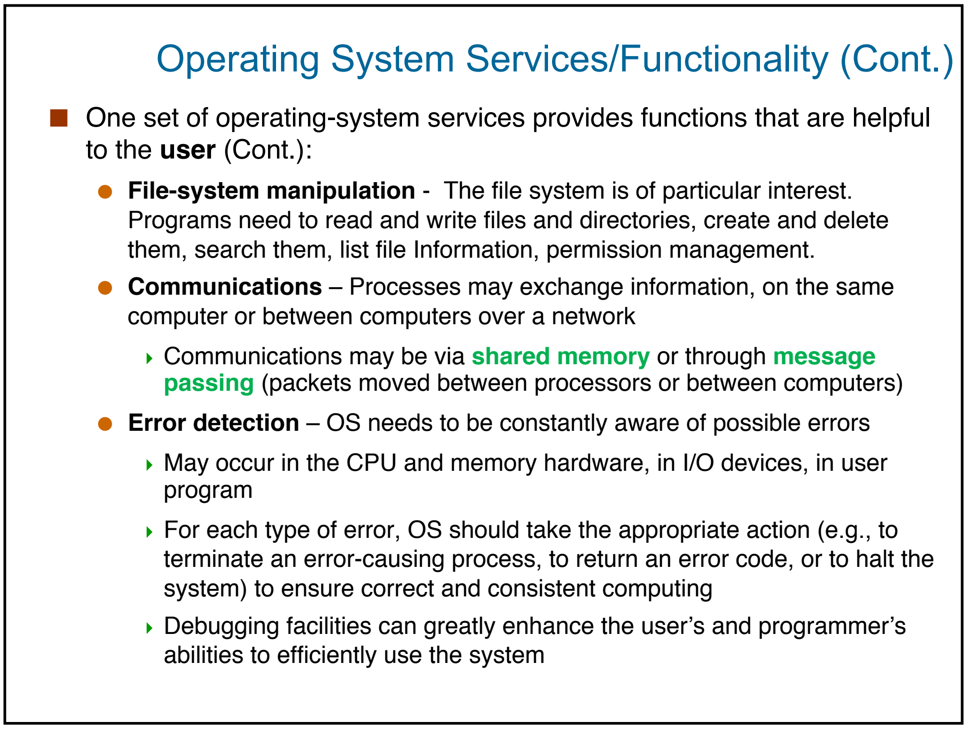
- No universally accepted definition
- "Everything a vendor ships when you order an operating system" is a good approximation
  - But varies wildly
- "The one program running at all times on the computer" is the **kernel**.
- Everything else is either
  - a **system program** (ships with the operating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

5

## Operating System Services/Functionality

- **Operating systems provide an environment for execution of programs and services to programs and users**
- One set of operating-system services provides functions that are helpful to the **user**:
  - **User interface** - Almost all operating systems have a user interface (**UI**).
    - Varies between **Batch** (early computers were batch systems processing jobs in bulk w/ predetermined input from files or other data sources), **Command-Line** **(CLI**, text commands and a method for entering text**)**, **Graphics User Interface** **(GUI**, a window system w/ a mouse**)**, **Touch-Screen Interface** (allow user to slide fingers across the screen or press buttons on the screen)
  - **Program execution** - The system must be able to **load** a program into memory, to **run** that program, and to **end** execution, either normally or abnormally (indicating error)
  - **I/O operations** - A running program may require I/O, which may involve a file or an I/O device while users usually cannot directly control I/O devices.

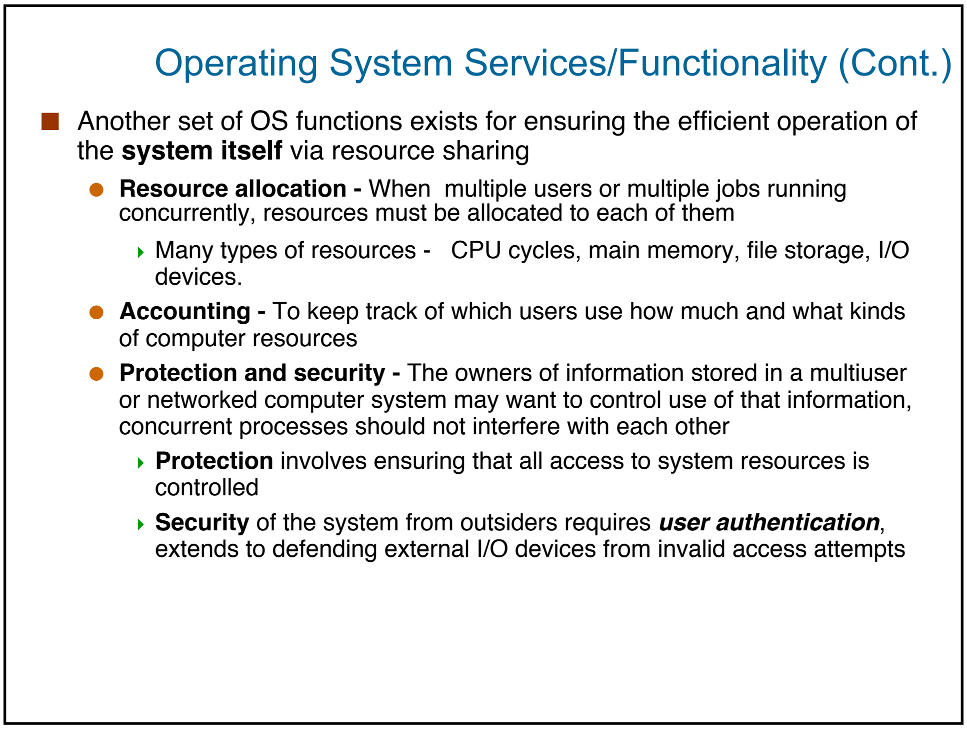
6



## Operating System Services/Functionality (Cont.)

- One set of operating-system services provides functions that are helpful to the **user** (Cont.):
  - **File-system manipulation** - The file system is of particular interest. Programs need to read and write files and directories, create and delete them, search them, list file Information, permission management.
  - **Communications** – Processes may exchange information, on the same computer or between computers over a network
    - Communications may be via **shared memory** or through **message passing** (packets moved between processors or between computers)
  - **Error detection** – OS needs to be constantly aware of possible errors
    - May occur in the CPU and memory hardware, in I/O devices, in user program
    - For each type of error, OS should take the appropriate action (e.g., to terminate an error-causing process, to return an error code, or to halt the system) to ensure correct and consistent computing
    - Debugging facilities can greatly enhance the user's and programmer's abilities to efficiently use the system

7

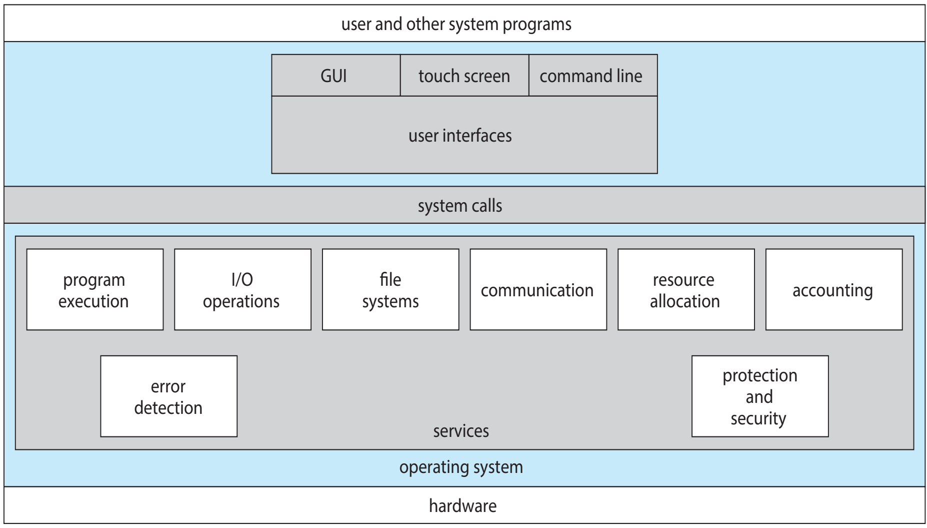


## Operating System Services/Functionality (Cont.)

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

8

## A View of Operating System Services



user and other system programs

GUI | touch screen | command line

user interfaces

system calls

program execution | I/O operations | file systems | communication | resource allocation | accounting

error detection | protection and security

services

operating system

hardware

9

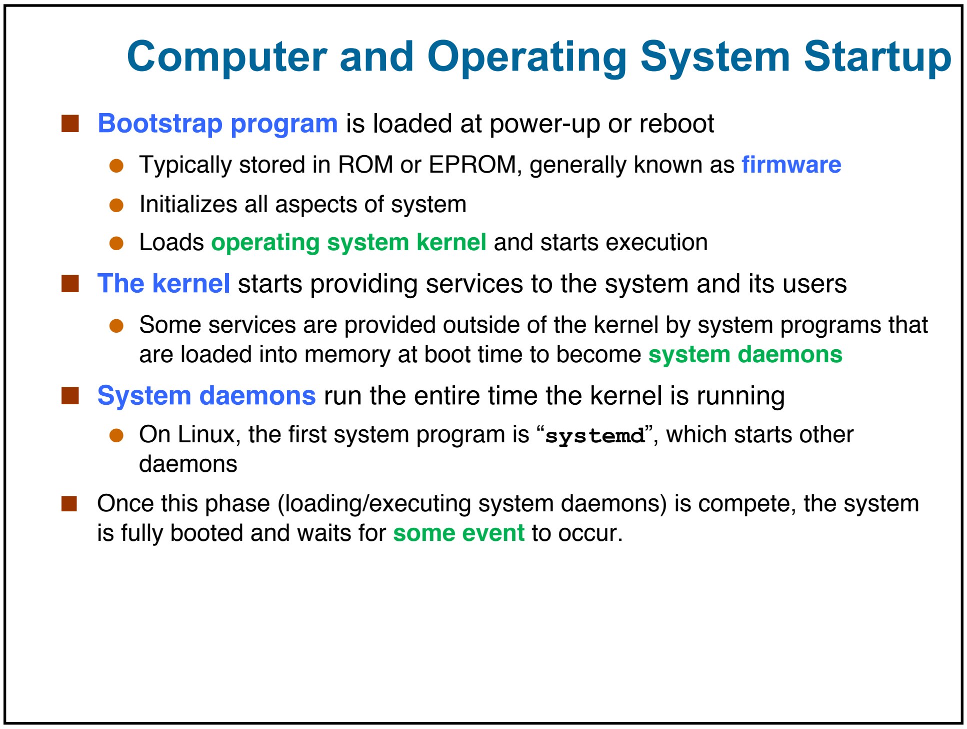
## System Calls

- Demo on cs3600a.msudenver.edu
  - Enter a directory of your choice and assume there is a file named **`srcFile.txt`**
  - Run the following command to display all the system calls to copy a source file to a destination file

```
strace cp srcFile.txt dstFile.txt
```

  - Linux systems provide the **`strace`** utility to trace system calls
  - Read Section 2.3 for more details on “System Calls”

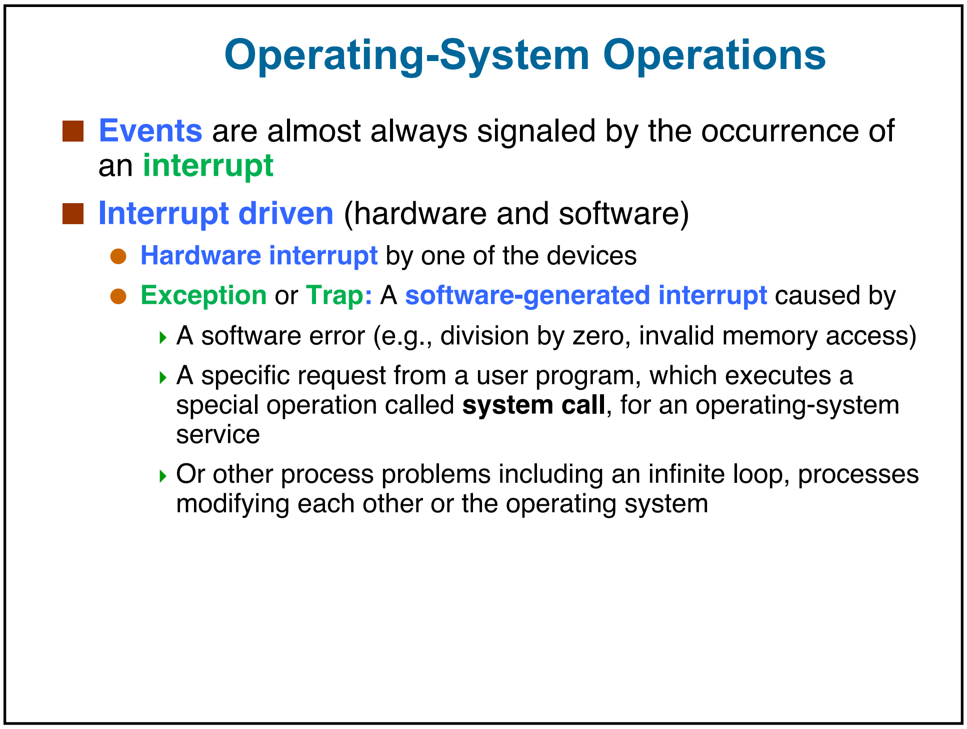
10



## Computer and Operating System Startup

- **Bootstrap program** is loaded at power-up or reboot
  - Typically stored in ROM or EPROM, generally known as **firmware**
  - Initializes all aspects of system
  - Loads **operating system kernel** and starts execution
- **The kernel** starts providing services to the system and its users
  - Some services are provided outside of the kernel by system programs that are loaded into memory at boot time to become **system daemons**
- **System daemons** run the entire time the kernel is running
  - On Linux, the first system program is "`systemd`", which starts other daemons
- Once this phase (loading/executing system daemons) is compete, the system is fully booted and waits for **some event** to occur.

11



## Operating-System Operations

- **Events** are almost always signaled by the occurrence of an **interrupt**
- **Interrupt driven** (hardware and software)
  - **Hardware interrupt** by one of the devices
  - **Exception** or **Trap:** A **software-generated interrupt** caused by
    - A software error (e.g., division by zero, invalid memory access)
    - A specific request from a user program, which executes a special operation called **system call**, for an operating-system service
    - Or other process problems including an infinite loop, processes modifying each other or the operating system

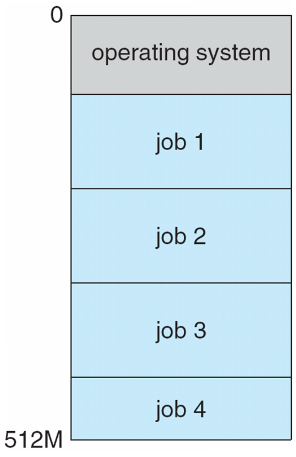
12

## Operating-System Operations (Cont.)

- **Multiprogramming** (**Batch system**) needed for efficiency
  - A single user/program cannot keep CPU and I/O devices busy at all times
  - Users typically *want* to run more than one program at a time
  - 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 it has to wait (for I/O for example), OS switches to another job
- **Timesharing** (**multitasking**) is logical extension in which 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 fit in memory

13

## Memory Layout for Multiprogrammed System



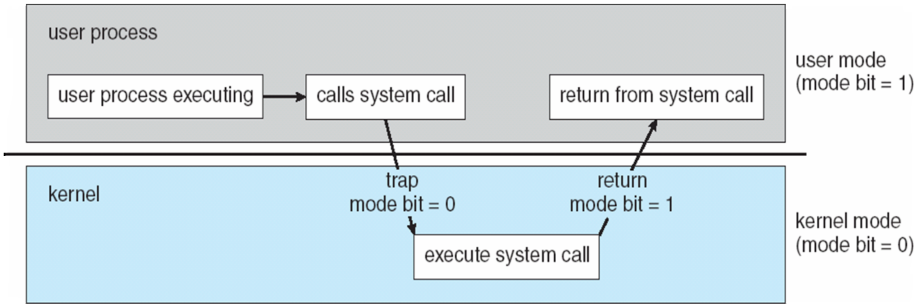
14

## Operating-System Modes

- **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
    - Some instructions designated as **privileged**, only executable in **kernel** mode
    - **System call** changes mode to **kernel**, return from call resets it to **user**
- Increasingly CPUs support **multi-mode** operations, e.g.,
  - ARM general processors except in the M-profile: ARMv4, ARMv7-A, etc.
  - **virtual machine manager** (**VMM**) mode for guest **VMs**

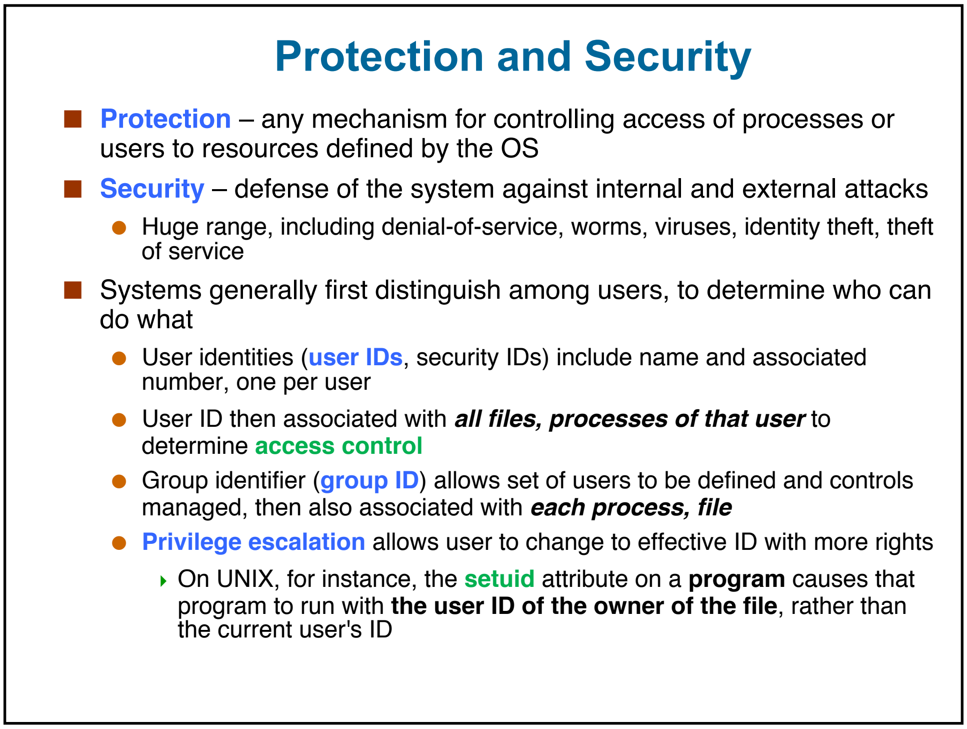
15

## Transition from User to Kernel Mode



- **Timer** to prevent infinite loop / 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

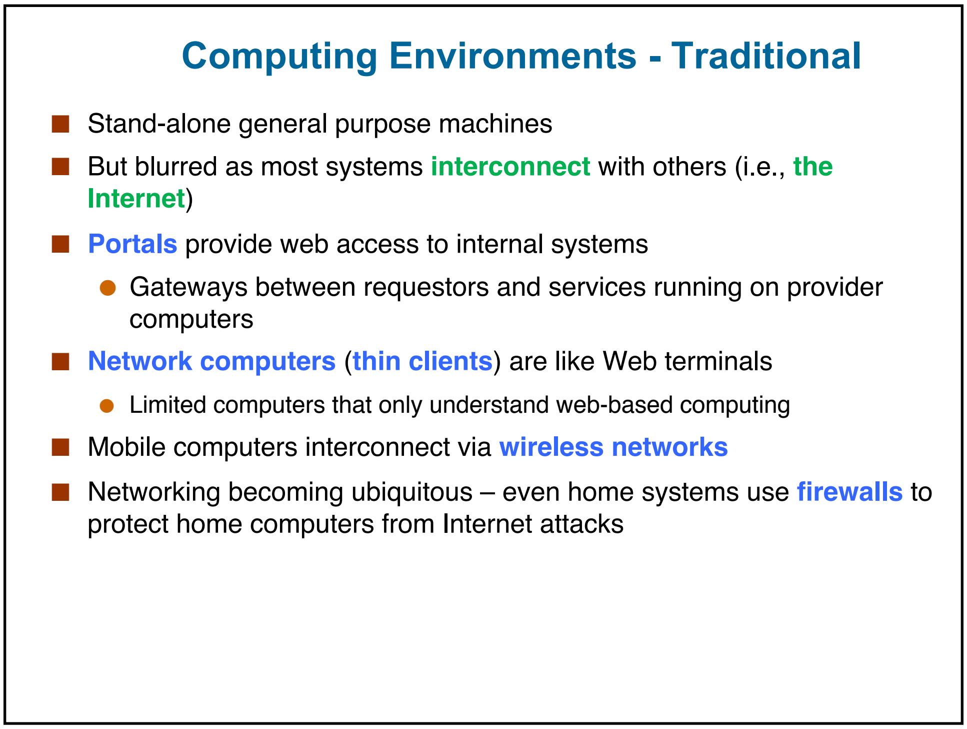
16



## 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
  - User identities (**user IDs**, security IDs) include name and associated number, one per user
  - User ID then associated with ***all files, processes of that user*** to determine **access control**
  - Group identifier (**group ID**) allows set of users to be defined and controls managed, then also associated with ***each process, file***
  - **Privilege escalation** allows user to change to effective ID with more rights
    - On UNIX, for instance, the **setuid** attribute on a **program** causes that program to run with **the user ID of the owner of the file**, rather than the current user's ID

17



## 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
  - Gateways between requestors and services running on provider computers
- **Network computers** (**thin clients**) are like Web terminals
  - Limited computers that only understand web-based computing
- Mobile computers interconnect via **wireless networks**
- Networking becoming ubiquitous – even home systems use **firewalls** to protect home computers from Internet attacks

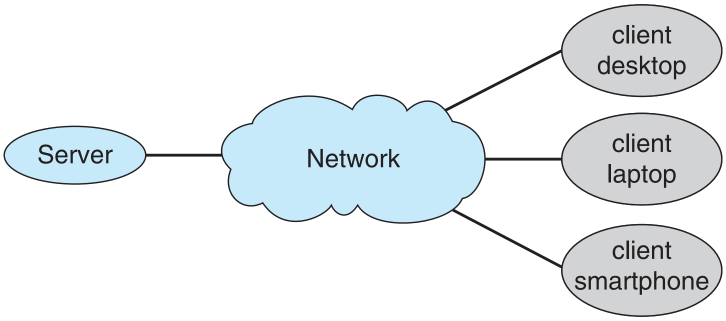
18

## Computing Environments - Mobile

- Handheld smartphones, tablets, etc.
- The functional difference between them and a "traditional" laptop?
  - Features on mobile devices have become so rich that the distinction in functionality between may be difficult to discern
  - In fact, a contemporary mobile device can now provide functionality that is either unavailable or impractical on a laptop computer
- Extra feature – more OS features (GPS, gyroscope)
- Allows new types of apps like ***augmented reality***
- Use IEEE 802.11 wireless, or cellular data networks for connectivity
- Leaders are **Apple iOS** and **Google Android**

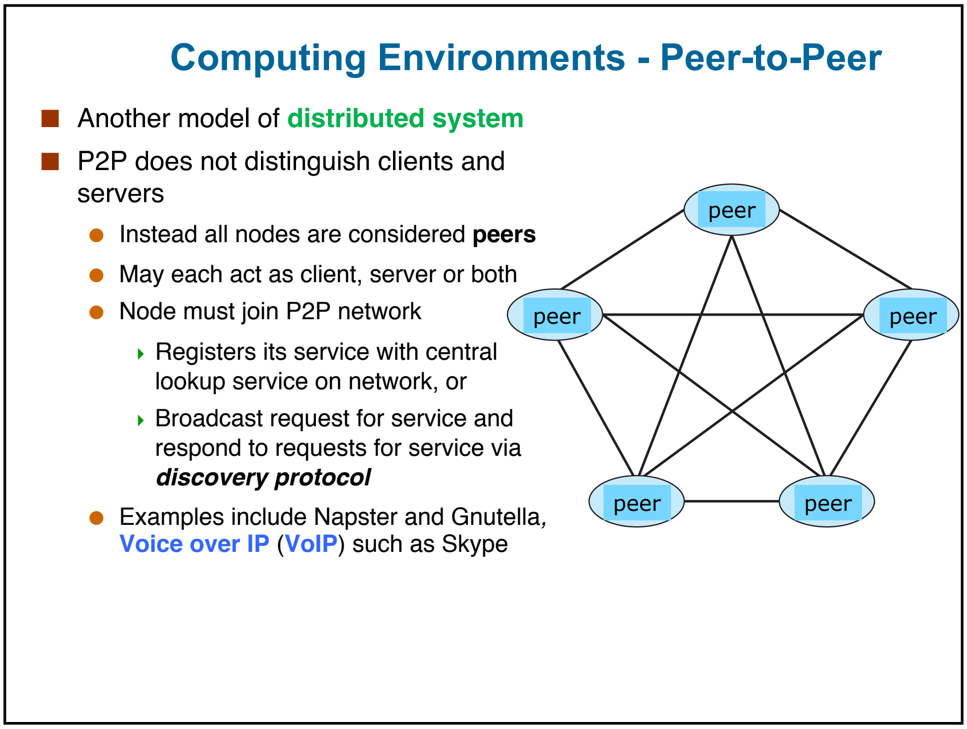
19

## Computing Environments – Client-Server



- Client-Server Computing: a form of specialized **distributed computing**
  - Dumb terminals supplanted by smart PCs
  - Many systems now **servers**, responding to requests generated by **clients**
    - **Compute-server system** provides an interface to client to request services (i.e., database)
    - **File-server system** provides interface for clients to store and retrieve files

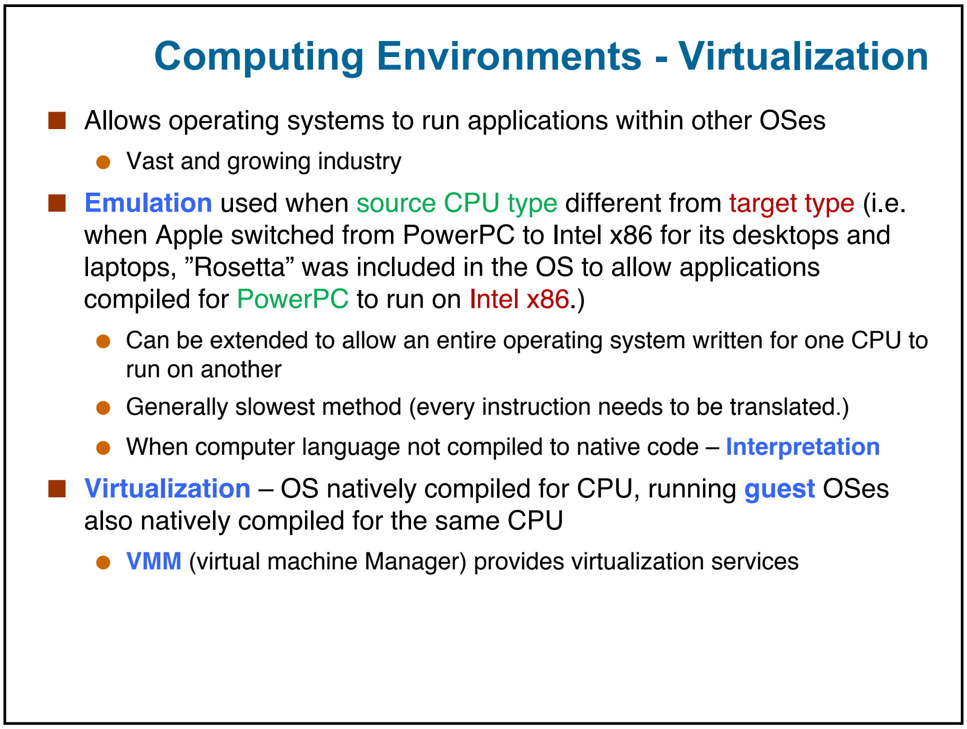
20



## 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
    - Registers its service with central lookup service on network, or
    - 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

21



## Computing Environments - Virtualization

- Allows operating systems to run applications within other OSes
  - Vast and growing industry
- **Emulation** used when source CPU type different from target type (i.e. when Apple switched from PowerPC to Intel x86 for its desktops and laptops, ”Rosetta” was included in the OS to allow applications compiled for PowerPC to run on Intel x86.)
  - Can be extended to allow an entire operating system written for one CPU to run on another
  - Generally slowest method (every instruction needs to be translated.)
  - When computer language not compiled to native code – **Interpretation**
- **Virtualization** – OS natively compiled for CPU, running **guest** OSes also natively compiled for the same CPU
  - **VMM** (virtual machine Manager) provides virtualization services

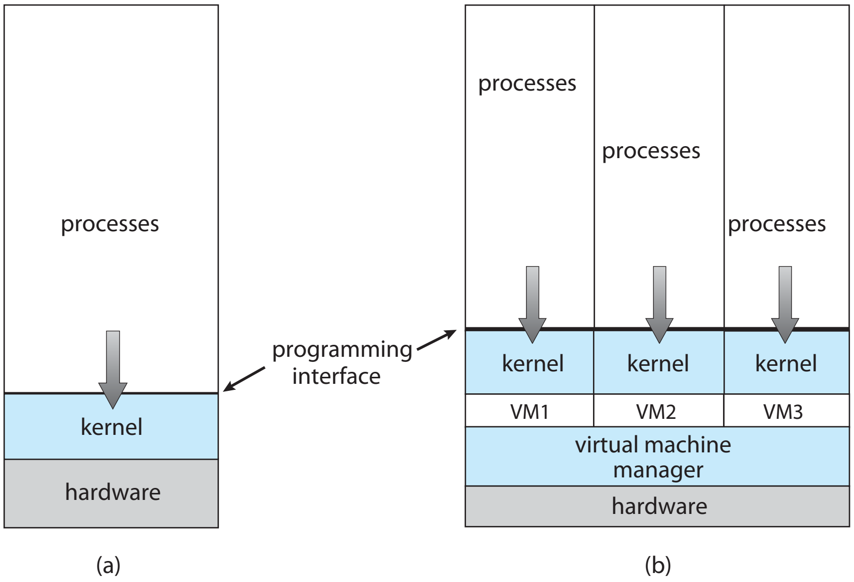
22

## Virtualization (Cont.)

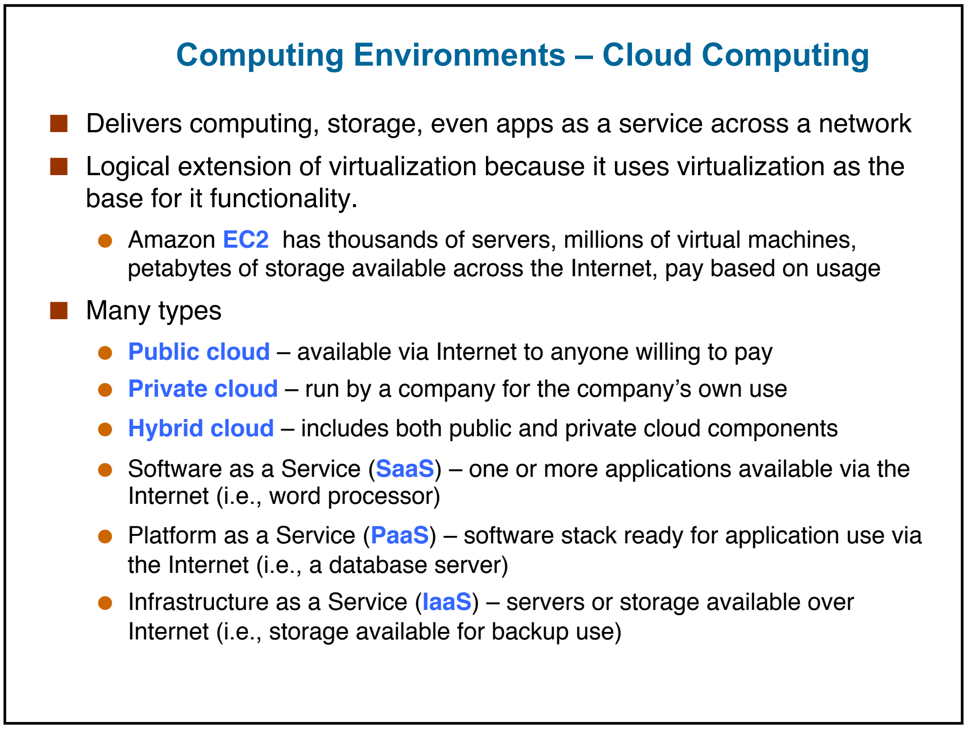
- Use cases involve laptops and desktops running multiple OSes for exploration or compatibility
  - Apple laptop running Mac OS X **host**, Windows 10 as a **guest**
  - Developing apps for multiple OSes without having multiple systems
  - QA testing applications without having multiple systems
  - Executing and managing compute environments within data centers
- Some VMMs like *VMware ESX* and *Citrix XenServer* no longer run on host operating systems but rather are the host operating systems
  - Provide services and resource management to virtual machine processes
  - There is no general purpose host then

23

## Virtualization w/o General Purpose Host OS



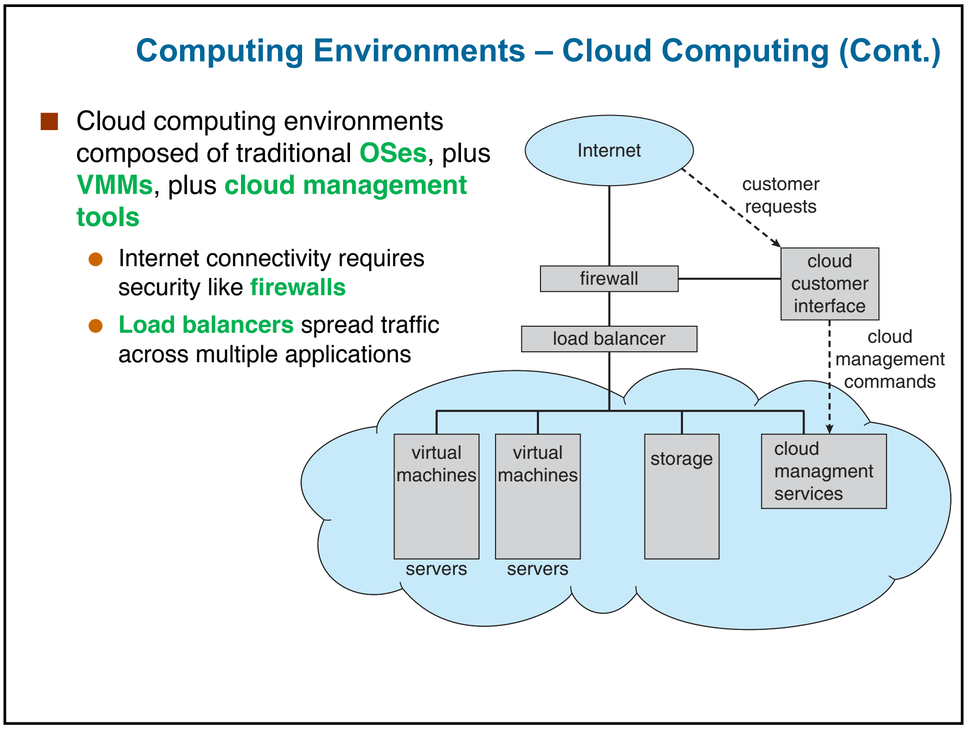
24



## Computing Environments – Cloud Computing

- Delivers computing, storage, even apps as a service across a network
- Logical extension of virtualization because it uses virtualization as the base for it functionality.
  - Amazon **EC2** has thousands of servers, millions of virtual machines, petabytes of storage available across the Internet, pay based on usage
- Many types
  - **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
  - Software as a Service (**SaaS**) – one or more applications available via the Internet (i.e., word processor)
  - Platform as a Service (**PaaS**) – software stack ready for application use via the Internet (i.e., a database server)
  - Infrastructure as a Service (**IaaS**) – servers or storage available over Internet (i.e., storage available for backup use)

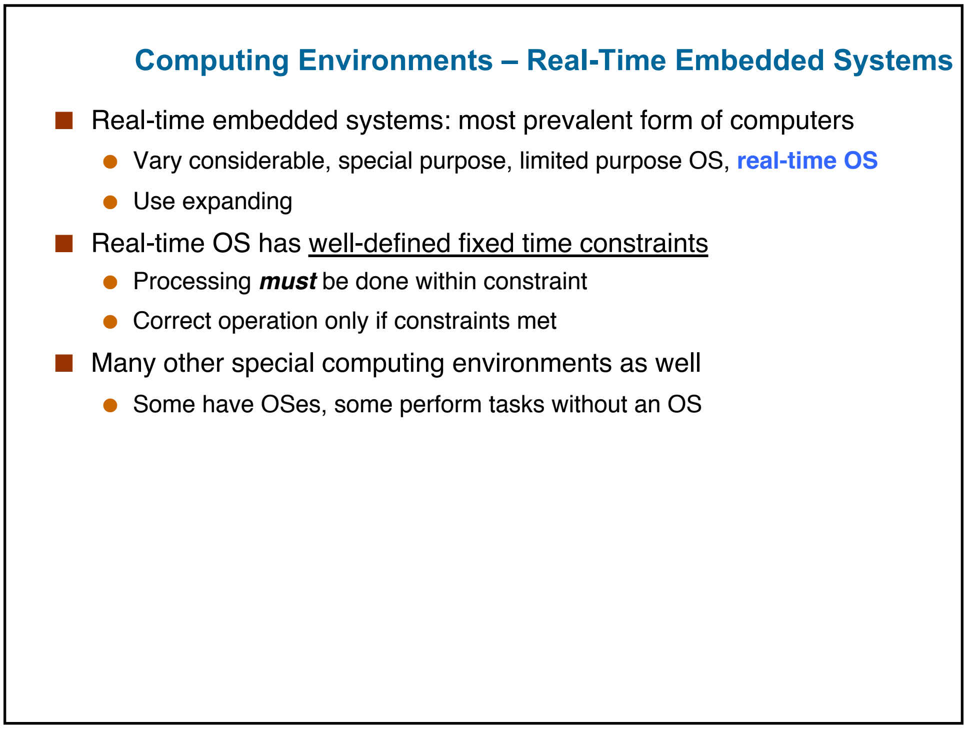
25



## Computing Environments – Cloud Computing (Cont.)

- 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

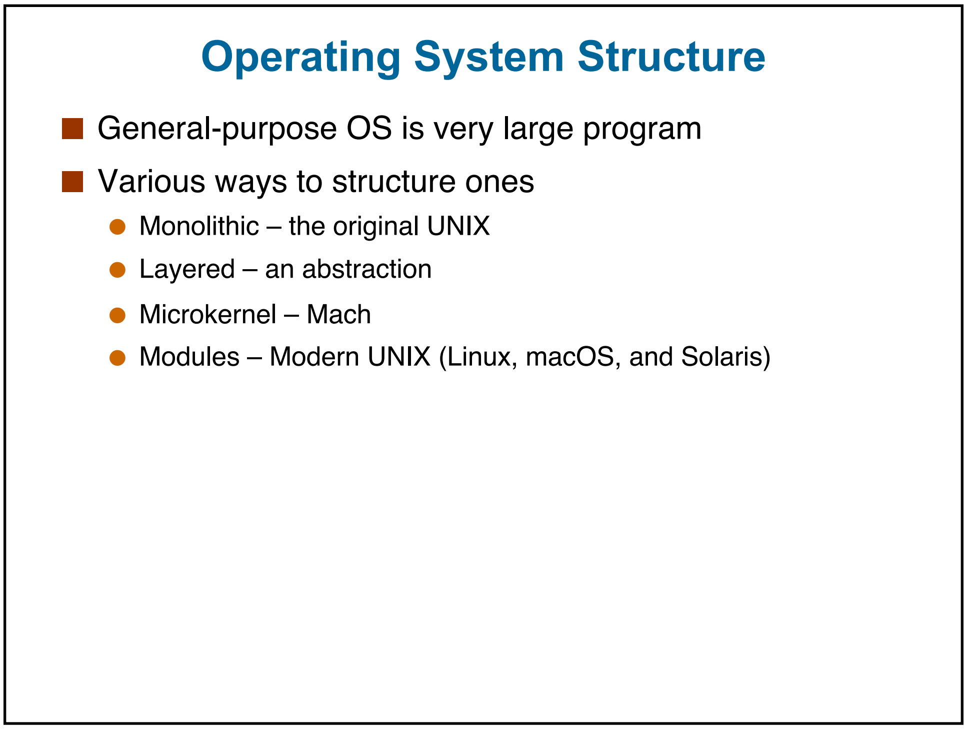
26



## Computing Environments – Real-Time Embedded Systems

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

27



# Operating System Structure

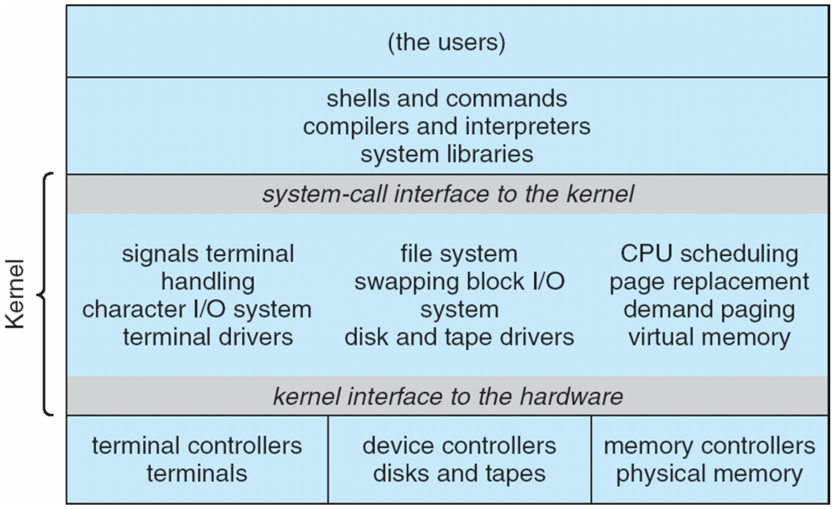
- General-purpose OS is very large program
- Various ways to structure ones
  - Monolithic – the original UNIX
  - Layered – an abstraction
  - Microkernel – Mach
  - Modules – Modern UNIX (Linux, macOS, and Solaris)

28

## Monolithic Structure -- UNIX

- **Monolithic structure**: place all of the functionality of the kernel into a single, static binary file that runs in a single address space.
  - tightly **coupled system:** changes to one part of the system can have wide-ranging effects on other parts
  - a **distinct performance advantage**: very little overhead in the system-call interface, and fast communication within the kernel
- **UNIX** – limited by hardware functionality, the original UNIX operating system has such limited structure and consists of two separable parts
  - **System programs**
  - **The kernel**
    - Consists of everything *below* the system-call interface and *above* the physical hardware
    - Provides the *file system*, *CPU scheduling*, *memory management*, and *other* operating-system functions
    - Combine an enormous amount of functionality into *one single address space* in *one level*

## Traditional UNIX System Structure



(the users)

shells and commands
compilers and interpreters
system libraries

*system-call interface to the kernel*

| | | |
|---|---|---|
| signals terminal handling<br>character I/O system<br>terminal drivers | file system<br>swapping block I/O system<br>disk and tape drivers | CPU scheduling<br>page replacement<br>demand paging<br>virtual memory |

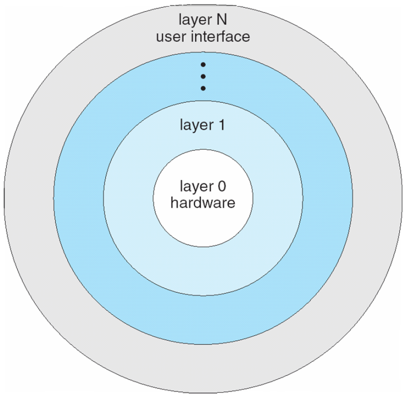
*kernel interface to the hardware*

| | | |
|---|---|---|
| terminal controllers<br>terminals | device controllers<br>disks and tapes | memory controllers<br>physical memory |

- Beyond simple but not fully layered
- As UNIX expanded, the kernel becomes large and difficult to manage

## Layered Approach for Modularity

- The operating system is divided into a number of layers, each built on top of lower layers.
  - The bottom layer (layer 0), is the hardware; the highest (layer N) is the user interface.
  - Each layer consists of data structures and a set of functions that can be invoked by higher layers
  - Each layer can invoke operations on lower layers.
  - Main advantage: simplicity of construction and debugging



- Layered systems have successfully used in computer networks (such as TCP/IP) and web applications. Few operating systems use a pure layered approach.
  - Challenging to appropriately define the functionality of each layer
  - Poor overall performance due to the overhead of requiring a user program to traverse through multiple layers to obtain an OS service

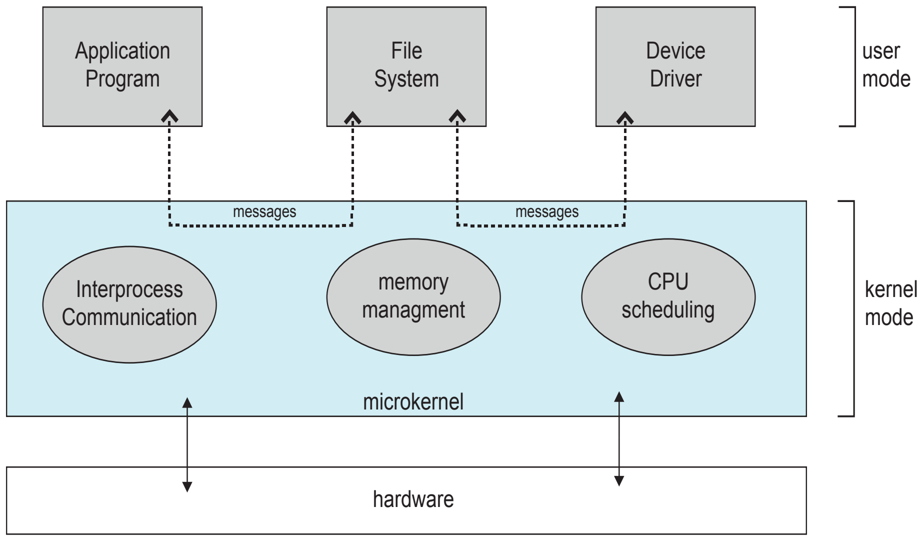
31

## Microkernel System Structure

- Remove all nonessential components from the kernel and implement them as user-level programs that reside in separate address spaces
- **Mach**: proposed in the mid-1980s by researchers to modularize the kernel using the **microkernel** approach
  - Mac OS X kernel (**Darwin**) for **macOS** and **iOS**: partly based on **Mach**
    - the best-known illustration of microkernel
- User modules never directly communicate, but indirectly by exchanging messages with the microkernel via **message passing**
- Benefits:
  - Easier to extend a microkernel
  - Easier to port the operating system to new architectures
  - More reliable (less code is running in kernel mode)
  - More secure
- Detriments:
  - Performance overhead of user space to kernel space communication

32

## Microkernel System Structure (Cont.)

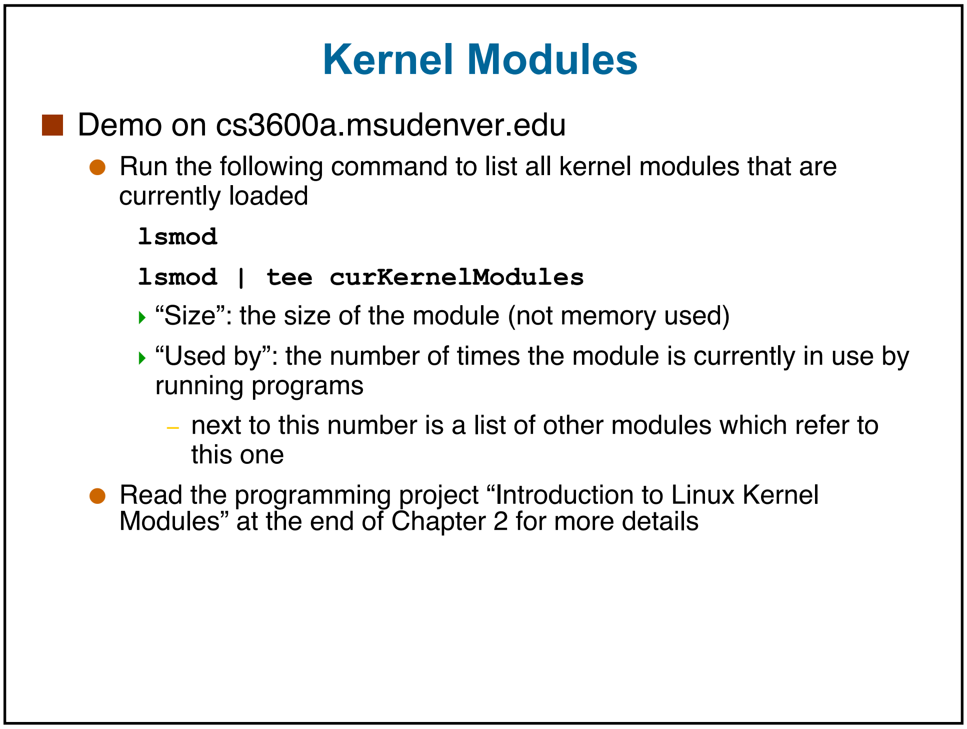


33

## Object-Oriented Modules

- Many modern operating systems implement **loadable kernel modules**
  - The kernel has a set of core components and can link in additional services via modules, at boot time or during run time.
  - An object-oriented approach
  - Each core component is separate
  - Each talks to the others over **known interfaces**
  - Each is loadable **as needed** within the kernel
- Like a **layered** system, each module has defined, protected interfaces
  - but more flexible, because any module can call any other module.
- Like the **microkernel** approach, the primary module has only core functions and knowledge of load and communicate with other modules
  - but more efficient, because modules do not need to invoke message passing via the kernel in order to communicate.
  - Modern UNIX such as Linux, Solaris, and macOS, as well as Windows

34



## Kernel Modules

- Demo on cs3600a.msudenver.edu
  - Run the following command to list all kernel modules that are currently loaded

    ```
    lsmod
    lsmod | tee curKernelModules
    ```

    - “Size”: the size of the module (not memory used)
    - “Used by”: the number of times the module is currently in use by running programs
      - next to this number is a list of other modules which refer to this one
  - Read the programming project “Introduction to Linux Kernel Modules” at the end of Chapter 2 for more details

35