CSE308 Operating Systems

Introduction to Operating Systems

S.Rajarajan
SoC
SASTRA

UNIT - I **11 Periods**

Introduction: What Operating systems do - Structure - Operations - Process management - **Operating system Structure:** Services - User Interfaces - Systems calls and types - Systems programs - Design and implementation - Operating System Structure - System boot

UNIT - II **11 Periods**

Process management: Process concept - Scheduling - Operations on processes - Interprocess communication - **Threads:** Overview - Multi-core programming - Multithreading models - **Process scheduling:** Basic concepts - Scheduling criteria - Algorithms: FCFS - SJFS - Priority - RR - Multilevel queue - Multilevel feedback - Thread scheduling: Contention scope - Pthread scheduling - Multiple-processor scheduling - **Process synchronization:** Background - Critical section problem - Peterson's solution - Synchronization hardware - Mutex locks - Semaphore - Classic problems of synchronization

UNIT - III **11 Periods**

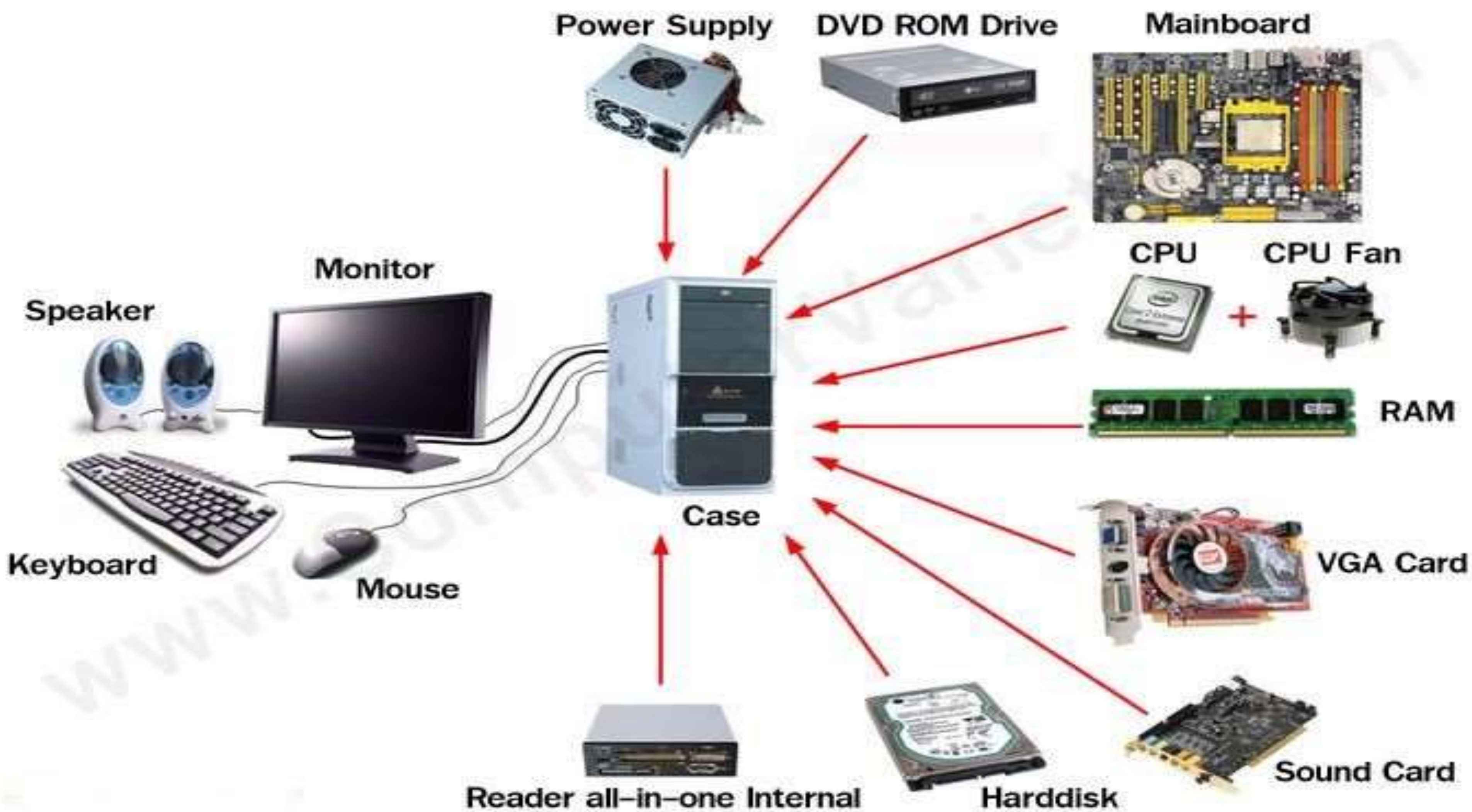
Deadlocks : System model - Characterization - Methods for handling deadlock - Prevention - Avoidance - Detection - Recovery from deadlocks - **Memory management:** Background - Swapping - Contiguous memory allocation - Paging - Structure of page tables - Segmentation - **Virtual memory:** Background - Demand paging - Copy-on-write - Page replacement - Allocation of frames - Thrashing

UNIT - IV **12 Periods**

File system: File concept - Access methods - Directory and disk structure - File system mounting - Protection - File allocation methods - **Mass Storage:** Magnetic disks - Disk structure - Disk scheduling: FCFS - SSTF - SCAN - C-SCAN - LOOK - Selection of an algorithm - **I/O systems:** Overview - I/O hardware - Application I/O interface - Kernel I/O subsystem

Components of a Computer

- Computer system 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


Power Supply
DVD ROM Drive
Mainboard
CPU
CPU Fan
Monitor
Speaker
RAM
Case
Keyboard
Mouse
VGA Card
Reader all-in-one Internal
Harddisk
Sound Card

What operating systems do ?

- Operating system provides the means for **proper use of these resources**
- An operating system is **similar to a government**
- It offers no direct service to users
- But provides an ***environment within which other programs can do*** useful work.

Use of a computer

- **End-user**
  - Browser, Word processor, Paint, Games, Spreadsheet etc.
- **Programmer**
  - IDE, Compiler, Memory, Hardware, Files, Input and Output
- **System administrator**
  - User accounts, Privileges, Protection and Security , Authentication, Networking, Database
- Operating systems facilitates all these services by abstracting the complexities

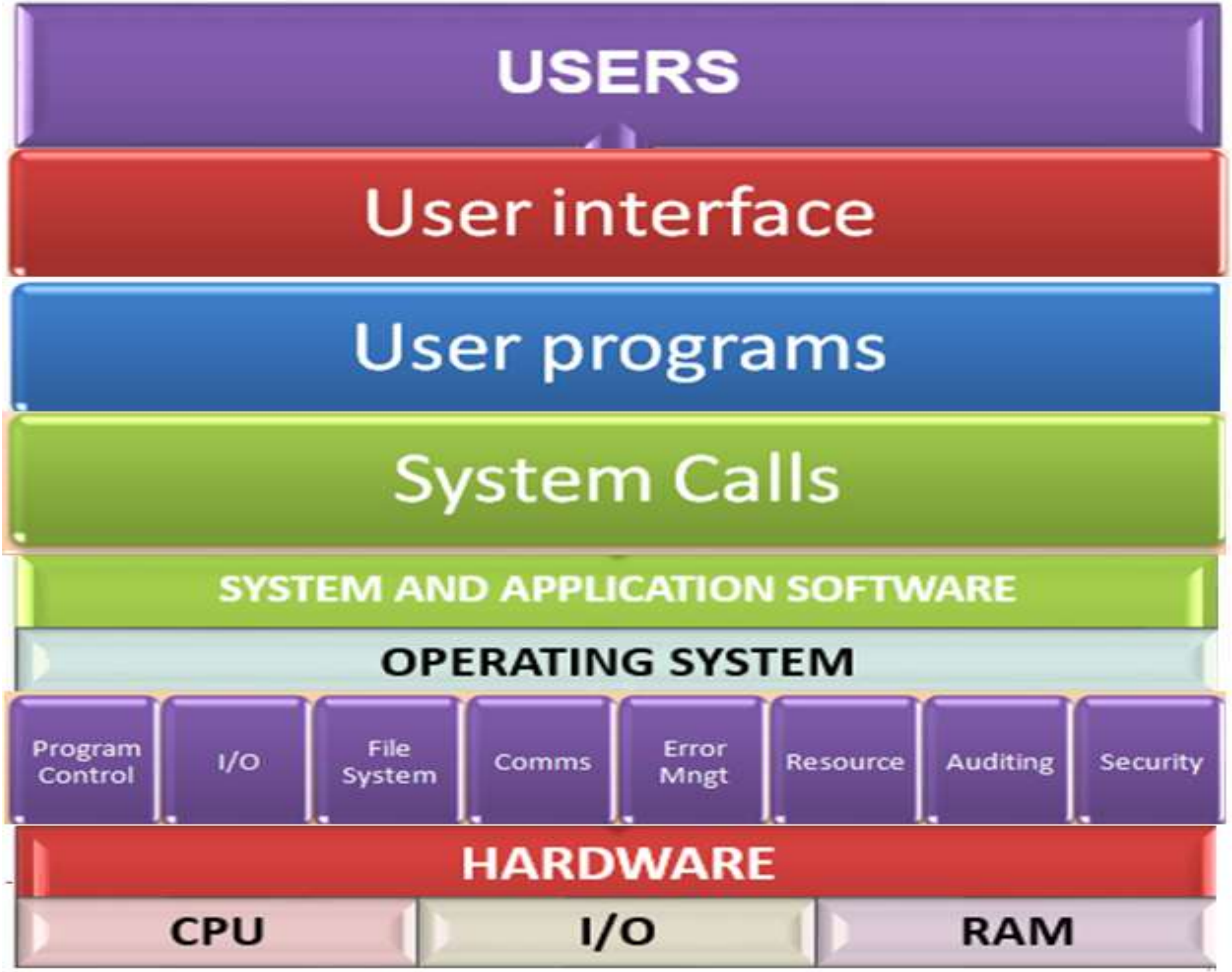
Computer Hardware

- **Processor**
  - CPU
  - GPU
  - I/O processor
  - Math co-processor
- **Memory**
  - Registers
  - Main memory (RAM)
  - Cache memory
  - Secondary memory(hard disk)

- **I/O devices**
  – Keyboard
  – Mouse
  – Monitor
  – Printer
- **Other components**
  – DMA
  – BUS
  – MMU

What is an Operating System ?

- It is a system software that
  - acts as an **interface** between hardware and user applications
  - **Manages** computer resources
  - **Enhances** the performance of computer
  - **Facilitates** the development and execution of programs/applications.
  - **Provides** file management services
- **System software** is a software that provides platform to other software


USERS
User interface
User programs
System Calls
SYSTEM AND APPLICATION SOFTWARE
OPERATING SYSTEM
Program Control
I/O
File System
Comms
Error Mngt
Resource
Auditing
Security
HARDWARE
CPU
I/O
RAM

Views of OS

- **User view**
  - varies **according to the interface** being used
  - **PC** operating system is designed mostly for **ease of use**
  - **Mainframe** OS is designed to **maximize resource utilization** - to assure that all available CPU time, memory, and I/O are used efficiently
  - **Workstations** connected to networks of other workstations and **servers**
  - **Embedded computers** little or no user view
  - Embedded computers in home devices and automobiles may have **numeric keypads** and may turn **indicator lights on or off** to show status

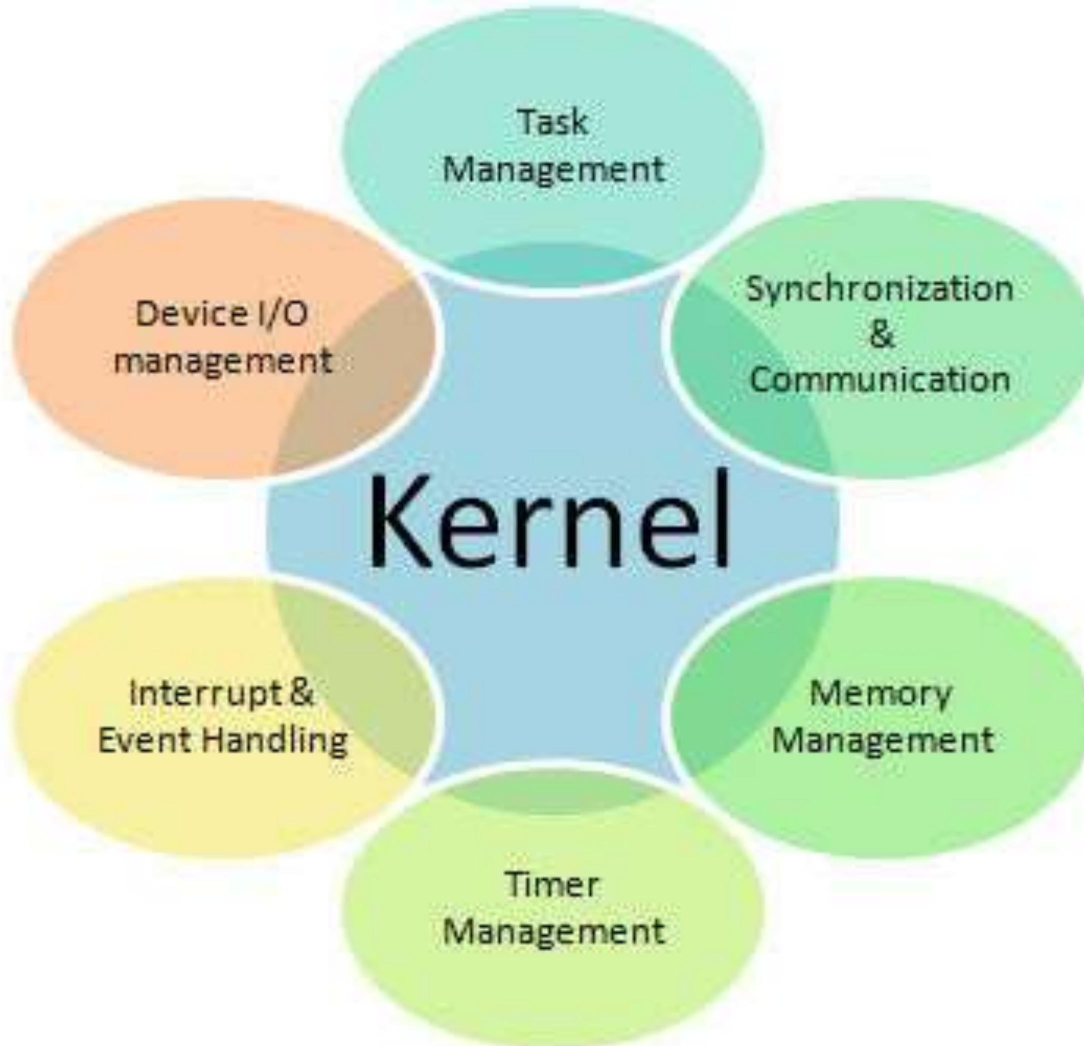
- **share resources** such as networking and servers, including file, compute, and print servers
- operating system is designed to facilitate **resource utilization**
- Operating systems are designed primarily to run without user intervention

- **System View**
  - From the computer's point of view, the operating system is the program most **intimately involved with the hardware**
  - we can view an operating system as a **resource allocator**
  - A computer system has **many resources** that may be required to solve a problem: **CPU time**, **memory space**, **file**-storage space, **I/O devices**, and
  - operating system **acts as the manager** of these resources
  - Facing numerous and possibly **conflicting requests** for resources, the operating system must decide **how to allocate them**
  - A slightly different view is operating system is a **control program**
  - control program **manages the execution** of **user programs** to **prevent errors** and **improper use** of the computer
  - especially concerned with the operation and control of I/O devices.

Kernel

- **Central component** of an operating system that manages operations of computer—usually called the **kernel**
- Along with the kernel, there are two other types of programs
  - **system programs- which are associated with the operating system but are not** necessarily part of the kernel
  - **application programs** which include all programs not associated with the operation of the system
- It is the **portion of the OS** code that is **always resident in memory** and **facilitates interactions** between **hardware and software components**
- It is the **core** of OS that provides basic services


Operating System
Kernel


Kernel
Task Management
Synchronization & Communication
Memory Management
Timer Management
Interrupt & Event Handling
Device I/O management

- During normal system startup, **boot-loader loads the kernel** from a storage device such as a hard drive into main memory
- Once the kernel is loaded into computer memory, the BIOS transfers control to the kernel

Middleware

- Mobile operating systems often include not only a core kernel but also **middleware—a set of software frameworks that provide additional** services to application developers
- **Middleware** is a type of computer software that provides services to software applications beyond those available from the operating system
- Apple's **iOS** and Google's **Android**—features a core kernel along with middleware that supports **database, multimedia, screen display, browsing and graphics**

Daemons

- Some services are provided **outside of the kernel**, by system programs that are loaded into memory at boot time to become **system processes, or system daemons** that run the entire time the kernel is running
- On UNIX, the first system process is **"init,"** and it starts many other daemons
- The daemon process names normally end with a ***d***.
- Some of the examples of daemon processes in Unix are:
  - **crond, syslogd, httpd, dhcpd**

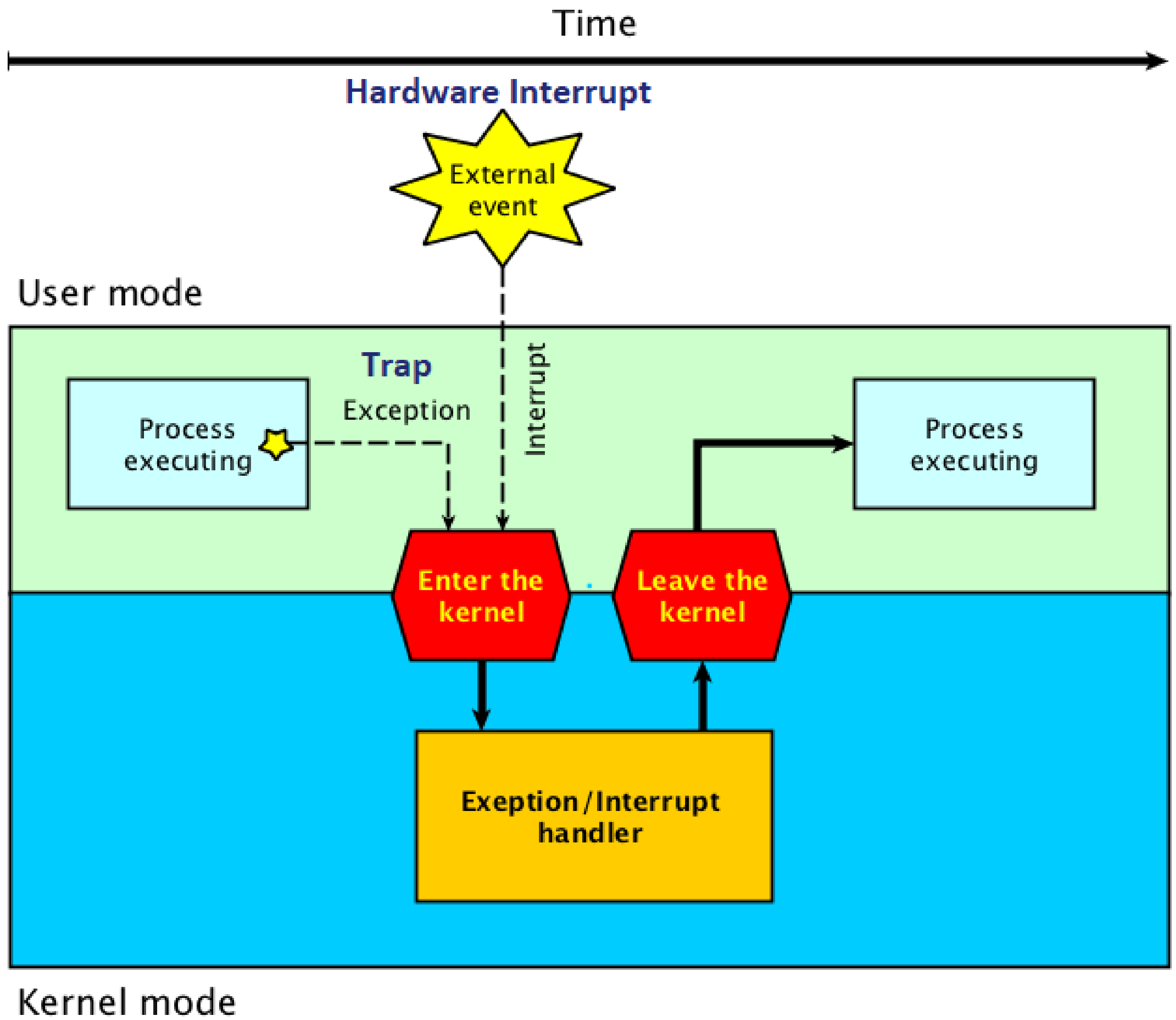
Zombie vs Orphan vs Daemon Processes

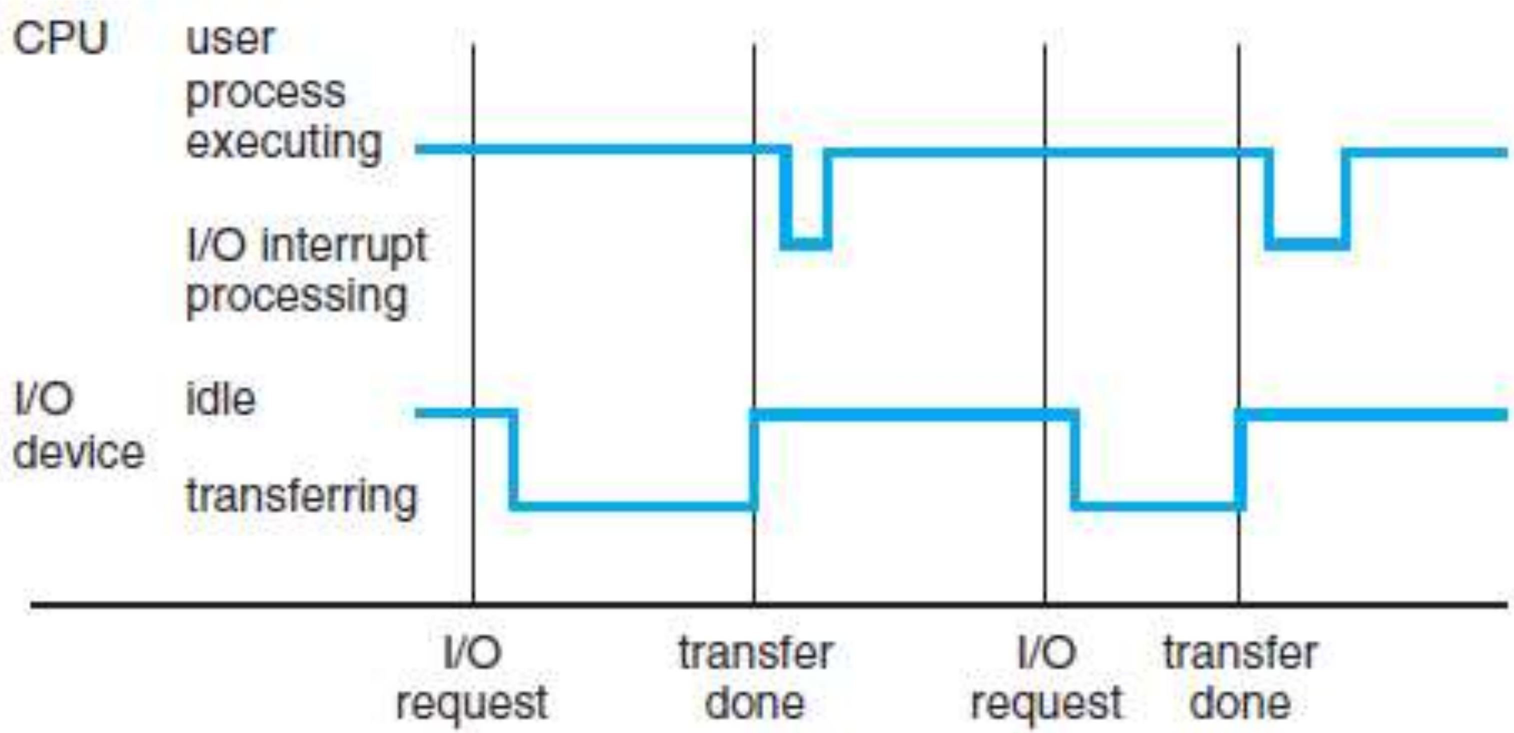
- A **zombie process** is a process whose execution is completed but it still has an entry in the process table.
- Zombie processes usually occur for child processes, as the parent process still needs to read its child's exit status
- **Orphan processes** are those processes that are still running even though their parent process has terminated or finished
- A **daemon process** is a background process that is not under the direct control of the user.
- This process is usually started when the system is bootstrapped and it terminated with the system shut down.

Interrupt

- **Occurrence of an event** is usually signaled by an **interrupt** from either the **hardware** or **software**
- Hardware may trigger an interrupt by **sending a signal** to the CPU, usually by way of the **system bus**
- Software may trigger an interrupt for executing a special operation called a **system call** or may indicate an **exception**
- When the CPU is interrupted, it stops what it is doing and immediately **transfers execution** to a **fixed location.**
- The fixed location usually contains the **starting address** where the **service routine (ISR)** for the interrupt is located
- The ISR executes; on completion, CPU resumes the interrupted process

- The interrupt must transfer control to **the appropriate interrupt service routine**
- The straightforward method for handling this transfer would be to invoke a **generic routine** to examine the interrupt information.
- The routine, in turn, would call the **interrupt-specific handler**
- A **table of pointers** to interrupt routines can be used to provide the necessary speed of handling
- Generally, the **table of pointers** is stored in low memory (the first hundred or so locations) called **interrupt vector**


Time
Hardware Interrupt
External event
User mode
Trap
Exception
Interrupt
Process executing
Process executing
Enter the kernel
Leave the kernel
Exeption/Interrupt handler
Kernel mode



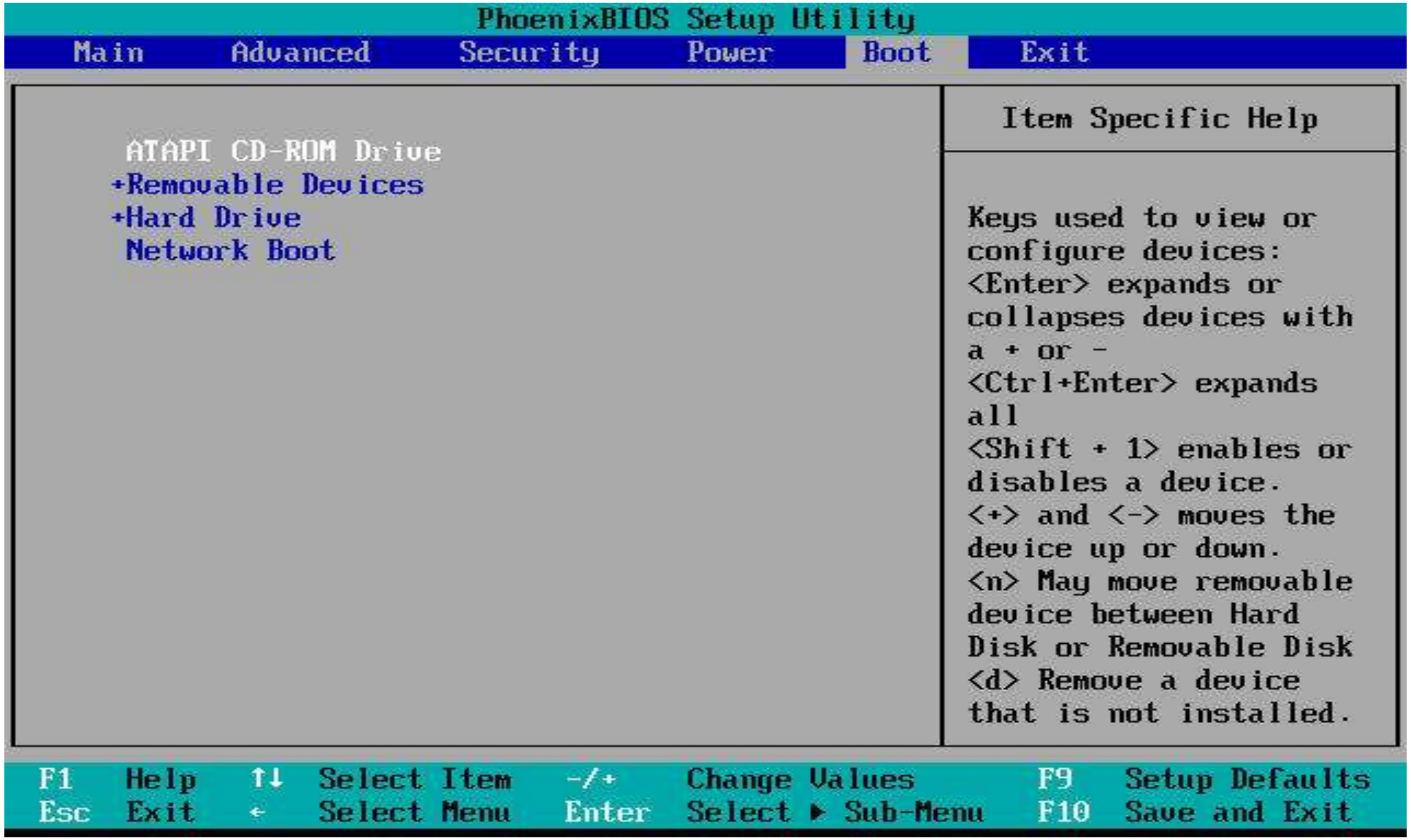
CPU
user process executing
I/O interrupt processing
I/O device
idle
transferring
I/O request
transfer done
I/O request
transfer done

Booting

- To load OS, a small program called **Bootstrap loader** that is stored in ROM is invoked.
- Since **ROM is non-volatile** it is preferred.
- The bootstrap loader **locates the OS kernel** at the **disk** and loads into **main memory.**
- In some systems, the bootstrap loader loads **another larger program** from disk that in turn loads the OS (**two stage booting)**

- Once the **kernel is loaded** and executing, it can **start providing services** to the system and its users
- Some services are provided outside of the kernel, by **system daemons** that run the entire time the kernel is running.
- On UNIX, the first system process is **“init,”** and it starts many other daemons.
- Once this phase is complete, the **system is fully booted**

BIOS


PhoenixBIOS Setup Utility
Main Advanced Security Power Boot Exit
ATAPI CD-ROM Drive
+Removable Devices
+Hard Drive
Network Boot
Item Specific Help
Keys used to view or configure devices:
<Enter> expands or collapses devices with a + or -
<Ctrl+Enter> expands all
<Shift + 1> enables or disables a device.
<+> and <-> moves the device up or down.
<n> May move removable device between Hard Disk or Removable Disk
<d> Remove a device that is not installed.
F1 Help ↑↓ Select Item -/+ Change Values F9 Setup Defaults
Esc Exit ← Select Menu Enter Select ▶ Sub-Menu F10 Save and Exit

Operating-System Structure

- **Multiprogramming** needed for efficiency
  - Operating system keeps **several jobs in memory** simultaneously
  - Since, in general, main memory is too small to accommodate all jobs, the jobs are **kept initially on the disk**
  - The **operating system picks** and **begins to execute** one of the jobs in memory
  - Eventually, the job may have to **wait for some task**, such as **an I/O operation**, to complete
  - In a **non-multiprogrammed system**, the CPU would **sit idle**
  - In a **multiprogrammed system**, the operating system simply **switches to**, and executes, **another job**
  - When that job needs to wait, the CPU switches to another job, and so on.

- **Time sharing**
  - logical extension of multiprogramming
  - In time-sharing systems, the **CPU executes multiple jobs** by **switching** among them, but the **switches occur so frequently** that the **users can interact** with each program while it is running
  - Time sharing requires an **interactive computer system**, which provides direct communication between the user and the system
  - User gives instructions to the operating system or to a program directly, using a **input device such as a keyboard, mouse**, or **touch screen**, and waits for immediate results on an **output device**
  - **response time should** be short—typically less than one second

- A time-shared operating system **allows many users to share** the computer simultaneously.
- As the system **switches rapidly from one user to the next**, each user is given the impression that the entire computer system is **dedicated to his use**, even though it is being shared among many users
- Time sharing and multiprogramming require that **several jobs be kept simultaneously in memory**
- If several jobs are ready to be brought into memory, and if there is not **enough room for all of them**, then the system must choose among them.
- Making this decision involves **job scheduling**
- Having several programs in memory at the same time requires some form of memory management

- if several jobs are ready to run at the same time, the system must choose which job will run first. Making this decision is **CPU scheduling**
- In a time-sharing system, the operating system must ensure **reasonable response time**
- This goal is sometimes accomplished through **swapping,** whereby processes are swapped in and out of main memory to the disk
- Amore common method for ensuring reasonable response time is **virtual memory, a** technique that allows the execution of a process that is not completely in main memory
- main advantage of the virtual-memory scheme is that it enables users to run programs that are larger than actual **physical memory.**
- Further, it abstracts main memory into a large, uniform array of storage, separating **logical memory as viewed by the user from physical memory**

Operating-System Operations

- Modern operating systems are **Interrupt driven** by hardware
- OS sits quietly, waiting for something to happen
- Events are almost always signaled by the occurrence of an **interrupt or a trap.**
- Software error or request creates **exception** or **trap**
  - Division by zero, request for operating system service
- For each type of interrupt, separate segments of code in the **interrupt service routine** determine what action to be taken.
- Since the **processes share the hardware and software** resources, we need to make sure that an **error in a process** could affect only that process.

Need of protection

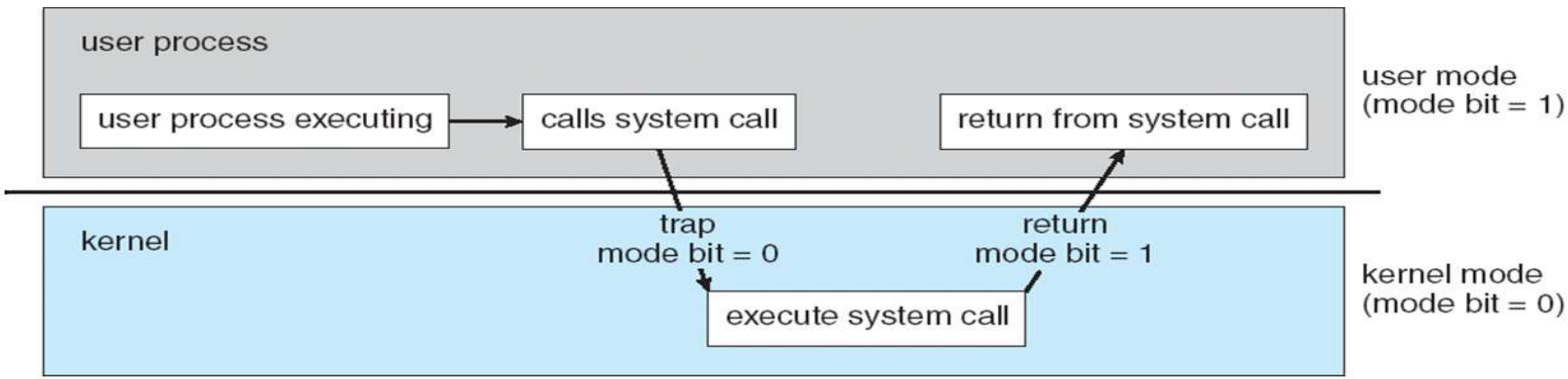
- With sharing, **many processes** could be adversely affected by a **bug in one program**
- In a multiprogramming system, one erroneous program might modify another program, the data of another program, or even the operating system itself
- Without **protection** against these sorts of errors, either the computer must **only one process at a time**
- A **properly designed operating system** must ensure that an incorrect (or malicious) program cannot cause other programs to execute incorrectly

Dual-Mode and Multimode Operation

- But with sharing, **many processes could be adversely affected** by a bug in one program
- **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

Transition from User to Kernel Mode

- When the computer system is executing on behalf of a **user application**, the system is in **user mode**
- However, when a user application requests a service from the operating system (**via a system call**), the system must transition from **user to kernel mode** to fulfill the request



- At **system boot time**, the system starts in **kernel mode**.
- Operating system is then loaded and starts user applications in user mode
- Whenever a **trap or interrupt** occurs, the hardware switches from user mode to kernel mode (that is, changes the state of the mode bit to 0)
- Thus, whenever the **operating system gains control** of the CPU, it is in **kernel mode**
- The **dual mode** of operation provides us with the means for protecting the **operating system from errant users—and** errant **users from one another**

- **Privileged instructions**
  - Some of the machine instructions that **may cause harm** are designated as **privileged instructions**
  - The hardware allows privileged instructions to be executed **only in kernel mode**
  - If an **attempt** is made to **execute a privileged instruction in user mode**, the hardware does not execute the instruction but rather treats it as **illegal and traps** it to the operating system
  - instruction to switch to kernel mode is an example of a privileged instruction
  - E.g I/O control, timer management, and interrupt 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 (PC)** 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 / 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**
- Providing mechanisms for **process scheduling on CPU**
- **Suspending and resuming** processes
- Providing mechanisms for **process synchronization**
- Providing mechanisms for **process communication(IPC)**
- Providing mechanisms for **deadlock handling**

Virtualization

- OS virtualization is the use of software to allow a piece of **hardware to run multiple operating system** images at the same time – host OS and guest OS.
- CPUs that support **virtualization** frequently have a separate mode to indicate when the **virtual machine manager** (VMM)—and the **virtualization management software**—is in control of the system
- In this mode, the **VMM has more privileges** than user processes but fewer than the kernel
- It needs that **level of privilege** so it can create and manage **virtual machines**, changing the CPU state to do so.

Lack of dual mode

- Lack of a **hardware-supported dual mode** can cause serious shortcomings in an operating system
- For instance, **MS-DOS** was written for the Intel 8088 architecture, which has no mode bit and therefore no dual mode
- A **user program** running awry **can wipe out the operating system** by writing over it with data; and **multiple programs are able to write to a device at the same time**, with potentially disastrous results
- Modern versions of the Intel CPU do provide dual-mode operation

Timer

- We must ensure that the operating system maintains control over the CPU
- We cannot allow a user program to get stuck in an **infinite loop** or to fail to call system services and **never return control** to the operating system
- To accomplish this goal, we can use a timer
- A timer can be **set to interrupt** the computer after a specified period.
- The period may be **fixed** (for example, 1/60 second) or **variable** (for example, from 1 millisecond to 1 second).

- The operating system **sets the counter**
- Every time the **clock ticks**, the counter is **decremented**
- When the counter reaches **0**, an **interrupt occur**
- Before turning over control to the user, the operating system ensures that the **timer is set** to interrupt
- If the timer interrupts, control transfers **automatically to the operating system**
- We can use the timer to **prevent a user program from running too long**
- A simple technique is to initialize a counter with the amount of time that a program is allowed to run – **round robin**