1151CS108 – OPERATING SYSTEMS

Faculty Name : Mr.A.Arul Prasath

Employee Id : TTS3118

Slot Number : S7 & S8

CO1 Explain the operating system program, structures and operations with system calls.

- ✓ **Operating system** (**OS**), **program** that manages a computer's resources, especially the allocation of those resources among other **programs**.
- ✓ Typical resources include the central processing unit (CPU), computer memory, file storage, input/output (I/O) devices, and network connections.
- ✓ The structure of the OS depends mainly on how the various common components of the operating system are interconnected .
- ✓ System call provides the services of the operating system to the user programs via Application Program Interface (API).

CO2 Apply the process management concept for real time problems.

- ✓ A **process** is basically a program in execution.
- ✓ **Process Management** refers to aligning of **processes** which includes:

1. Scheduling processes and threads on the CPUs.
2. Creating and deleting both user and system processes.
3. Suspending and resuming processes.
4. Providing mechanisms for process synchronization.
5. Providing mechanisms for process communication.

CO3 Illustrate CPU scheduling algorithms and to handle the deadlock for the given situation.

- ✓ **CPU scheduling** is a process which allows one process to use the **CPU** while the execution of another process is on hold(in waiting state) due to unavailability of any resource like I/O etc.

1. First-Come, First-Served (FCFS) Scheduling.
2. Shortest-Job-Next (SJN) Scheduling.
3. Priority Scheduling. Shortest Remaining Time.
4. Round Robin(RR) Scheduling and
5. Multiple-Level Queues Scheduling.

CO4 Explain the concepts of various memory management techniques.

✓ Memory management is the functionality of an operating system which handles or manages primary memory and moves processes back and forth between main memory and disk during execution.

1. Segmentation.
2. Paging.
3. Swapping.

CO5 Summarize the storage concepts of disk and file.

- ✓ Meta data of a file which stores the directory structure and information about the file.
- ✓ File content which contains the actual file content which is part of the data.
- ✓ File **storage** leads to File Systems, which will have directories, files, regular files and etc file related meta data inside them.

COURSE PRE-REQUISITES:

1150CS102 Data Structures

CO – PO Matrix: NBA

CO Nos.	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO 1	PSO 2	PSO 3
CO1	1	2	2											1	1
CO2	1	3	2	2	2		1	2	1		1	2	2	3	3
CO3	1	3	2	2	1				1				2	3	3
CO4	1	2	2	2	1			2				1	2	2	2
CO5	1	2	2	1		1						1		2	2

CO – SO Matrix: EAC

CO Nos.	SO1	SO2	SO3	SO4	SO5	SO6	SO7
CO1	1						1
CO2	3			2		3	1
CO3	3	2		2		2	1
CO4	2	2		2		2	1
CO5		1				1	

CO – SO Matrix: CAC

CO Nos.	SO1	SO2	SO3	SO4	SO5	SO6
CO1	1					1
CO2	3			2		3
CO3	3	2		2		3
CO4	2	3		2		2
CO5		1				2

COURSE CONTENT

UNIT I : OPERATING SYSTEMS OVERVIEW

UNIT II : PROCESS MANAGEMENT

UNIT III : SCHEDULING AND DEADLOCK MANAGEMENT

UNIT IV : STORAGE MANAGEMENT

UNIT V : STORAGE STRUCTURE

COURSE CONTENT

UNIT I OPERATING SYSTEMS OVERVIEW 9

Operating system overview: Objectives – Functions - Computer System Organization - Operating System Structure - Operating System Operations - System Calls - System Programs.st

COURSE CONTENT

UNIT II PROCESS MANAGEMENT **9**

Processes: Process Concept - Process Scheduling - Operations on Processes - Inter process Communication - Process Synchronization: The Critical-Section Problem - Semaphores - Classic Problems of Synchronization - Monitors. Case Study: Windows 10 operating systems.

COURSE CONTENT

UNIT III SCHEDULING AND DEADLOCK MANAGEMENT 9

CPU Scheduling: Scheduling Criteria - Scheduling Algorithms - Deadlocks: Deadlock Characterization - Methods for Handling Deadlocks - Deadlock Prevention - Deadlock Avoidance - Deadlock Detection - Recovery from Deadlock. Case Study: MAC operating system

COURSE CONTENT

UNIT IV STORAGE MANAGEMENT 9

Main Memory: Swapping - Contiguous Memory Allocation – Segmentation – Paging - Virtual Memory: Demand Paging - Page Replacement - Allocation of Frames - Thrashing. Case Study: Android operating system.

COURSE CONTENT

UNIT V STORAGE STRUCTURE 9

Mass Storage Structure: Disk Structure - Disk Scheduling - Disk Management File - System Interface: File Concepts - Directory Structure - File Sharing – Protection - File System. Case Study: Linux operating system

TOTAL : 45 Periods

LEARNING RESOURCES

Text Books

1. Abraham Silberschatz, Peter Baer Galvin and Greg Gagne, “Operating System Concepts”, 9th Edition, John Wiley and Sons Inc., 2012.
2.Richard Petersen, “Linux: The Complete Reference”, 6th Edition, Tata McGraw-Hill, 2008.

Reference Books

1. Andrew S. Tanenbaum, “Modern Operating Systems”, 4th Edition, Prentice Hall, Wesley, 2014.
2.William Stallings, “Operating Systems – Internals and Design Principles”, 7th Edition, Prentice Hall, 2011.
3. Harvey M. Deitel, “Operating Systems”, 7th Edition, Prentice Hall, 2003.
4. D M Dhamdhere, “Operating Systems: A Concept-Based Approach”, 2nd Edition, Tata McGraw-Hill Education, 2007.
5. Charles Crowley, “Operating Systems: A Design-Oriented Approach”, Tata McGraw Hill Education”, 1996.

Digital Resources

1.http://www.tutorialspoint.com/operating_system/
2.http://www.mu.ac.in/myweb_test/MCA%20study%20material/OS%20-%20PDF.pdf
3.http://codex.cs.yale.edu/avi/os-book/OS8/os8c/slide-dir/PDF-dir/ch2.pdf
4.http://www.freebookcentre.net/CompuScience/Free-Operating-Systems-Books-download.html

Why we need to learn OS?

- ✓ OS is a key part of a computer system
- ✓ It is the most important software that runs on a computer. It manages the computer's
    1. Memory
    2. Processes
    3. Software
    4. Hardware.
- ✓ It also allows you to communicate with the computer
- ✓ Understand how computers work under the hood

UNIT I OPERATING SYSTEMS OVERVIEW

- ✓ **Operating system overview: Objectives**
- ✓ **Functions**
- ✓ **Computer System Organization**
- ✓ **Operating System Structure**
- ✓ **Operating System Operations**
- ✓ **System Calls**
- ✓ **System Programs.**

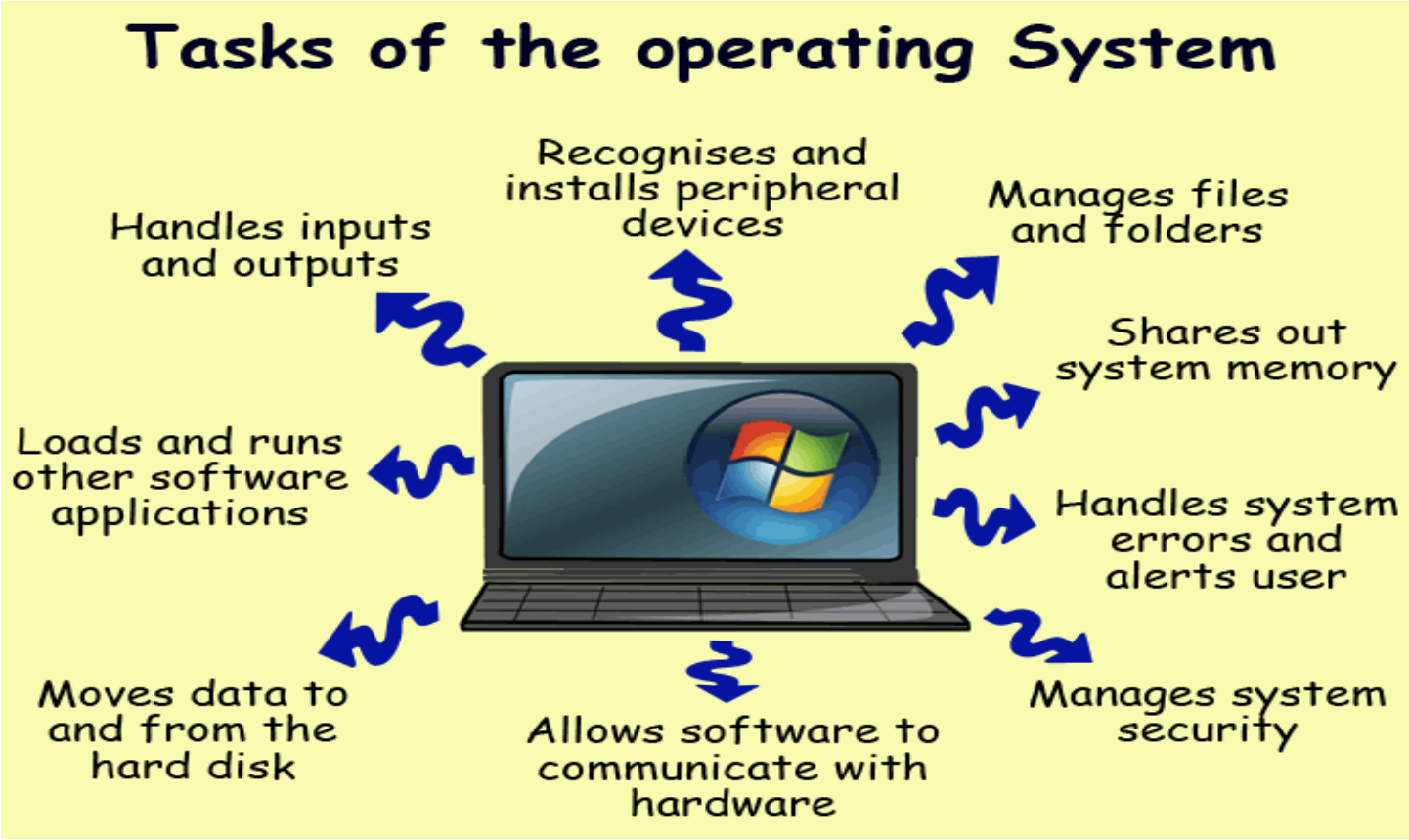
OPERATING SYSTEMS - Definition

- ✓ An **Operating System** (OS) is an interface between a computer user and computer hardware.
- ✓ Acts as an intermediary between user of computer & computer hardware.

Goals of OS:

1. Execute user programs and make solving user problems easier.
2. Make the computer system convenient to use.
3. Use the computer hardware in an efficient manner.

GOALS OF AN OPERATING SYSTEMS

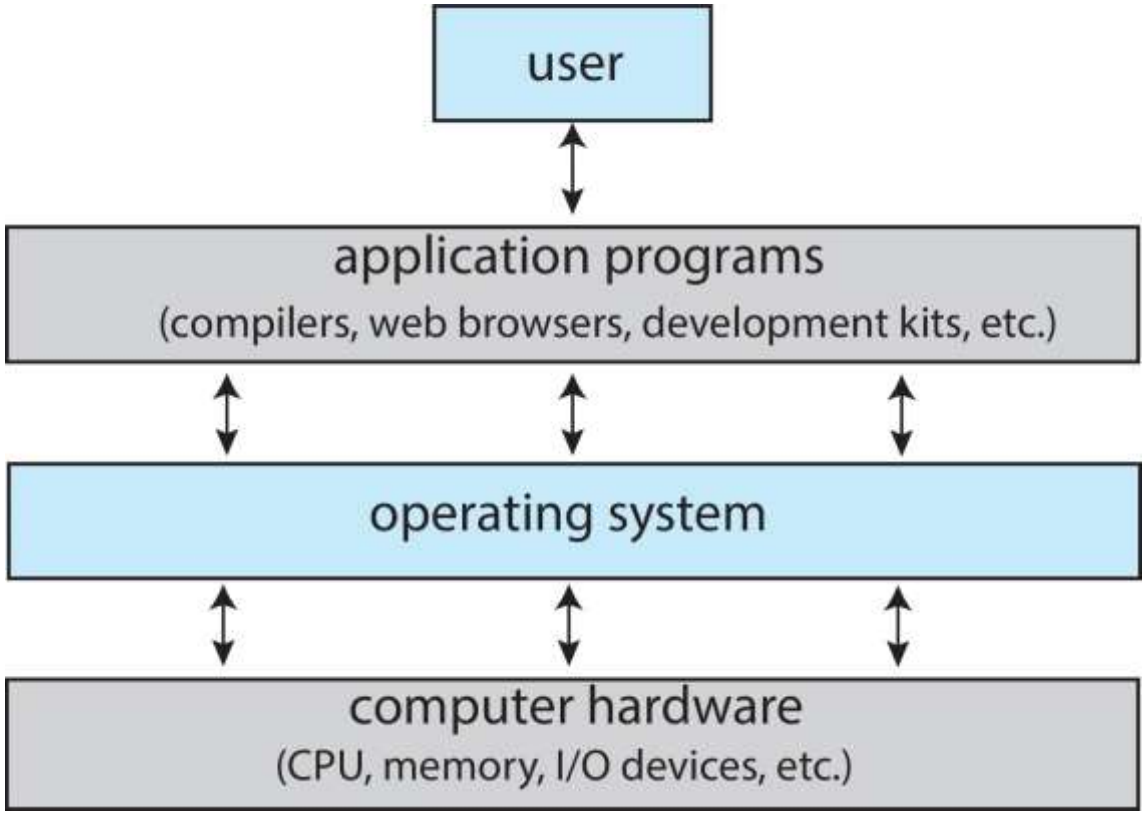


COMPUTER SYSTEM STRUCTURE

✓ Computer system can be divided into **four** components:

1) Hardware – provides basic computing resources
   Eg. CPU, memory, I/O devices
2) Operating system
   Controls and coordinates use of hardware among various applications and users
3) 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
4) Users
   People, machines, other computers

ABSTRACT VIEW OF COMPONENTS OF COMPUTER



WHAT OPERATING SYSTEMS DO???

Depends on the point of view

- ✓ Users want convenience, **ease of use** and **good performance**
  - Don't care about **resource utilization**
- ✓ But shared computer such as **mainframe** or **minicomputer** must keep all users happy
  - Operating system is a **resource allocator** and **control program** making efficient use of HW and managing execution of user programs
- ✓ Users of dedicate systems such as **workstations** have dedicated resources but frequently use shared resources from **servers**
- ✓ Mobile devices like smartphones and tables are resource poor, optimized for usability and battery life
  - Mobile user interfaces such as touch screens, voice recognition
- ✓ Some computers have little or no user interface, such as embedded computers in devices and automobiles
  - Run primarily without user intervention

EXAMPLES OF OPERATING SYSTEMS

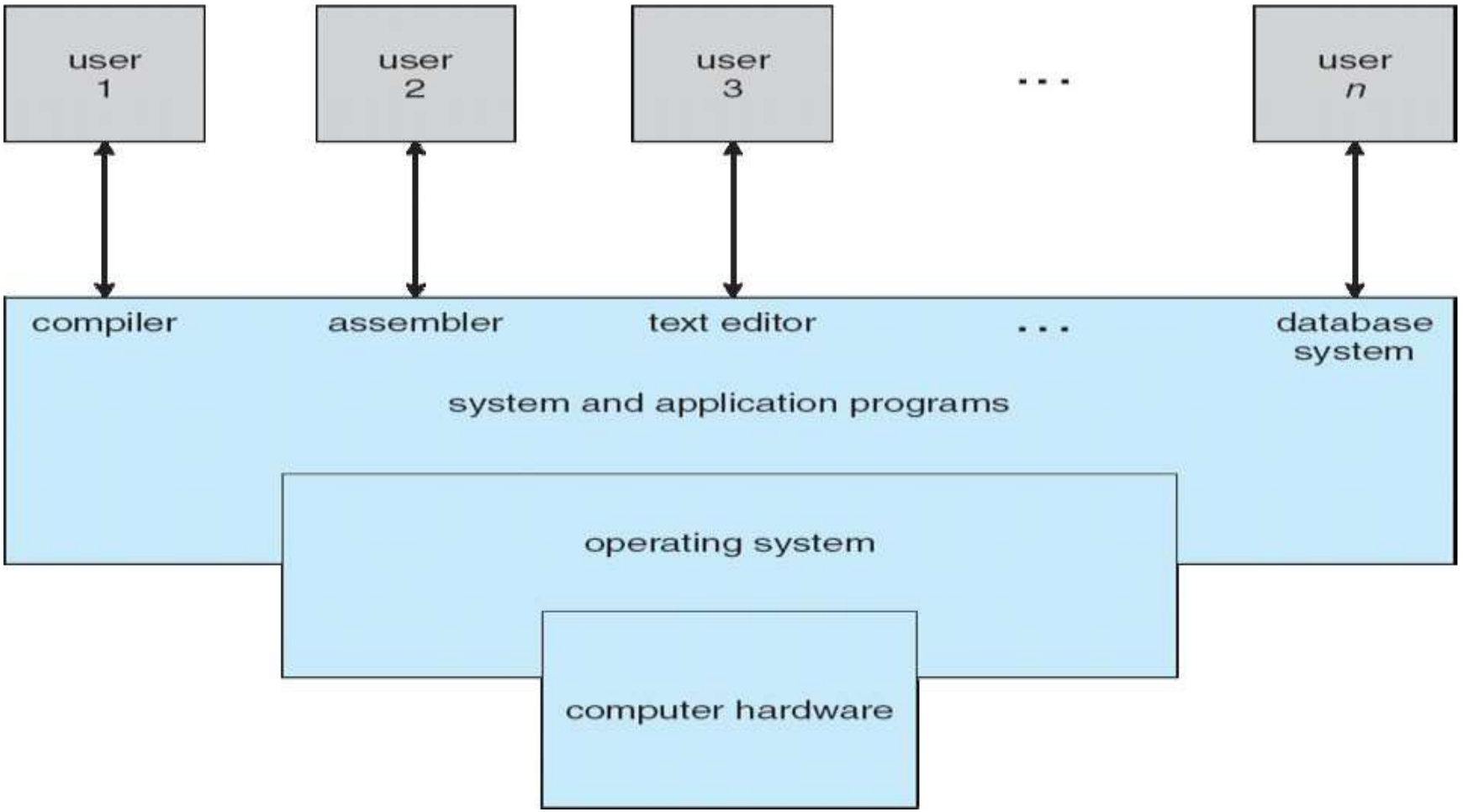
IMPORTANT FUNCTIONS OF AN OPERATING SYSTEMS

1. Memory Management
2. Processor Management
3. Device Management
4. File Management
5. Security
6. Control over system performance
7. Job accounting
8. Error detecting aids
9. Coordination between other software and users

Four Main Components of Computer

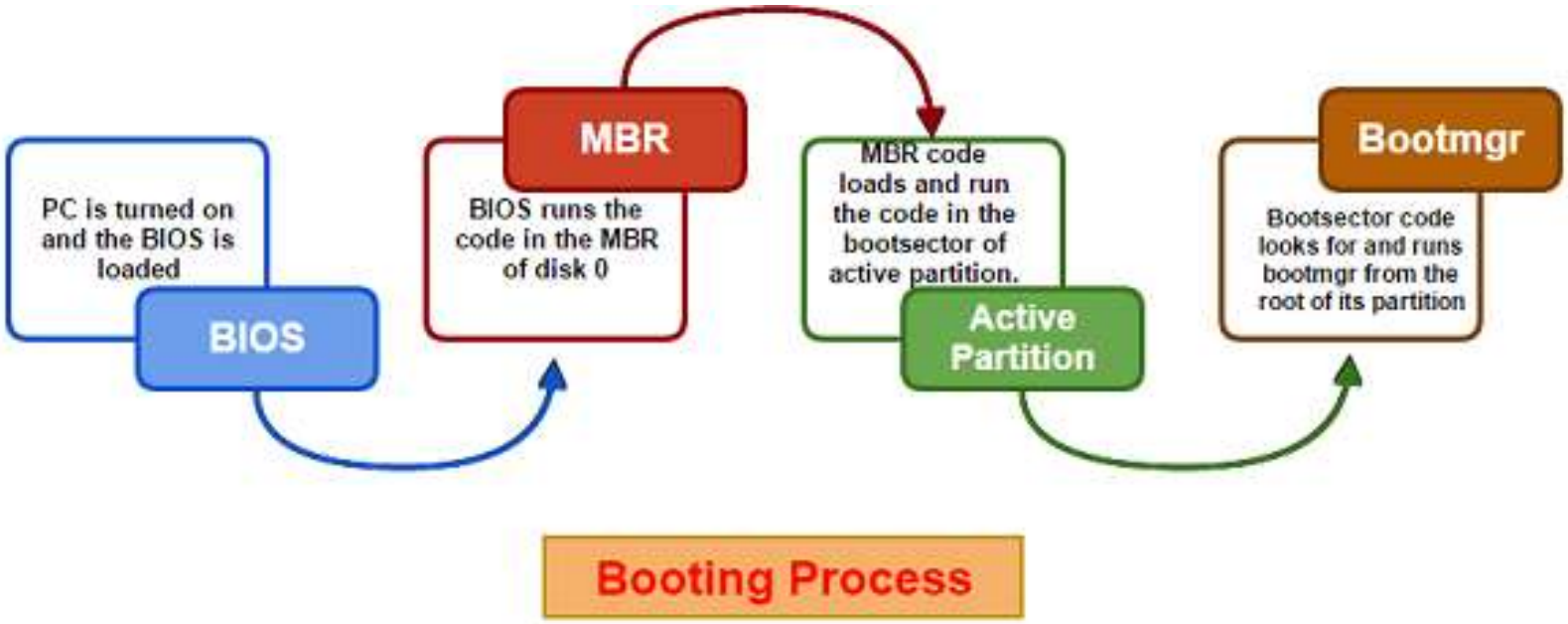
1. Hardware
2. Operating System
3. Application Programs
4. Users

VIEWS OF OPERATING SYSTEMS



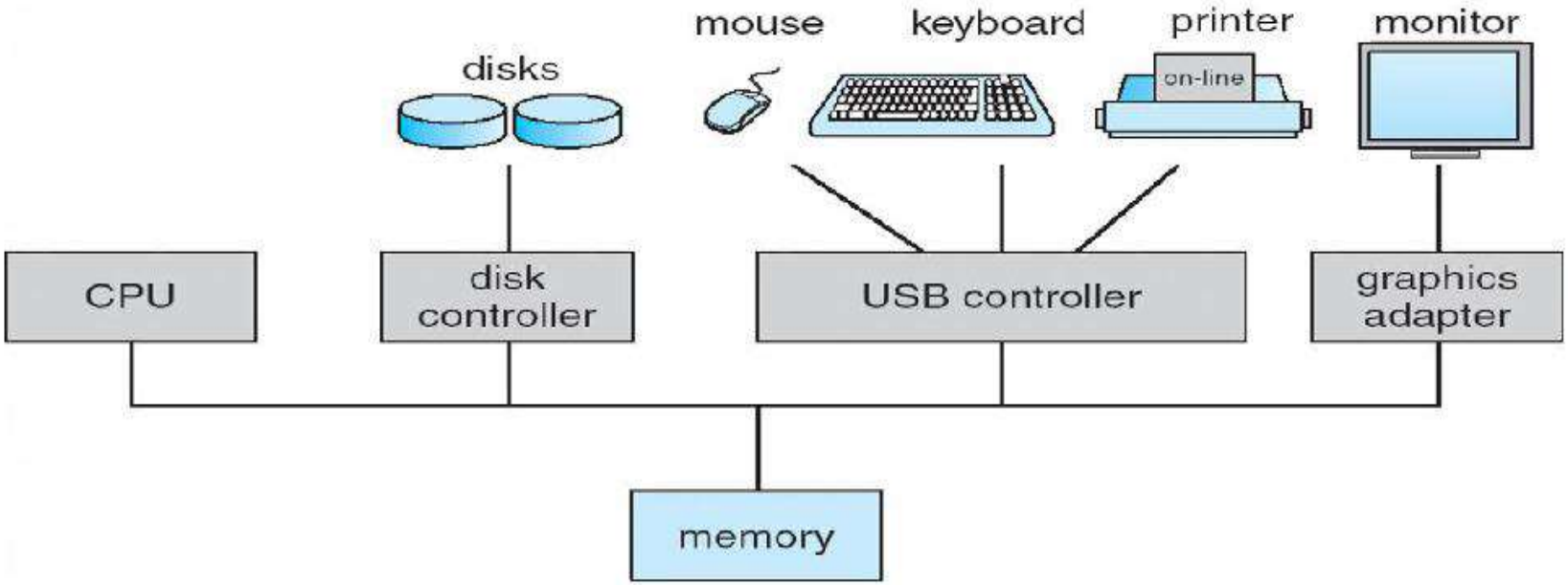
COMPUTER START UP

- ✓ **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.



Booting Process

COMPUTER SYSTEM ORGANIZATION



COMPUTER SYSTEM ORGANIZATION

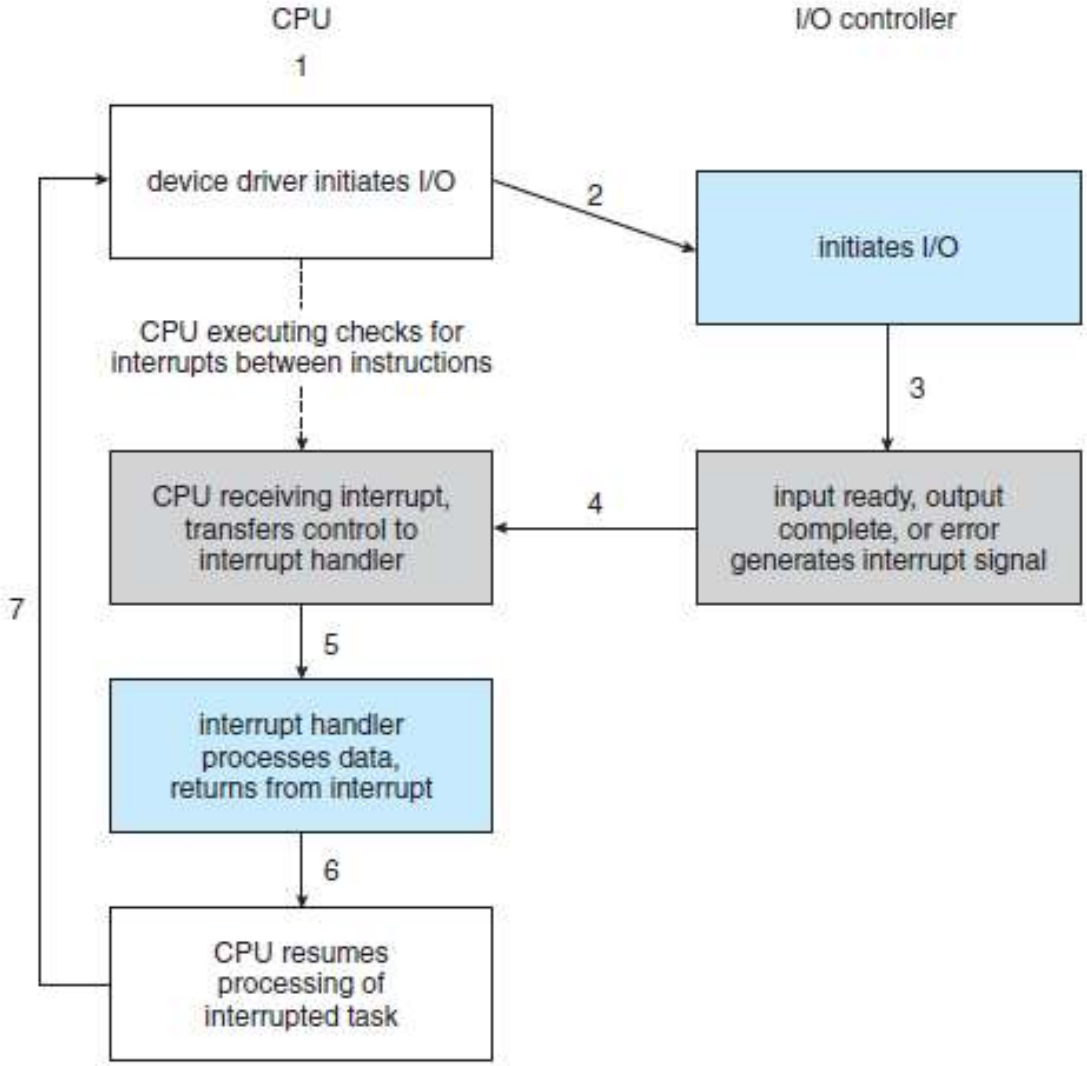
Computer-System Operation

- ✓ One or more CPUs, device controllers connect through common bus providing access to shared memory.
- ✓ 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.
- ✓ Each device controller type has an operating system device driver to manage it.
- ✓ CPU moves data from/to main memory to/from local buffers.
- ✓ I/O is from the device to local buffer of controller.
- ✓ Device controller informs CPU that it has finished its operation by causing an interrupt.

INTERRUPTS

- ✓ Device controller informs CPU that it has finished its operation by causing an interrupt.
- ✓ 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 DRIVEN IO CYCLE

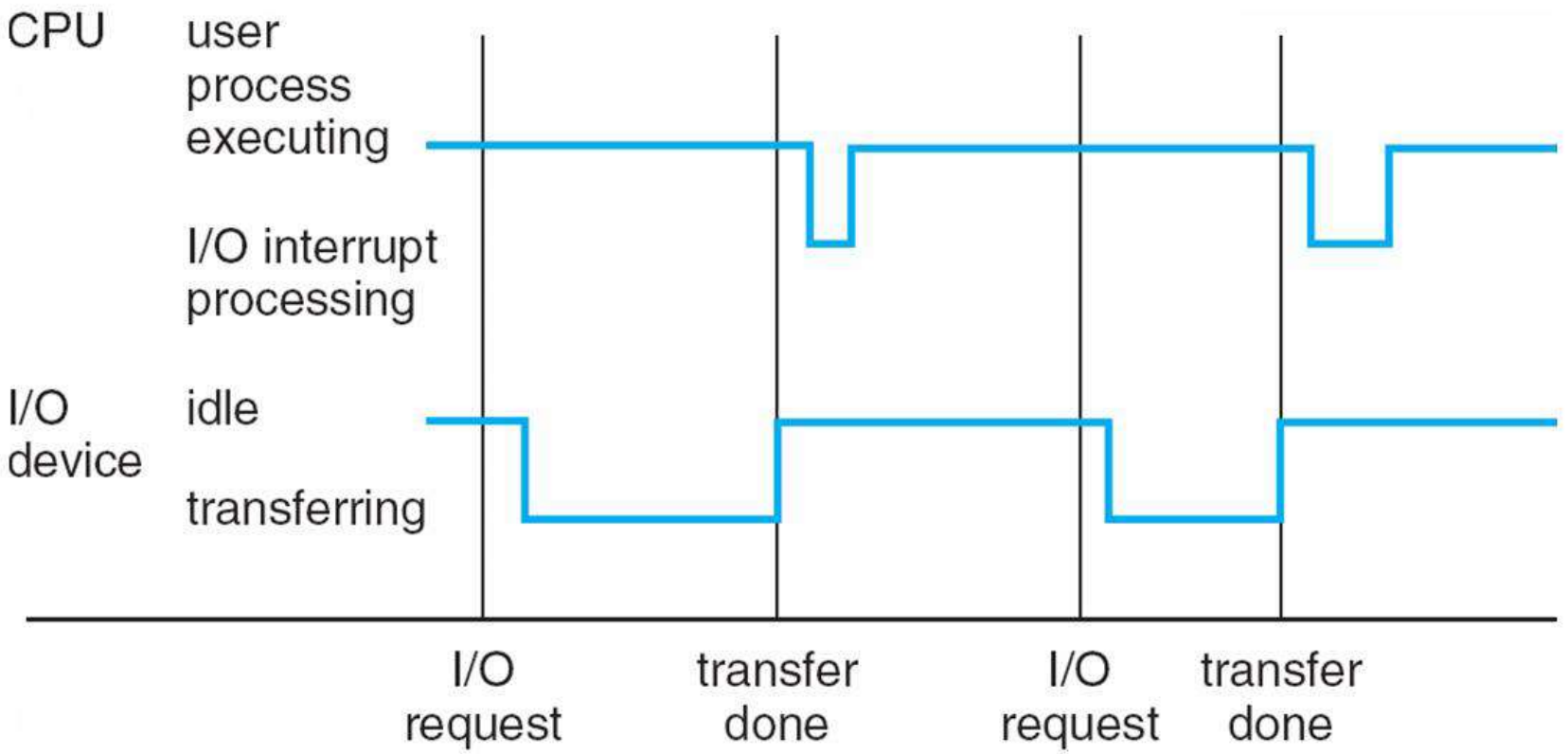


INTERRUPT HANDLING

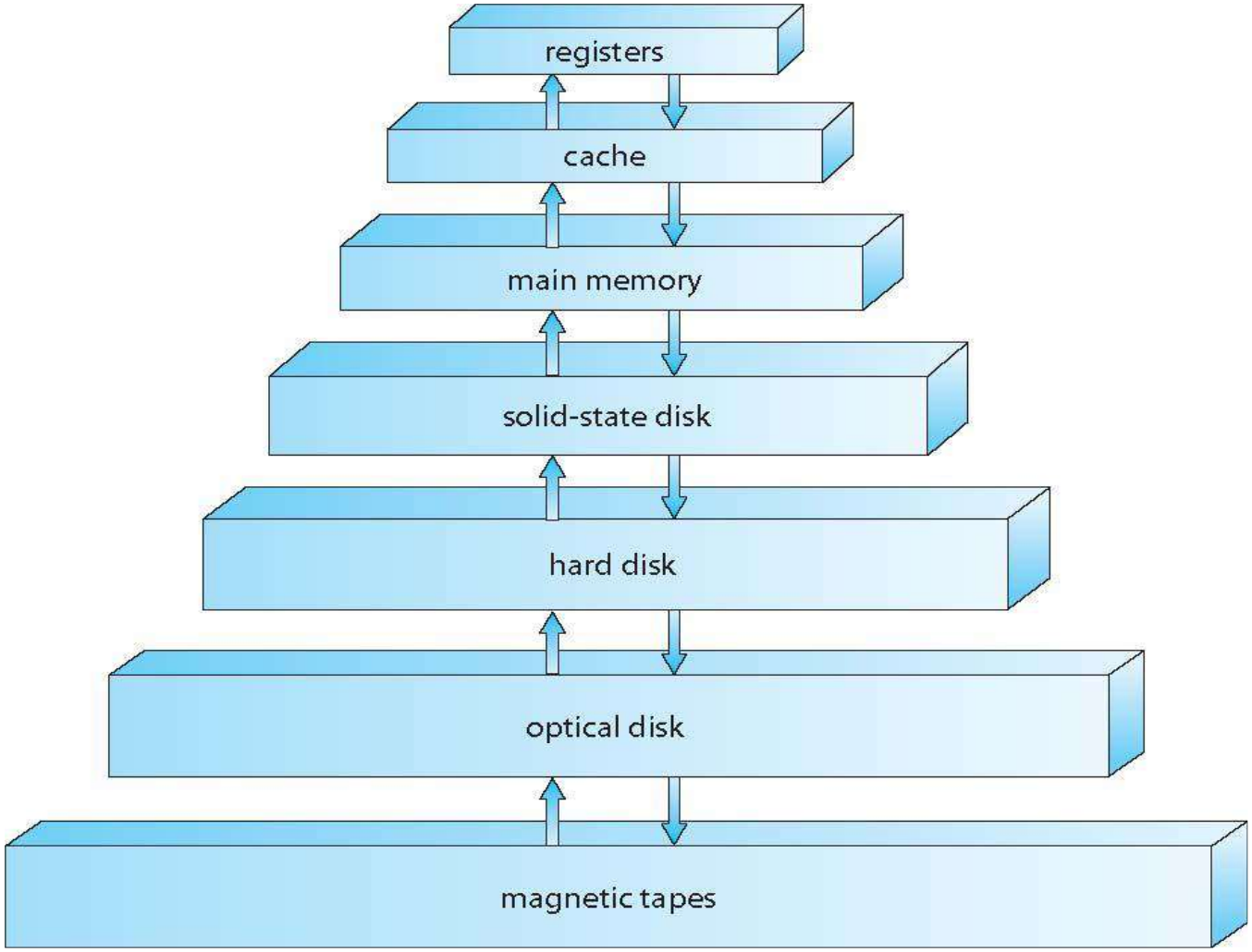
- ✓The OS preserves the state of the CPU by storing registers and the program counter.
- ✓Also determines which type of interrupt has occurred:

1. **Polling -** The interrupt controller polls (send a signal out to) each device to determine which one made the request.
2. **Vectored** interrupt system - Separate segments of code determine what action should be taken for each type of interrupt.

INTERRUPT TIMELINE



STORAGE HIERARCHY



I/O STRUCTURE

- ✓ Storage is the only one of many types of I/O devices within a computer.
- ✓ A large portion of OS code is dedicated to managing I/O.
- ✓ Because of its importance of reliability & performance.
- ✓ Computer system consists of CPU & multiple devices controllers through common bus.
- ✓ Each device controller is in charge of a specific types of devices.

I/O STRUCTURE

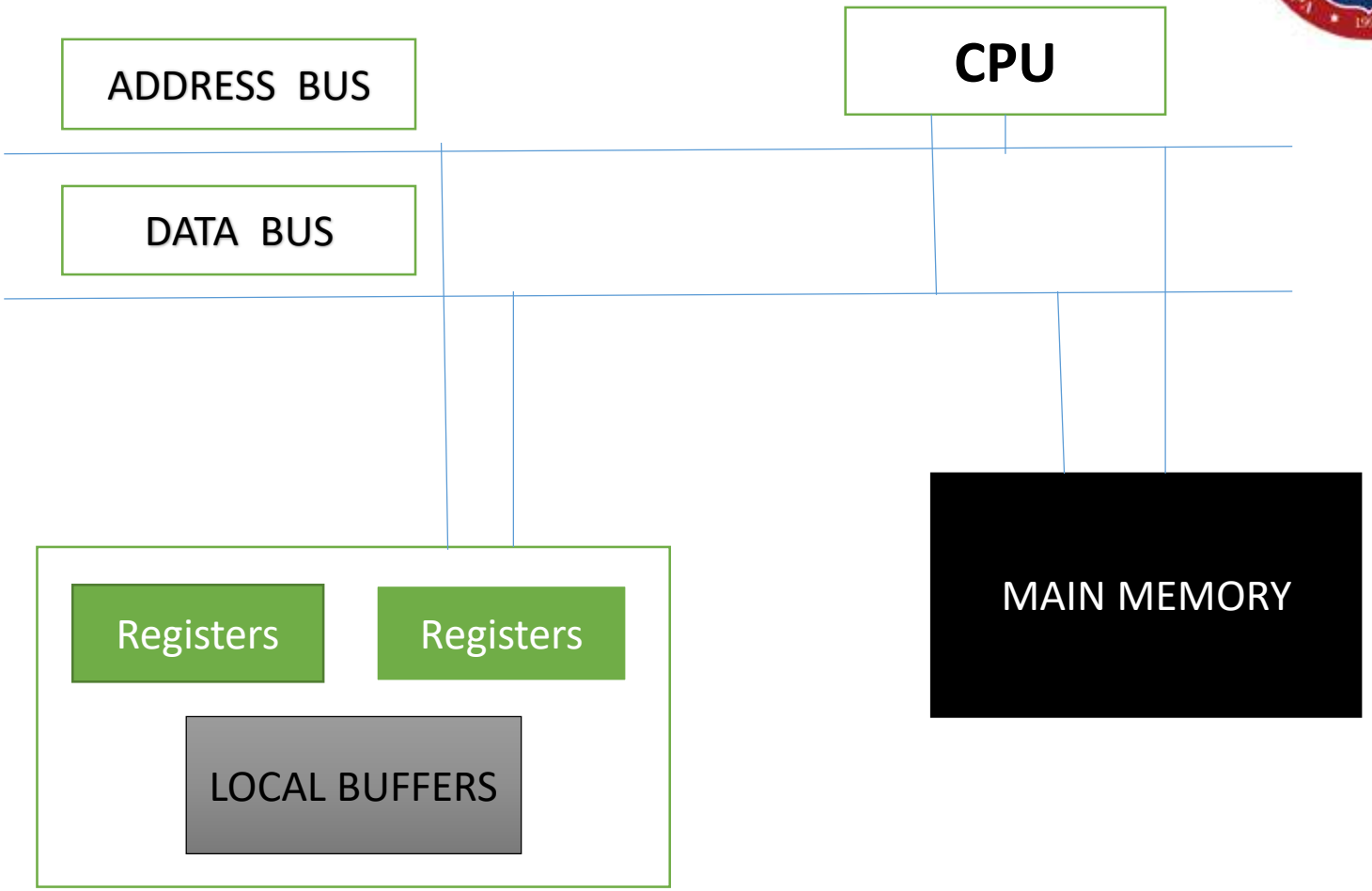
DEVICE CONTROLLER

- ✓ Maintains local **buffer storage** & set of **Special purpose registers.**

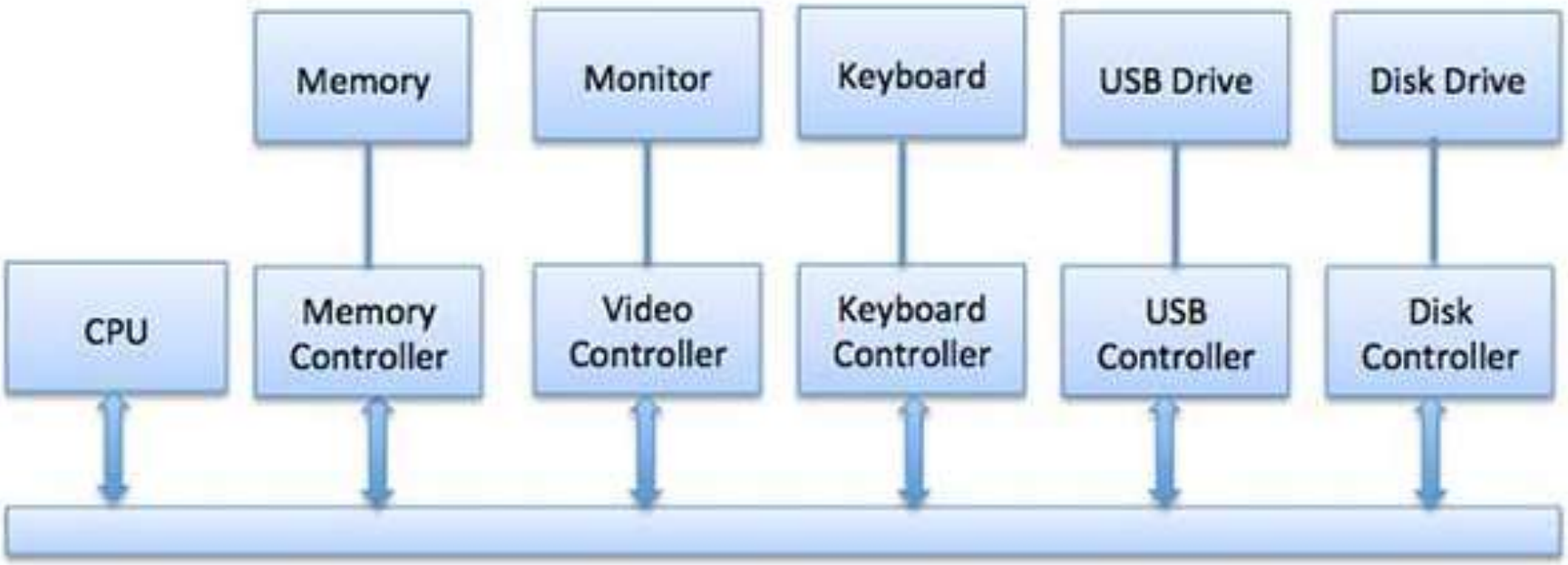
DEVICE DRIVER

- ✓ OS have a device driver for each device controller.
- ✓ Device driver understands the device controller & gives uniform interface to the rest of the OS.
- ✓ Device controller transfers blocks of data from buffer storage directly to main memory without CPU intervention.
- ✓ Only one interrupt is generated per block, rather than the one interrupt per byte.

DEVICE CONTROLLER



DEVICE CONTROLLER



WORKING OF I/O OPERATIONS

- ✓ To start an IO operations ,device driver loads the appropriate registers within the device controller.
- ✓ Device controller in turns examines the content of these registers to determine what action to take.
- ✓ The controller starts the transfer of data from the device to its local buffer.
- ✓ Once the transfer of data is complete ,the device controller informs the device driver via an interrupt that it has finished its operations.
- ✓ Device driver then return the control to OS.

Issues:

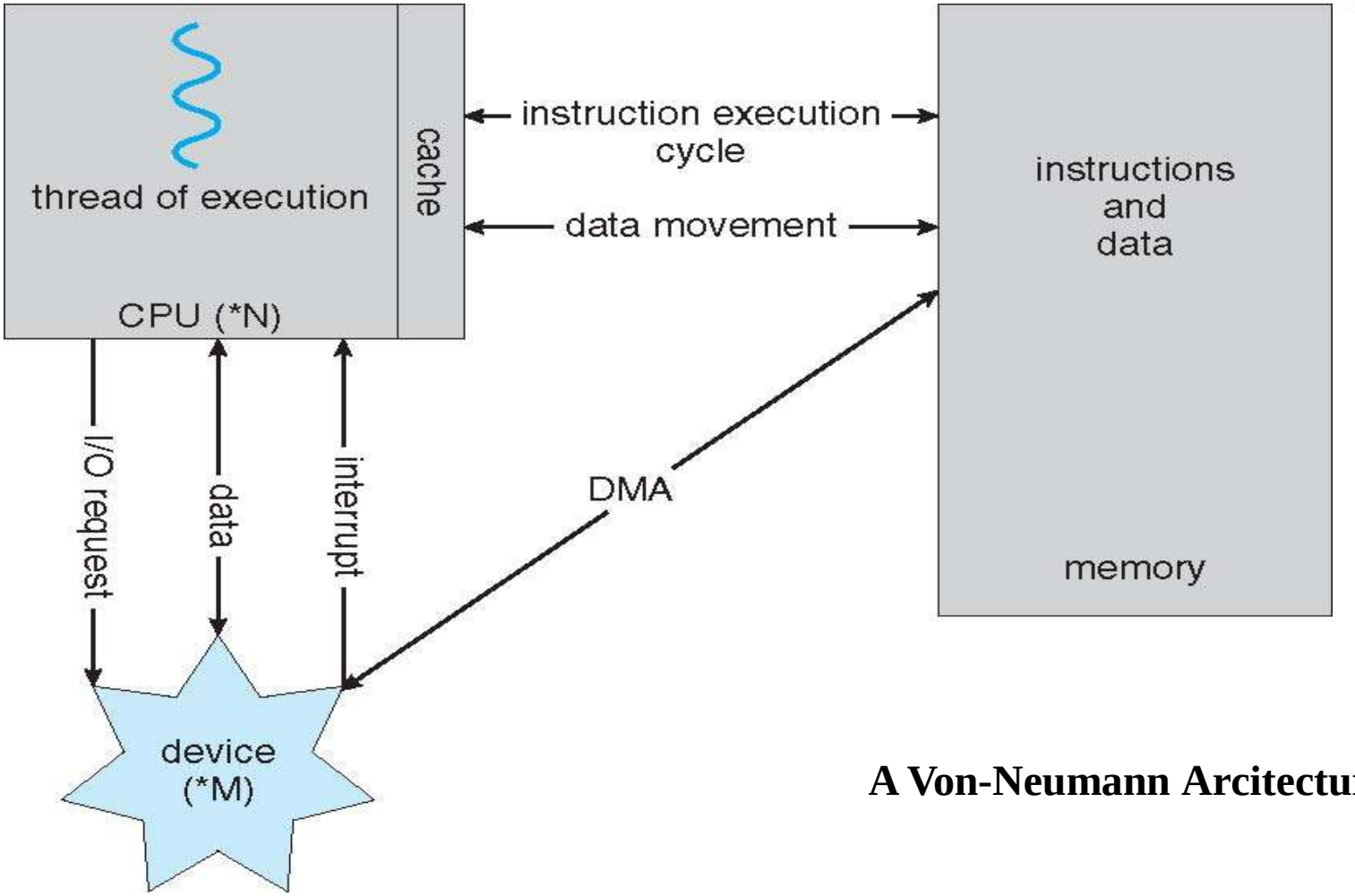
- ✓ **This form of interrupt driven IO is fine for moving small amounts of data.**
- ✓ **But can produce high overhead when used for bulk data movement.**

WORKING OF I/O OPERATIONS

DMA (DIRECT MEMORY ACCESS)

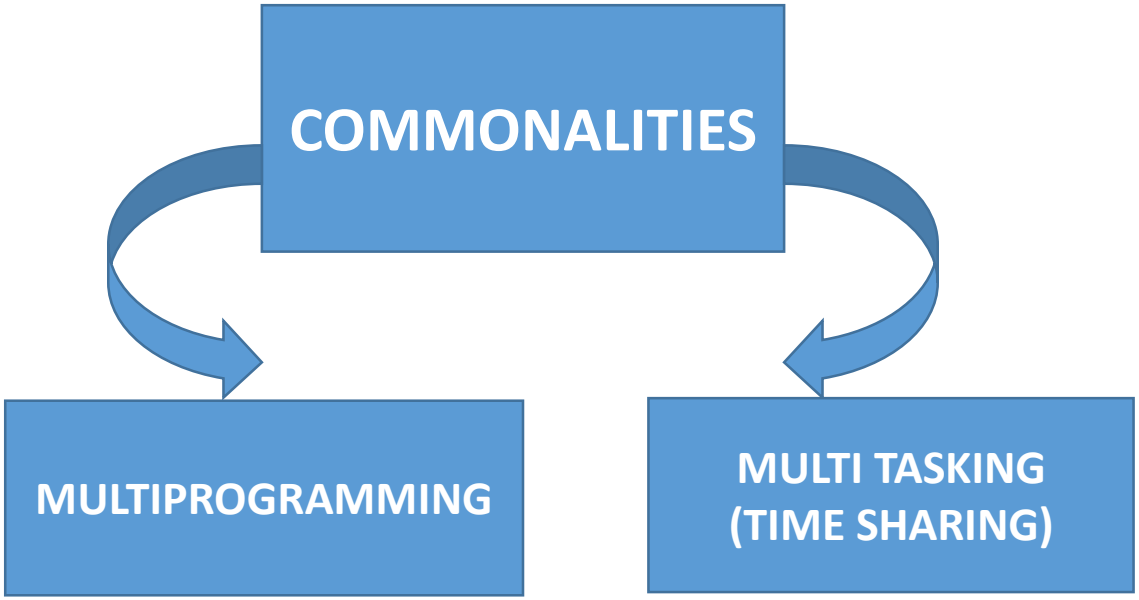
- ✓ To solve the above problem, DMA is used.
- ✓ After setting up buffers, pointers & counters for IO devices, the device controller transfers an entire block of data directly to or from its own buffer storage to memory without intervention of CPU.
- ✓ Only one interrupt is generated per block, to tell the device driver that the operation has completed.
- ✓ While the device controller is performing these operations, CPU is available to accomplish other works.

HOW A MODERN COMPUTER WORKS



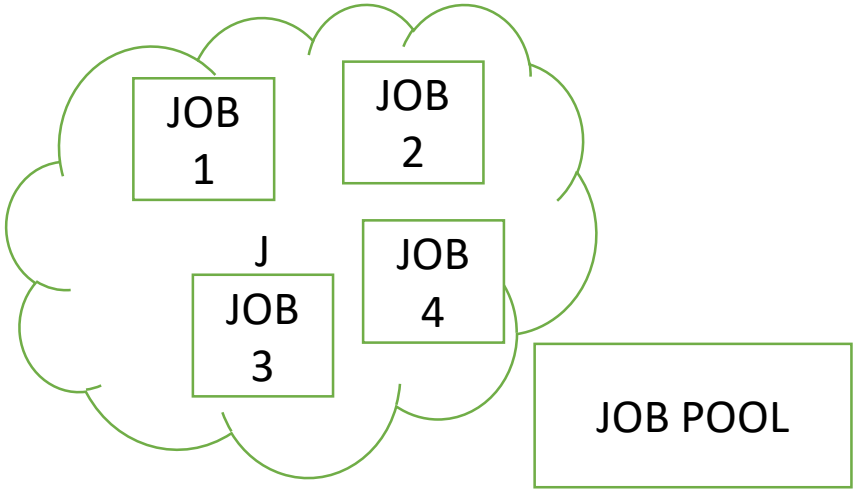
A Von-Neumann Arcitecture

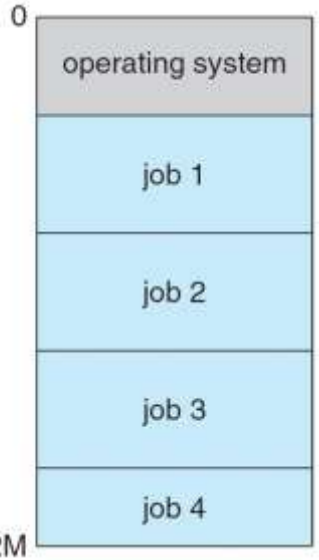
OPERATING SYSTEM STRUCTURE



OPERATING SYSTEM STRUCTURE

- ✓ **Multiprogramming** (**Batch system**) needed for efficiency.
- ✓ Capability of running multiple programs by the CPU.
- ✓ It increases CPU utilization by organizing jobs (code and data) so CPU always has one to execute.
- ✓ Single program (user) cannot keep CPU and I/O devices busy at all times.
- ✓ 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.





OPERATING SYSTEM STRUCTURE

- ✓ **Timesharing** (**multitasking**) is logical extension in which CPU switches occurs 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.

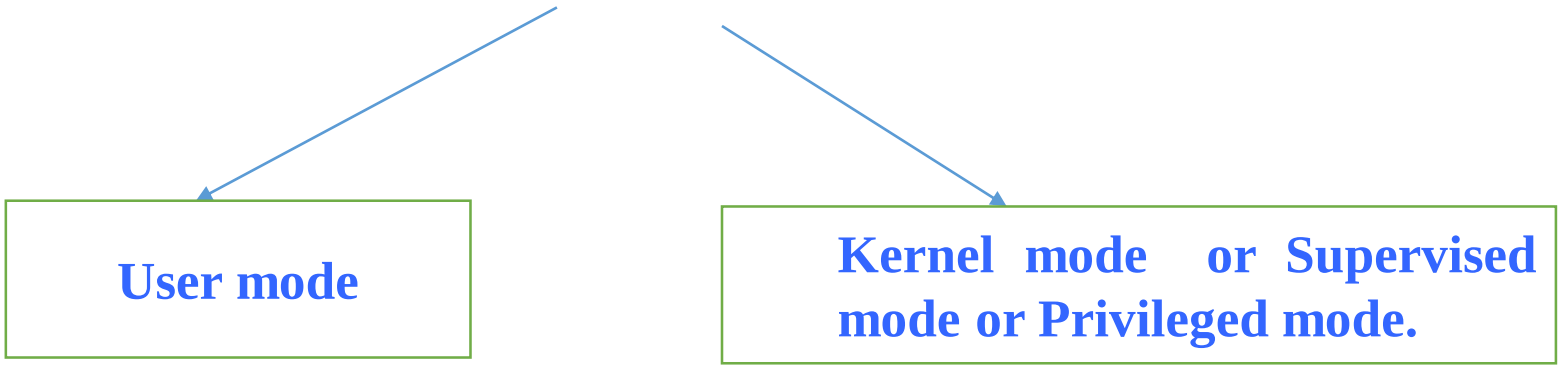
OPERATING SYSTEM OPERATIONS

- ✓ **Interrupt driven** (hardware and software)
  - ✓ Hardware interrupt by one of the devices
  - ✓ Software interrupt (**exception** or **trap):**
    - ✓ Software error (**e.g., division by zero or invalid memory access**)
    - ✓ Request for operating system service
    - ✓ For each type of interrupt separate segment of code in the OS determine what action should be taken.
    - ✓ ISR [Interrupt Service Routine] provided to deal with interrupt.
- ✓ A properly designed OS ensure that an incorrect or malicious program cannot cause other programs to execute incorrectly.

DUAL MODE & MULTI MODE OPERATIONS

✓ **Dual-mode** operation allows OS to protect itself and other system components.

2 MODES OF OPERATIONS

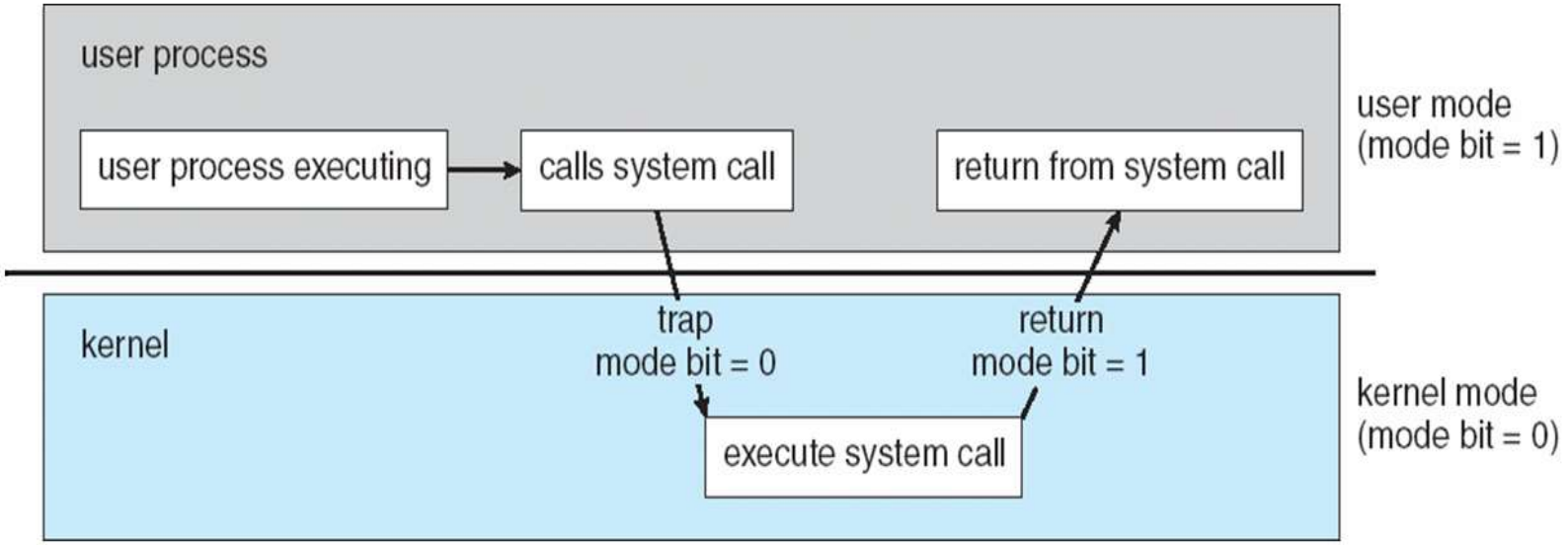


DUAL MODE & MULTI MODE OPERATIONS

Mode bit

- ✓ Added to hardware of the computer.
- ✓ To indicate the current mode
  - **- Kernel (0)**
  - **- User (1)**
- ✓ System executing a user application – User mode.
- ✓ User Application request a service from OS (via System call).
- ✓ System must translate from user mode to kernel mode to fulfill the request.

DUAL MODE & MULTI MODE OPERATIONS



SYSTEM BOOT

- ✓ Hardware starts in kernel mode.
- ✓ OS is then loaded & starts user application in user mode.
- ✓ Whenever trap or interrupt occurs ,the hardware switches from user mode to kernel mode.
- ✓ OS gains the control of computer in kernel mode.
- ✓ The system always switches to user mode before passing control to a user program.

PRIVILEGED INSTRUCTIONS (SYSTEM CALLS)

- ✓ The hardware allows these instructions to be executed only in kernel mode.

Examples:

- ✓ The instructions to switch to kernel mode.
- ✓ I/O control
- ✓ Timer management
- ✓ Interrupt management

BEYOND TWO MODES [MULTI MODE OPERATIONS]

- ✓ CPU uses more than one bit to set & test the mode.
- ✓ CPU supports virtualization frequently have a separate mode to indicate when the **Virtual Machine Manager(VMM)** & the **virtualization management software VMs** is in control of the system.
- ✓ VMM has more privileges than user processes but fewer than the kernel.

Example

- ✓ Intel 64 family of CPU supports 4 privileged levels & virtualization but does not have separate mode for virtualization.

SYSTEM CALL

- ✓ Provides the user program to ask the operating system to perform tasks reserved for the OS.
- ✓ It is invoked in variety of ways.

- Usually it takes the form of a trap to a specific location in the interrupt vector.
- Trap executed by a generic trap instruction, syscall instruction to invoke a system call.

- ✓ When system call is executed, hardware treated it as software interrupt.
- ✓ Control passes through interrupt vector to a service routine & mode bit is set to kernel mode.
- ✓ The kernel examines the interrupting instruction to determine what system call has occurred.

SYSTEM CALL

- ✓ A parameter indicates what type of service the user program is requesting.
- ✓ Additional information passed in registers, on the stack or in memory.
- ✓ Kernel verifies that parameters are correct & legal, execute the request & returns control to the instruction following the system call.

No bit mode

- Lack of hardware supported with dual mode cause serious short comings.

Example

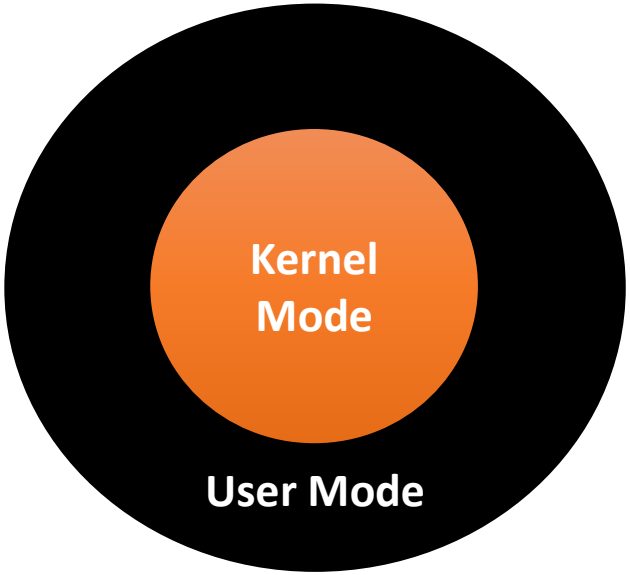
- MS DOS – Intel Architecture has no mode bit & no dual mode.

Example for Dual mode

- Windows 10, UNIX, Linux.

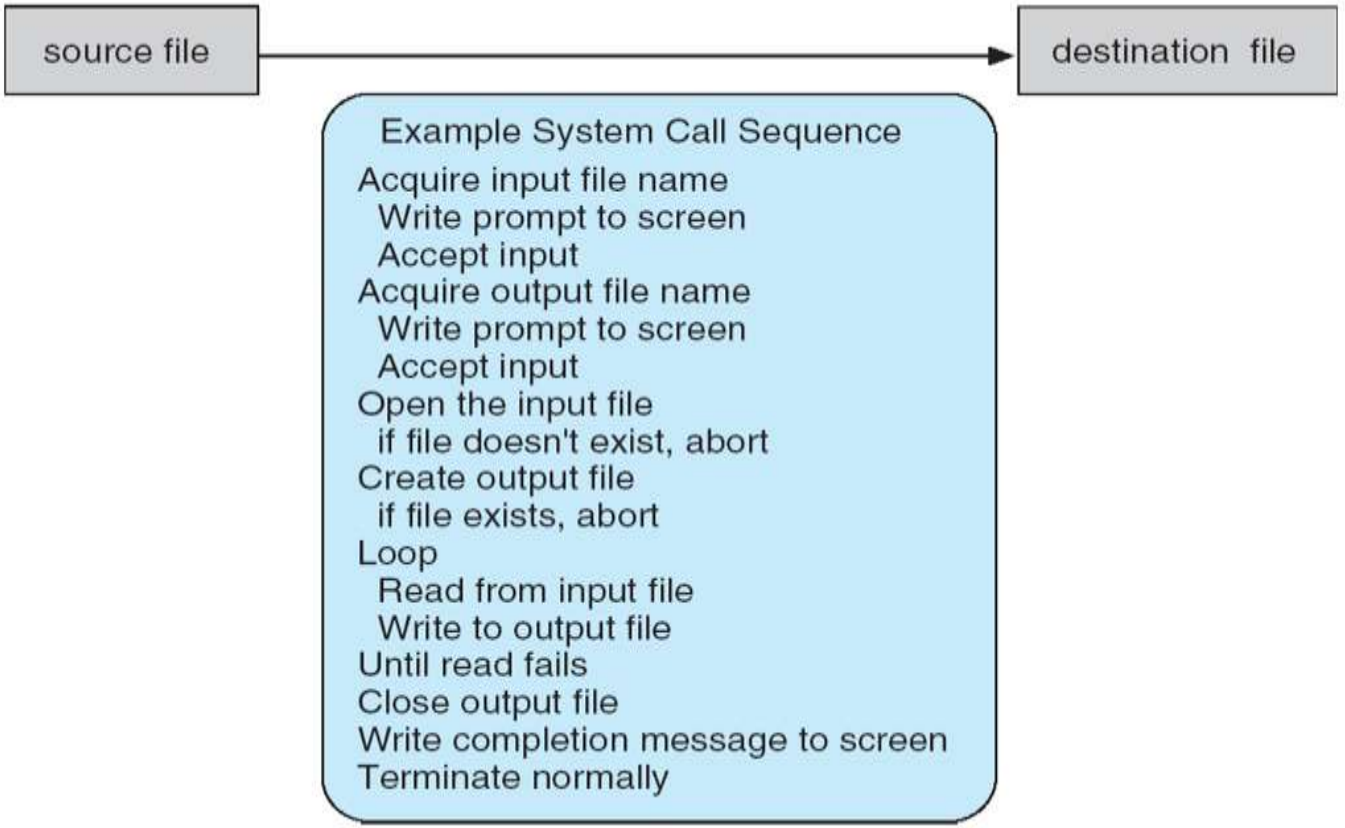
SYSTEM CALLS

- ✓ Programming interface to the services provided by the OS.
- ✓ Typically written in a high-level languages. (C or C++)



EXAMPLES - SYSTEM CALLS

- ✓ System call sequence to copy the contents of one file to another file.



EXAMPLES - SYSTEM CALLS

- ✓ Acquire input File name:
- ✓ To enter the file name
- ✓ 2 ways are there
  - Through keyboard
  - Mouse
- ✓ Frequently system execute thousands of system calls per second.

SYSTEM CALLS - APIs

- ✓ Specifies set of functions that are available to an application programmer, including parameters passed to each functions & return values the programmer can expect.
- ✓ Mostly accessed by programs via a high-level **Application Programming Interface (API)** rather than direct system call.
- ✓ Three most common APIs are
  1. Win32 API for Windows,
  2. POSIX API for POSIX-based systems (including virtually all versions of UNIX, Linux, and Mac OS X).
  3. Java API for the Java virtual machine. (JVM)

EXAMPLES OF STANDARD APIS

EXAMPLE OF STANDARD API

As an example of a standard API, consider the `read()` function that is available in UNIX and Linux systems. The API for this function is obtained from the `man` page by invoking the command

```
man read
```

on the command line. A description of this API appears below:

```
#include <unistd.h>

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

return value — function name — parameters

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

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

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

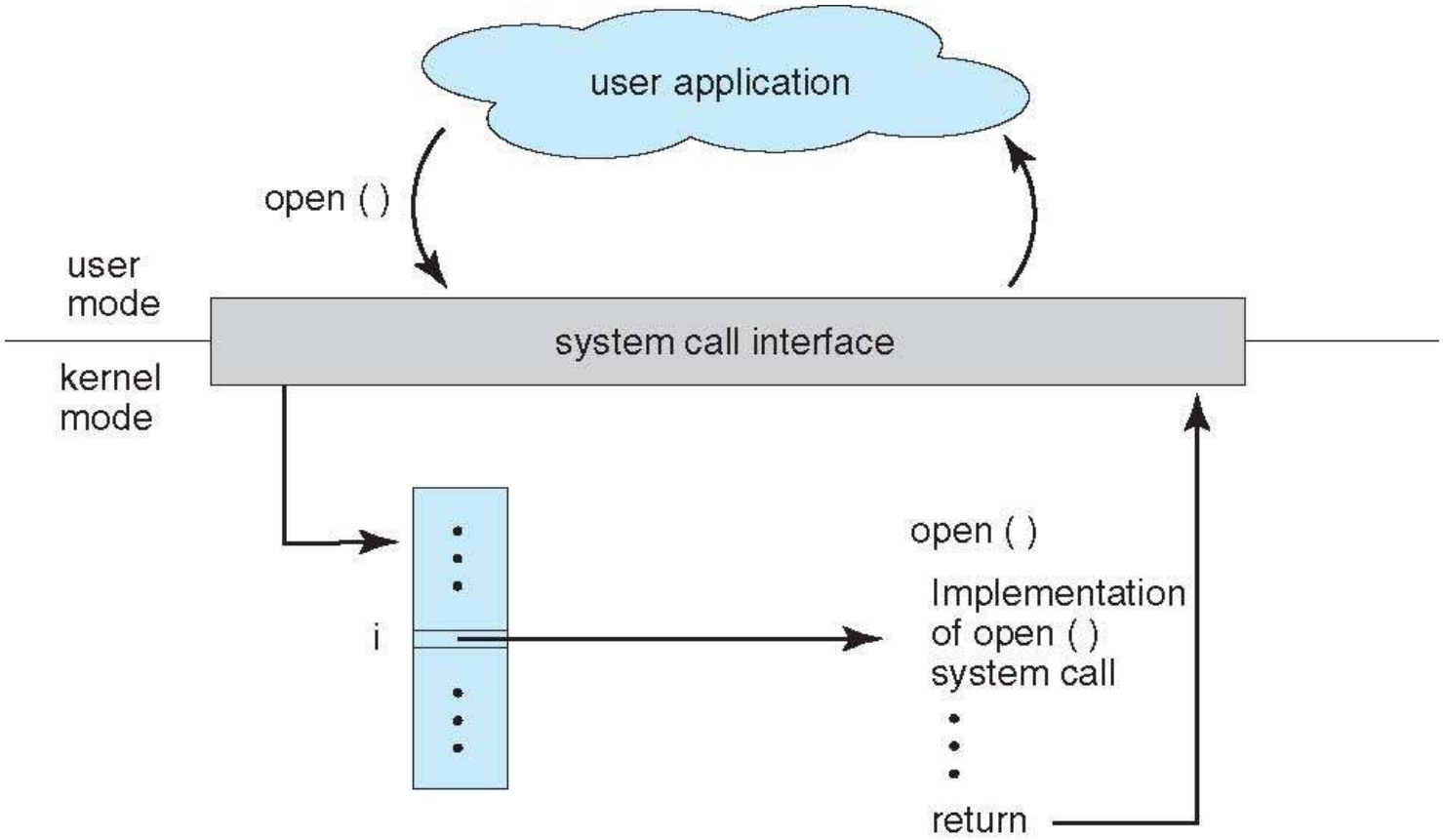
SYSTEM CALLS - APIs

- ✓ Why application programmer prefer programming according to API than invoking system calls?
- ✓ Program portability.
- ✓ Application programmer design a program using an API so as to expect their program to compile & run on any system that supports the same API.

SYSTEM CALLS INTERFACE

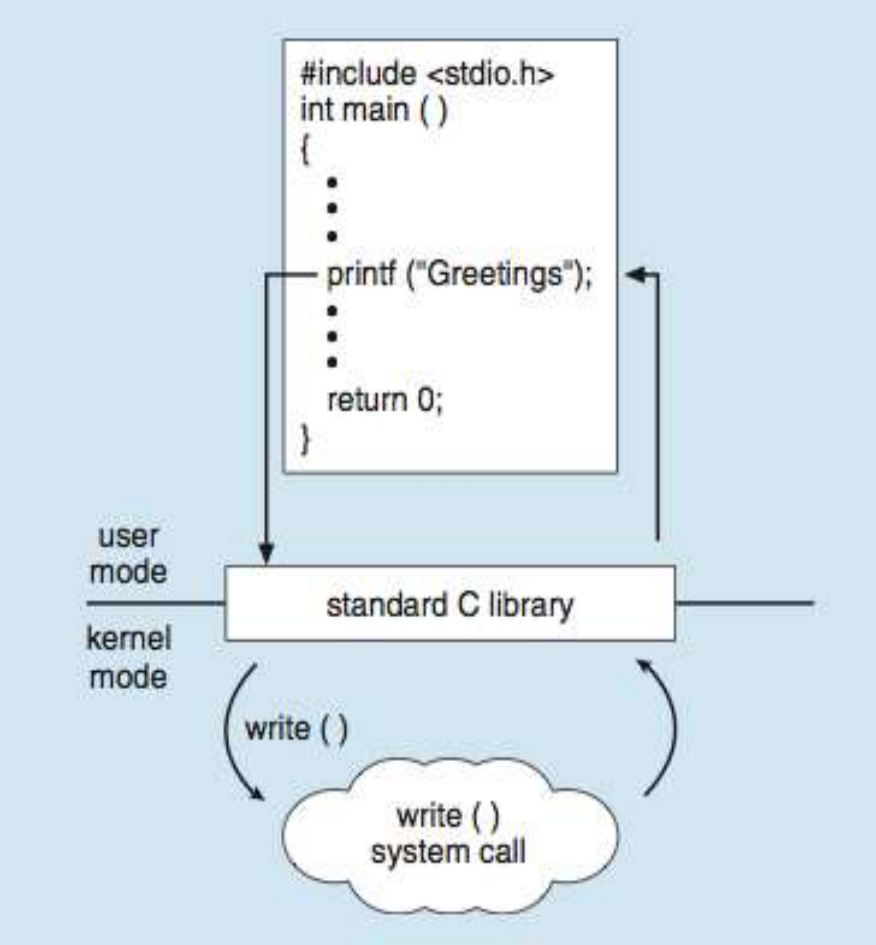
- ✓ Many programming languages provides a system call interface.
- ✓ It serves as the link to system calls made available by operating system.
- ✓ It intercepts function calls in the API & invokes the necessary system call written the OS.
- ✓ A number is associated with each system call.
- ✓ It maintains a table indexed according to these numbers.
- ✓ It invokes intended system call in OS kernel & returns status of the system calls & any return values.
- ✓ The caller need know nothing about how the system call is implemented.
- ✓ Just need only obey the API & understand what OS will do as a result of system call.
- ✓ Most detail of OS interface hidden from programmer by API.
- ✓ Managed by run time support library (set of functions built into libraries included. with compiler)

API – SYSTEM CALL – OS RELATIONSHIP



STANDARD C LIBRARY EXAMPLE

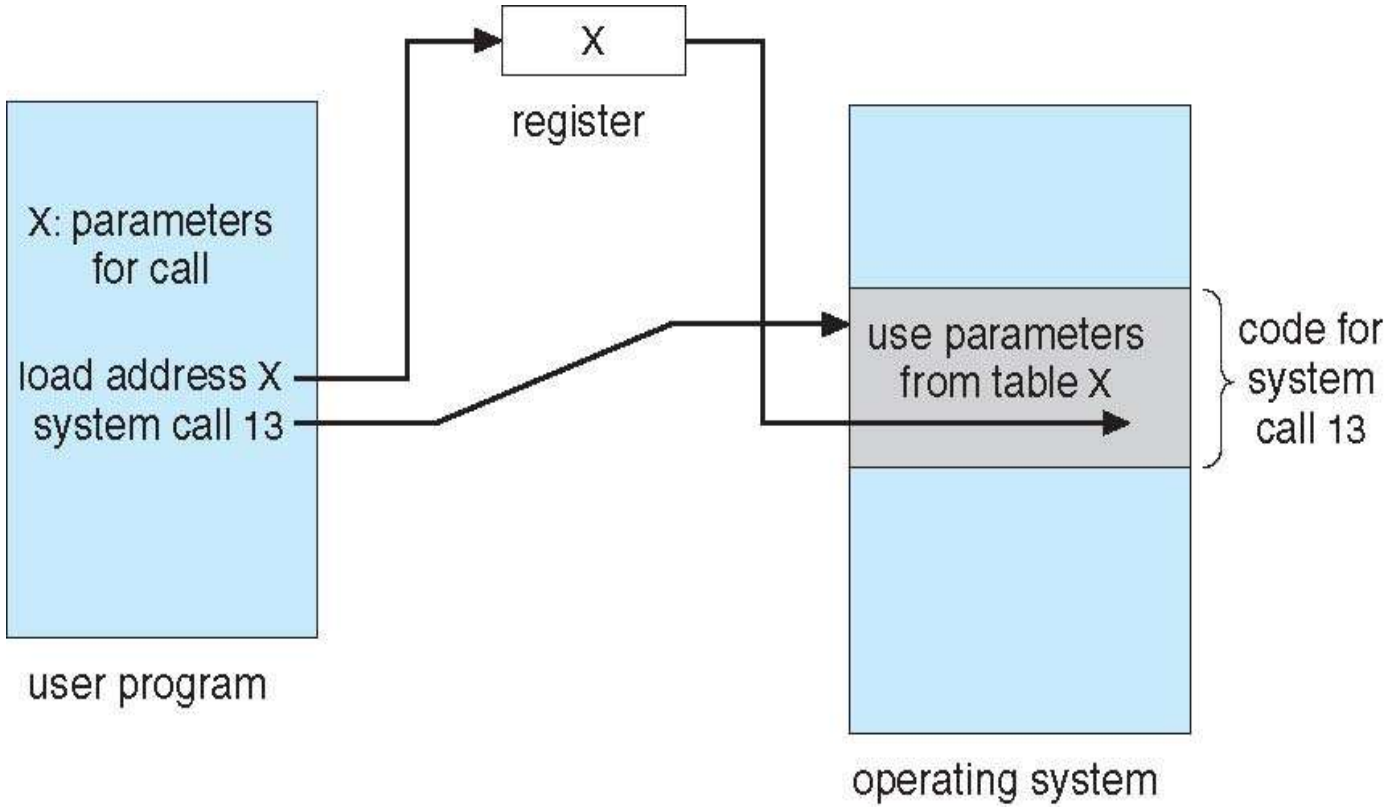
- ✓ C program invoking printf() library call, which calls write() system call.



SYSTEM CALLS PARAMETER PASSING

- ✓ Often, more information is required than simply identification of desired system call.
- ✓ Exact type and amount of information vary according to OS and call.
- ✓ Three general methods used to pass parameters to the OS
  - Simplest: pass the parameters in registers.
  - In some cases more parameters than registers are used.
    - ✓ Parameters stored in a block, or table, in memory, and address of block passed as a parameter in a register.
    - ✓ This approach taken by Linux and Solaris.
    - ✓ Parameters placed, or **pushed**, onto the **stack** by the program and **popped** off the stack by the operating system.
    - ✓ Block and stack methods do not limit the number or length of parameters being passed.

SYSTEM CALLS PARAMETER PASSING



TYPES OF SYSTEM CALLS

- ✓ **Process control**
  - create process, terminate process
  - end, abort
  - load, execute
  - get process attributes, set process attributes
  - wait for time
  - wait event, signal event
  - allocate and free memory
  - Dump memory if error
  - **Debugger** for determining **bugs, single step** execution
  - **Locks** for managing access to shared data between processes

TYPES OF SYSTEM CALLS

- ✓ **File management**
  - create file, delete file
  - open, close file
  - read, write, reposition
  - get and set file attributes
- ✓ **Device management**
  - request device, release device
  - read, write, reposition
  - get device attributes, set device attributes
  - logically attach or detach devices

TYPES OF SYSTEM CALLS

✓ **Information maintenance**

- get time or date, set time or date.
- get system data, set system data.
- get and set process, file, or device attributes.

✓ **Communications**

- create, delete communication connection.
- send, receive messages if **message passing model** to **host name** or **process name.**

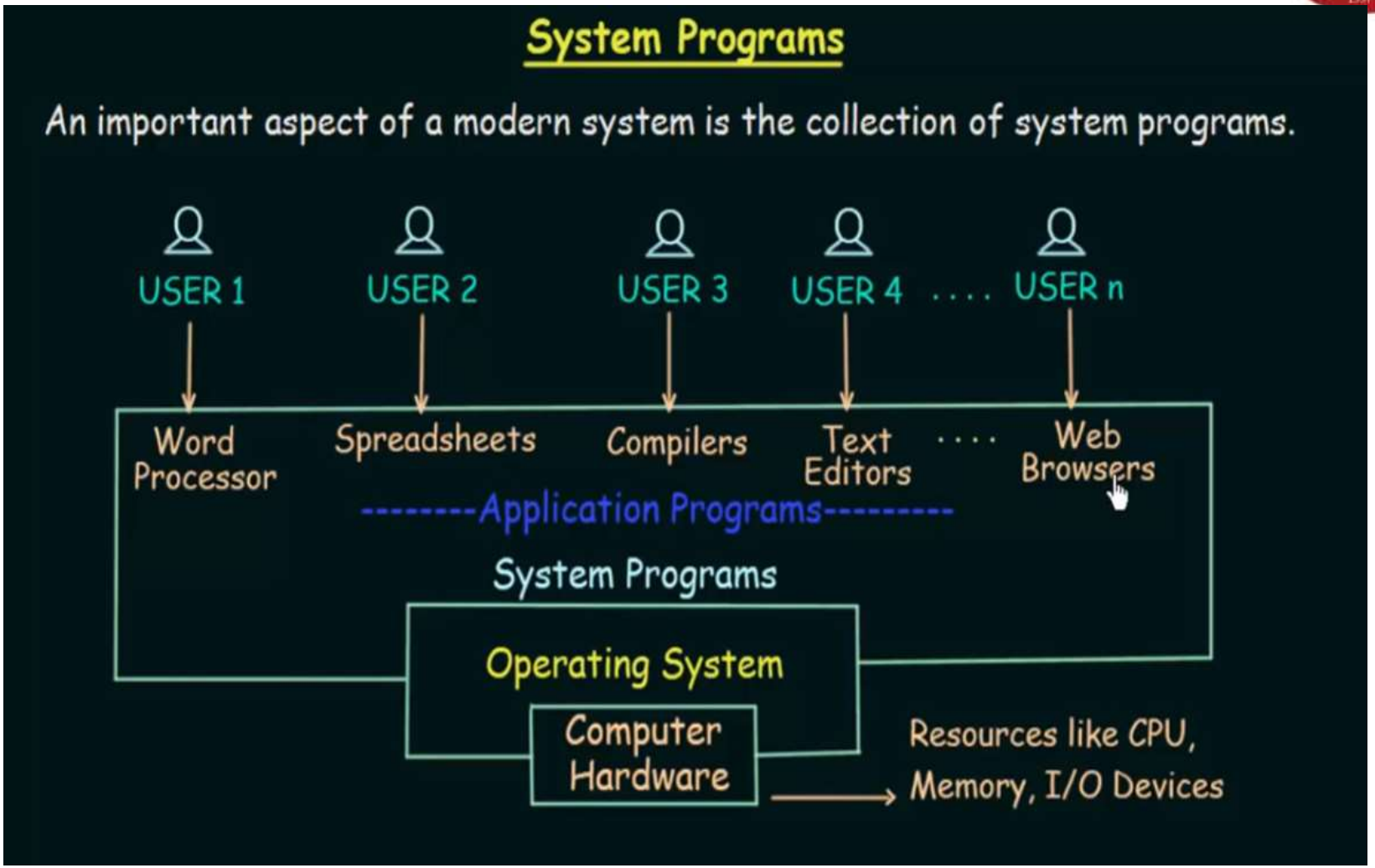
From **client** to **server**

- **Shared-memory model** create and gain access to memory regions
- transfer status information.
- attach and detach remote devices.

EXAMPLES OF WINDOWS AND UNIX SYSTEM CALLS

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

SYSTEM PROGRAMS



SYSTEM PROGRAMS - DEFINITION

- ✓ Provide a convenient environment for program development and execution.
  - Some of them are simply user interfaces to system calls;
  - others are considerably more complex.
- ✓ System programs are divided into following categories
  - File manipulation
  - Status information sometimes stored in a File modification
  - Programming language support
  - Program loading and execution
  - Communications
  - Background services
  - Application programs.

SYSTEM PROGRAMS - CATEGORIES

File Management

- ✓ Create,
- ✓ delete,
- ✓ copy,
- ✓ rename,
- ✓ print,
- ✓ dump,
- ✓ list, and
- ✓ generally manipulate files and directories.

SYSTEM PROGRAMS - CATEGORIES

✓**Status information**

- Ask the system for info

✓**Simple information's**

- date,
- time,
- amount of available memory,
- disk space,
- number of users

✓**Complex information's**

- Detailed performance
- logging,
- debugging information.

SYSTEM PROGRAMS - CATEGORIES

✓File modification

- Several Text editors may be available to create and modify the content of files stored on disk or other storage devices.
- Special commands to search contents of files or perform transformations of the text

SYSTEM PROGRAMS - CATEGORIES

✓**Programming-language support**

- Compilers,
- Assemblers,
- Debuggers and
- Interpreters
- For some common programming languages (such as C, C++, JAVA, Visual Basic & PERL)
- Provided to the user with the operating system.

SYSTEM PROGRAMS - CATEGORIES

✓**Program loading and execution**

- Once a program is assembled or compiled , it must be loaded into memory to be executed.
- Absolute loaders,
- Relocatable loaders,
- linkage editors, and
- overlay-loaders,
- Debugging systems for higher-level or machine language are needed.

Thank You