

## ICSC332: Operating Systems (40 hrs total)

### Introduction to basic concepts (3hrs)

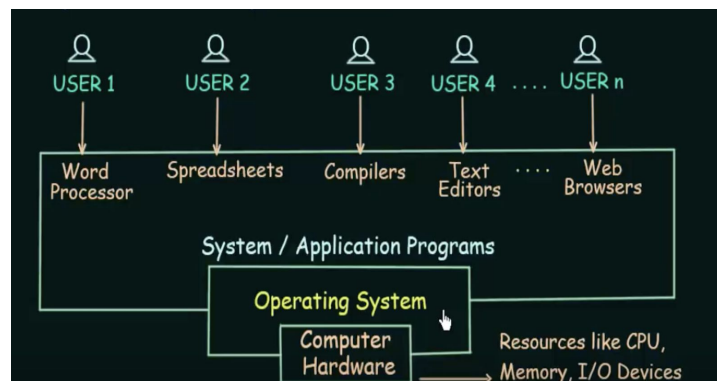
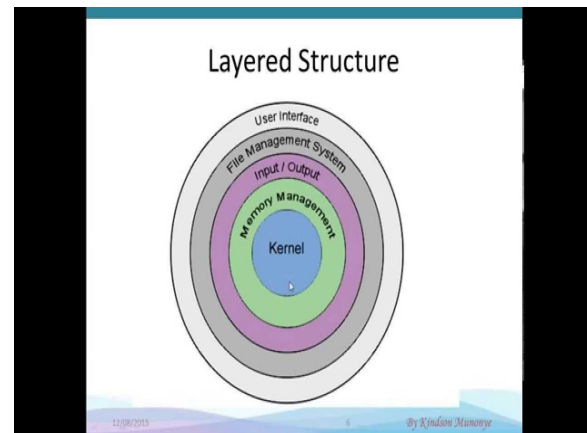
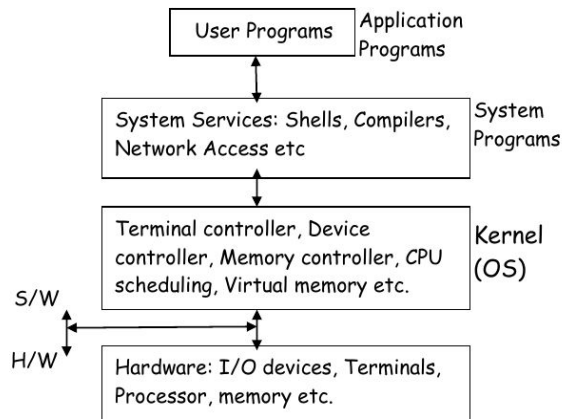
Interrupt processing, single user systems, multi-user systems, I/O systems

#### Introduction:

**Operating System (OS)** is a program that *manages* the computer hardware.

- It also provides a basis for application programs and acts as an *intermediary* between the computer User and computer Hardware.
- Examples of Operating Systems:
  - Windows
  - Linux (Open Source)
  - Ubuntu (Open Source)
  - Mac OS X (MacBook)
  - iOS (i phones)
  - Android

#### Structure of an Operating System:



### *Computer Hardware:*

- CPU
- Memory (RAMs)
- I/O devices (input/output devices)
  - Input Devices:
    - Keyboards
    - Mouse
    - Microphones
  - Output Devices:
    - Monitors
    - Speaker

### *Application Programs/Systems = Can be directly used by User:*

- Word Processor | Ex. Microsoft Word
- Spreadsheets | Ex. Microsoft Excel
- Text Editors | Ex. Visual Studio Code (IDEs)
- TextEditors | Ex. Notepad
- Web Browser | Ex. Google Chrome

### *Users:*

- Person who uses the application!

**\*\*\*The operating system does all the hard work for us! Works as an *intermediary* between the User and the Hardware!**

### *Types of Operating System (OS):*

- Batch OS
- Time Sharing OS
- Distributed OS
- Network OS
- Real Time OS
- MultiProgramming OS/ Processing/ Tasking OS (Look @classnotes 10/09)

### *Functions of Operating System (OS):*

- It is an interface between the User & Hardware
- Allocation of Resources
- Management of Memory, Security, etc

### *Goals of Operating System (OS):*

- Convenience
- Efficiency
- Both

### ***Basics of Operating System (OS):***

Some basic knowledge of the structure of Computer System is required to understand how Operating Systems work.

→ A modern general purpose computer system consists of one or more *CPUs* (depending on the power of your computer) and a number of *device controllers* connected through a common bus that provides access to *shared memory*.

*CPU* (Central Processing Unit) → Brain of the Computer

*Device Controllers:*

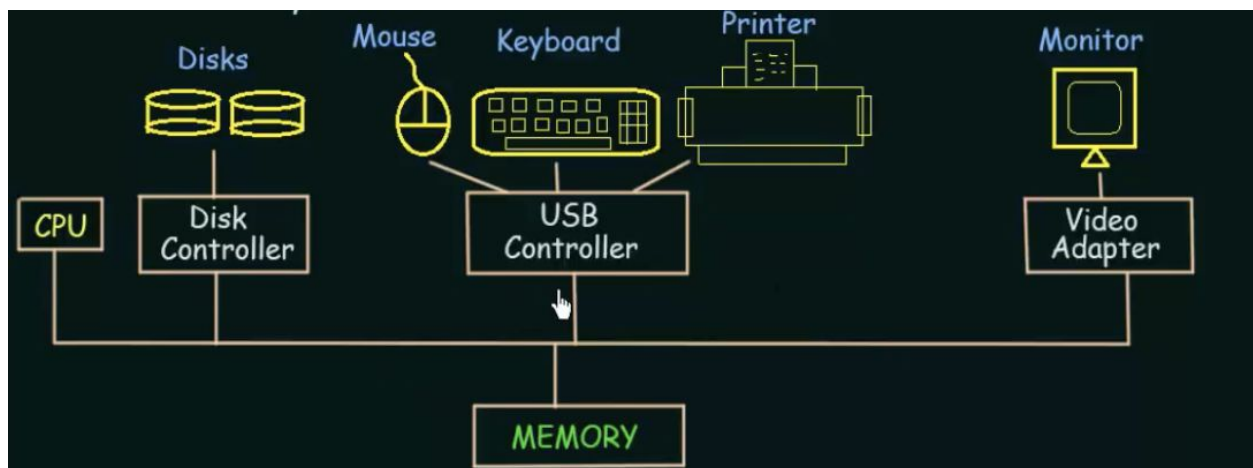
- Disk Controller:
  - Disk
- USB Controller
  - Mouse
  - Keyboard
  - Printer
- Video Adapter
  - Monitor

*Shared Memory*

→ Each device controller is in charge of a specific type of device

→ The CPU and the device controllers can execute concurrently, competing for memory cycles

→ To ensure orderly access to the shared memory, a memory controller is provided whose function is to synchronize access to the memory



## SOME IMPORTANT TERMS:

- 1) Bootstrap Program:
  - a) The initial program that runs when a computer is powered up or rebooted
  - b) It is stored in the ROM (Read Only Memory)
  - c) It must know how to run the OS and start executing that system
  - d) It must locate and load into memory the OS kernel (HEART OF THE OS)
- 2) Interrupt:
  - a) The occurrence of an event is usually signalled by an Interrupt from Hardware or Software
  - b) Hardware may trigger an interrupt at any time by sending a signal to the CPU, usually by the way of the system bus
- 3) System Call (Monitor Call):
  - a) Software may trigger an interrupt by executing a special operation called System Call

When the CPU is interrupted, it stops what it was doing and immediately transfers execution to a *fixed location*.

The fixed location usually contains the starting address where the service routine of the interrupt is located

The Interrupt Service Routine Executes.

On completion, the CPU resumes the interrupted computation

## ***Input/Output structure (I/O):***

- Storage is only 1 of the many types of I/O devices w/in a computer.
- A large portion of OS code is dedicated to managing I/O, both b/c of its importance to the **reliability** and **performance** of a system and b/c of the varying nature of the devices.
- A general purpose computer system consists of CPUs and multiple *device controllers* that are connected thru a common bus.
- Each device controller (maintains the local buffer storage & set of special purpose registers) is in charge of a specific type of device
  - One way of communicating with devices is through **registers** associated with each port. Registers may be one to four bytes in size, and may typically include ( a subset of ) the following four:
    - The ***data-in register*** is read by the host to get input from the device.
    - The ***data-out register*** is written by the host to send output.
    - The ***status register*** has bits read by the host to ascertain the status of the device, such as idle, ready for input, busy, error, transaction complete, etc.

- Working of an I/O Operation:

The diagram illustrates the working of an I/O operation. It shows a CPU (Thread of execution) connected to a cache. The cache is connected to a block containing 'Instructions and Data' and 'MEMORY'. Arrows show 'instruction execution cycle' between CPU and cache, and 'data movement' between cache and the block. A 'DMA' arrow points from the 'Device' (represented by a star) to the 'MEMORY' block. The CPU sends an 'i/o request' to the Device and receives 'data' back. The Device sends an 'interrupt' to the CPU.

  - > To start an I/O operation, the device driver loads the appropriate registers within the device controller
  - > The device controller, in turn, examines the contents 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 operation
  - > The device driver then returns control to the operating system

This form of interrupt-driven I/O is fine for moving small amounts of data but can produce high overhead when used for bulk data movement

To solve this problem, Direct Memory Access (DMA) is used

  - > After setting up buffers, pointers, and counters for the I/O device, the device controller transfers an entire block of data directly to or from its own buffer storage to memory, with no intervention by the CPU

For devices that transfer large quantities of data ( such as disk controllers ), it is wasteful to tie up the CPU transferring data in and out of registers one byte at a time. Instead this work can be off-loaded to a special processor, known as the ***Direct Memory Access, DMA, Controller***. DMA is used to solve the problem of interrupt-driven I/O when used for bulk data movement.

- The device controller transfers an entire block of data directly to or from its own buffer storage to memory, with no intervention of the CPU.
- One interrupt is generated per block using DMA, to tell the device driver that the operation is complete
- While the device controller is performing these operations, the CPU is available to accomplish other works. (Greatest advantage of DMA)

## Computer System Architecture:

Types of Computer Systems based on # of General Purpose Processors:

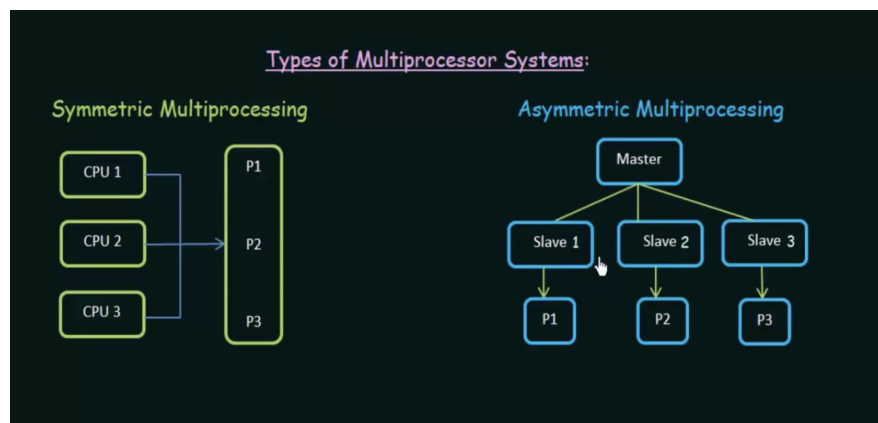
- 1) Single Processor Systems: 1 Processor
- 2) Multiprocessor Systems: 2+ Processors
- 3) Clustered Systems: 2+ Clustered/Coupled together Processors

Single Processor Systems:

- There is only 1 CPU capable of executing a *general purpose* instruction set including instructions from user processes.
- There are *special purpose processors* as well that are meant to perform device specific task. Ex. a microprocessors just for the keyboard

Multiprocessor Systems:

- Known as parallel systems or tightly coupled systems.
- Has 2 or more processors which requires the processors to closely communicate, sharing the computer bus and sometimes the clock, memory and peripheral devices aka resources.
- Advantages: Increased throughput (More CPUs so more efficient & faster) and economy scale (more affordable to have/maintain than single processors) Increased reliability (we have more processors to depend on)
- Two types of Multiprocessing systems:

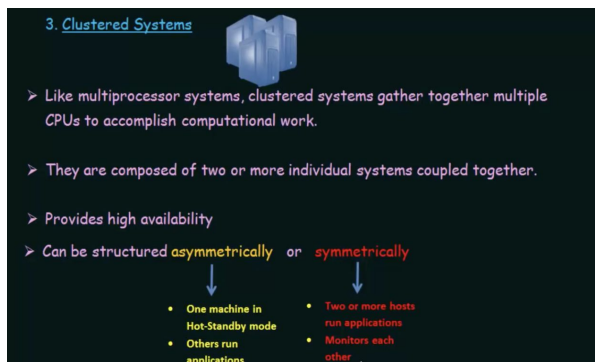


The multiple-processor systems in use today are of two types. Some systems use [asymmetric multiprocessing](#), in which each processor is assigned a specific task. A master processor controls the system; the other processors either look to the master for instruction or have predefined tasks. This scheme defines [a master-slave relationship](#). The master processor schedules and allocates work to the slave processors. The most common systems use [symmetric multiprocessing \(SMP\)](#),

in which each processor performs all tasks within the operating system. SMP means that all processors are peers; no master-slave relationship exists between processors.

#### Clustered Systems:

- Like multiprocessor systems, clustered systems gather together multiple CPUs to accomplish computational work.
- They are composed of two or more individual systems coupled together
- Provides high availability. Because even if one processor goes down there are multiple available to ensure the process/job is done.



#### Operating System Structure:

##### Multiprogramming & Multitasking:

- OS vary greatly in their makeup internally
- However they have **commonalities**:
  - Multiprogramming
  - Time Sharing (Multitasking)

##### Multiprogramming:

- A single user cannot in general, keep either the CPU or the I/O devices busy at all times.
- Multiprogramming *increases* CPU utilization by organizing jobs (code and data) so that the CPU always has one to execute.
- Multiprogramming systems provide an environment in which various system resources (ex. CPU, memory, and peripheral devices) are utilized efficiently, but they do not provide for user interaction with the computer system.

##### Time Sharing (Multitasking):

- CPU executes multiple jobs by switching among them
- 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 b/w the user and the system. (diff b/w multiprogramming and multitasking)

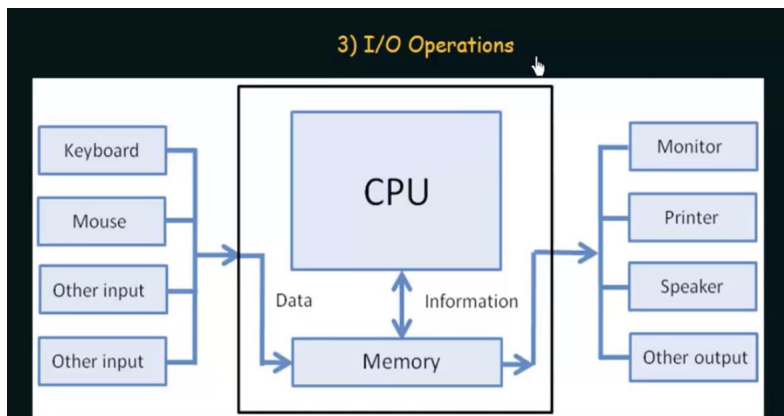
- A time shared operating system allows many users to share the computer simultaneously.

Uses CPU scheduling and multiprogramming to provide each user with a small portion of a time-shared computer. Each user has at least one separate program in memory. A program loaded into memory and executing is called a “*process*”. In a time-sharing system, the operating system must ensure reasonable response time. This is sometimes accomplished through **swapping**, in which processes are swapped in and out of main memory to the disk. A more common method for achieving this goal is **virtual memory**, a technique that allows the execution of a process that is not completely in memory.

The 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. This arrangement frees programmers from concern over memory-storage limitations.

### ***Operating System Services:***

- An OS provides an environment for the execution of programs.
  - It provides certain services to programs and to users of those programs.
- 1) **User Interface** is something that allows the user to interact with the OS or the computer itself
    - a) Command Line Interface (CLI)
      - i) Terminal
      - ii) Command Prompt
    - b) Graphical User Interface (GUI)
  - 2) **Program Execution** the OS must provide for the execution of the programs (be able to run the program)
  - 3) **I/O Operations** OS controls the usage of the I/O



- 4) **File System Manipulation** (ex cd, ls, etc)



- 5) Communications
- 6) Error Detection
- 7) Resource Allocation
- 8) Accounting (tracking user, and the usage of resources by each user)
- 9) Protection and Security

### ***Operating System Operations:***

Modern operating systems are **interrupt driven**. If there are no processes to execute, no I/O devices to service, and no users to whom to respond, an operating system will sit quietly, waiting for something to happen. Events are almost always signaled by the occurrence of an interrupt or a trap. A **trap (or an exception)** is a software-generated interrupt caused either by an error (for example, division by zero or invalid memory access) or by a specific request from a user program that an operating-system service be performed. The interrupt-driven nature of an operating system defines that system's general structure. For each type of interrupt, separate segments of code in the operating system determine what action should be taken. An interrupt service routine is provided that is responsible for dealing with the interrupt.

### ***User Operating System Services:***

There are two fundamental approaches for users to interface with the OS:

- 1) Provide a **Command Line Interface** that allows users to directly enter commands that are to be performed by the OS.
- 2) Allows the user to interface with the OS via a Graphical User Interface (GUI).

Command Interpreter:

- Some OSs include the command interpreter in the kernel.
- On systems with multiple command interpreters to choose from, the interpreters are known as shells. Examples:
  - Bourne shell
  - C shell
  - BASH
  - Korn Shell, etc

### ***System Calls:***

System calls provide an *interface* to the services made available by an Operating System.

In order to ensure the proper execution of the operating system, we must be able to distinguish between the execution of operating-system code and user- defined code. The approach taken by

most computer systems is to provide hardware support that allows us to differentiate among various modes of execution.

At the very least, we need two separate **modes** of operation: **user mode** and **kernel mode** (also called **supervisor mode**, **system mode**, or **privileged mode**).

**Kernel Mode and User Mode** are two modes in which a program can execute.

*User mode:* A program does not have direct access to the memory to the hardware and such resources. *If a crash occurs in User mode the entire system does not crash/ therefore user mode is safer.*

*Kernel Mode:* A program has direct access to the memory, to the hardware, and such resources. Because kernel mode has access to all of these resources we consider it as a privileged mode. *If a crash occurs in Kernel during execution then the entire system would crash/halt.*

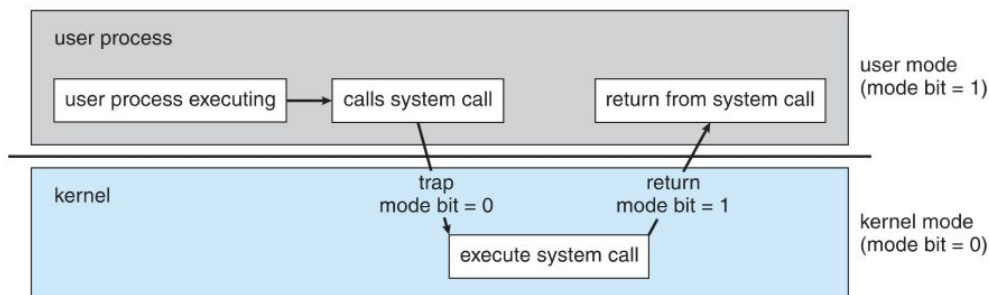
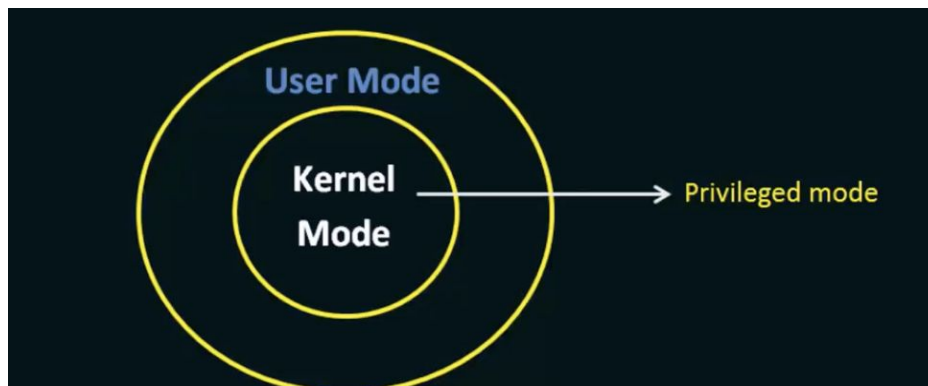


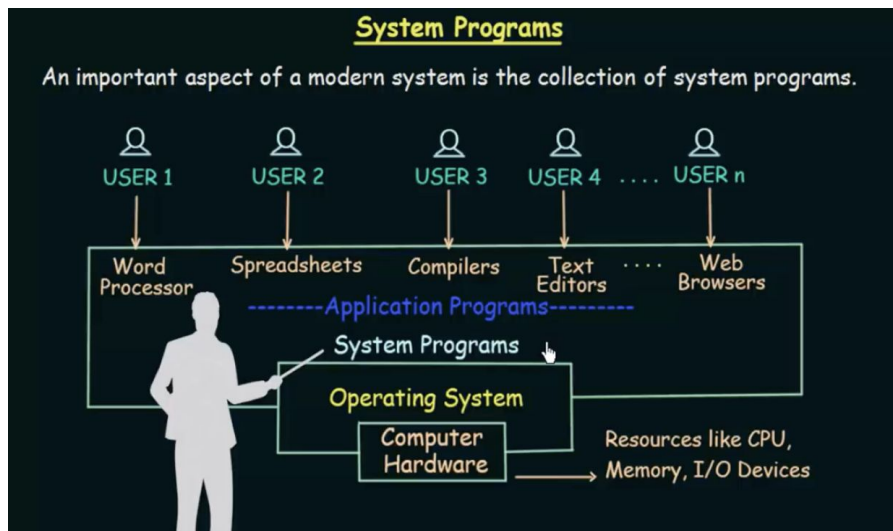
Figure 1.10: Transition from user to kernel mode.

- When User need resources from hernel this is called context switching.
- System call is the programatic way in which a computer program request service from the kernel of the OS.
- These calls are generally available as routines written in C and C++

### Types of System Calls:

1. Process Control (System Calls used to control processes aka programs)

- a. End, abort
  - b. Load, execute
  - c. Create, terminate process
  - d. Get and set process attributes
  - e. Wait for time
  - f. Wait event, signal event
  - g. Allocate and free memory
2. File Manipulation
    - a. Create file, delete file
    - b. Open, close
    - c. Read, write, reposition
    - d. Get file attributes, set file attributes
  3. Device Management
    - a. Request and release device
    - b. Read, write, reposition
    - c. Get and set device attributes
    - d. Logically attach or detach devices
  4. Information Maintenance
    - a. Get or set time or date
    - b. Get or set system data
    - c. Get and set process, file, or device attributes
  5. Communication
    - a. Create delete communication connection
    - b. Send, receive message
    - c. Transfer status information
    - d. Attach or detach remote devices



## System Programs:

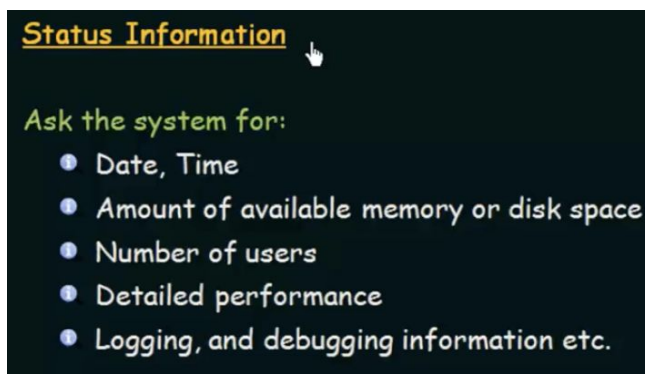
- System programs provide a convenient environment for program dev. And execution.
- Some of them are simply user interfaces to system calls.

System Programs can be divided into the following categories:

### 1. File Management



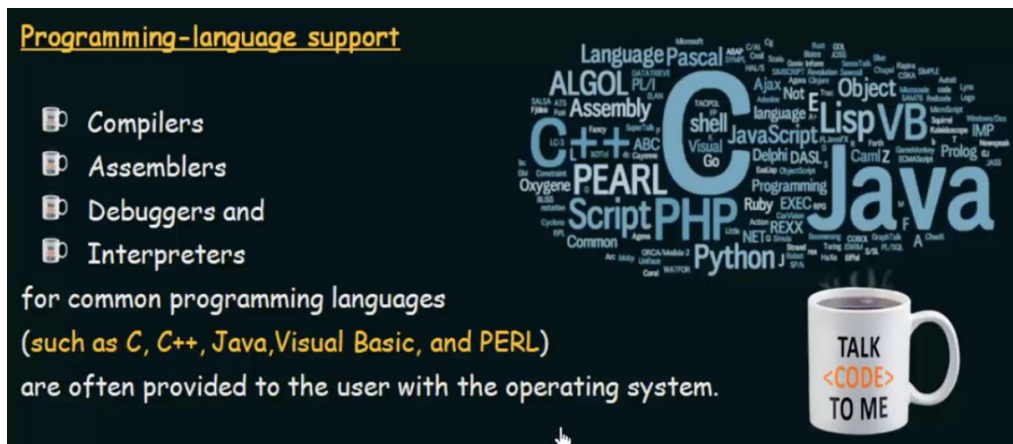
### 2. Status Information:



### 3. File Modification

- Several text editors are available to create and modify the content stored on disk or storage devices.

### 4. Programming- Language Support



## 5. Program Loading and Executions


Program loading and execution

Once a program is assembled or compiled, it must be loaded into memory to be executed.

The system may provide:

- Absolute loaders
- Relocatable loaders
- Linkage editors and
- Overlay loaders

Debugging systems for either higher-level languages or machine language are needed as well.




## 6. Communications

Communications

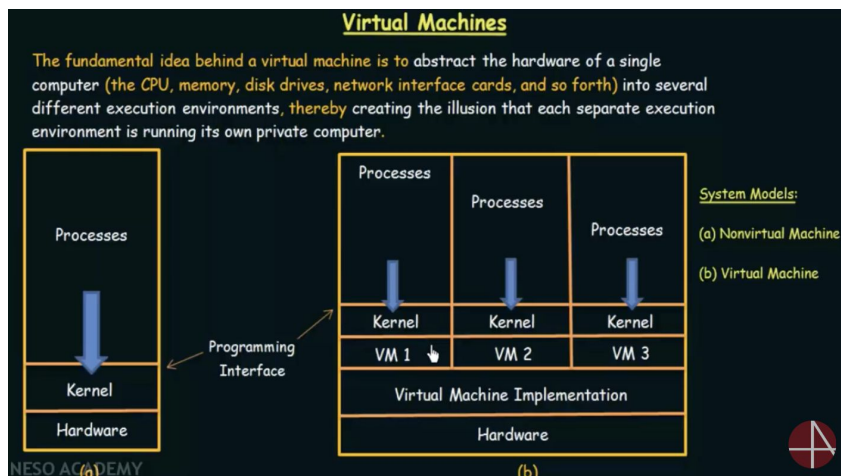
These programs provide the mechanism for:

- Creating virtual connections among processes, users, and computer systems.
- Allowing users to send messages to one another's screens
- To browse webpages
- To send electronic-mail messages
- To log in remotely or to transfer files from one machine to another.



Structures of OS: (Come back to the video)

Virtual Machines:



The VM Software is ran in Kernel Mode

VM itself (VM1, VM2, etc) are run in User Mode

**IMPLEMENTATION**

Virtual Machine Software -	Runs in Kernel mode
Virtual Machine itself -	Runs in User Mode

Just as the physical machine has two modes, however, so must the virtual machine.

Consequently, we must have:

- A virtual user mode and
- A virtual kernel mode

BOTH OF WHICH RUN IN A PHYSICAL USER MODE

Class specific terms for quiz:

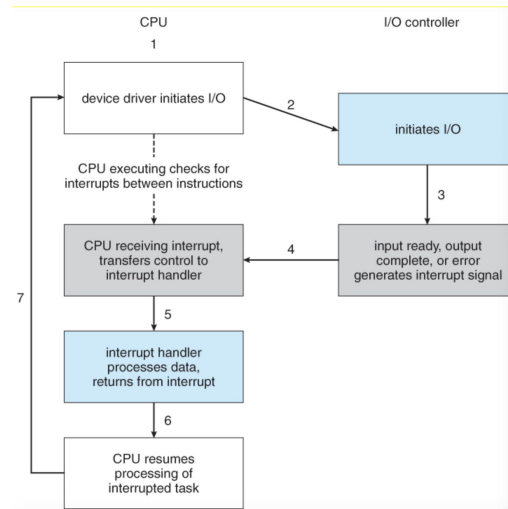
[https://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/13\\_IOSystems.html](https://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/13_IOSystems.html)

<http://csis.pace.edu/lixin/teaching/cs371/interrupt.pdf>

## Interrupts

- Interrupts allow devices to notify the CPU when they have data to transfer or when an operation is complete, allowing the CPU to perform other duties when no I/O transfers need its immediate attention.
- The CPU has an ***interrupt-request line*** that is sensed after every instruction.
  - A device's controller ***raises*** an interrupt by asserting a signal on the interrupt request line.
  - The CPU then performs a state save, and transfers control to the ***interrupt handler*** routine at a fixed address in memory. ( The CPU ***catches*** the interrupt and ***dispatches*** the interrupt handler. )
  - The interrupt handler determines the cause of the interrupt, performs the necessary processing, performs a state restore, and executes a ***return from interrupt*** instruction to return control to the CPU. ( The interrupt handler ***clears*** the interrupt by servicing the device. )

- ( Note that the state restored does not need to be the same state as the one that was saved when the interrupt went off. See below for an example involving time-slicing. )



### Interrupt-driven I/O cycle.

- The above description is adequate for simple interrupt-driven I/O, but there are three needs in modern computing which complicate the picture:
  - The need to defer interrupt handling during critical processing,
  - The need to determine *which* interrupt handler to invoke, without having to poll all devices to see which one needs attention, and
  - The need for multi-level interrupts, so the system can differentiate between high- and low-priority interrupts for proper response.
- These issues are handled in modern computer architectures with *interrupt-controller* hardware.
  - Most CPUs now have two interrupt-request lines: One that is *non-maskable* for critical error conditions and one that is *maskable*, that the CPU can temporarily ignore during critical processing.
  - The interrupt mechanism accepts an *address*, which is usually one of a small set of numbers for an offset into a table called the *interrupt vector*. This table ( usually located at physical address zero ? ) holds the addresses of routines prepared to process specific interrupts.
  - The number of possible interrupt handlers still exceeds the range of defined interrupt numbers, so multiple handlers can be *interrupt chained*. Effectively the addresses held in the interrupt vectors are the head pointers for linked-lists of interrupt handlers.



- Figure 13.4 shows the Intel Pentium interrupt vector. Interrupts 0 to 31 are non-maskable and reserved for serious hardware and other errors. Maskable interrupts, including normal device I/O interrupts begin at interrupt 32.
- Modern interrupt hardware also supports *interrupt priority levels*, allowing systems to mask off only lower-priority interrupts while servicing a high-priority interrupt, or conversely to allow a high-priority signal to interrupt the processing of a low-priority one.

### Blocking and Non-blocking I/O

- With *blocking I/O* a process is moved to the wait queue when an I/O request is made, and moved back to the ready queue when the request completes, allowing other processes to run in the meantime.
- With *non-blocking I/O* the I/O request returns immediately, whether the requested I/O operation has ( completely ) occurred or not. This allows the process to check for available data without getting hung completely if it is not there.

Caching involves keeping a *copy* of data in a faster-access location than where the data is normally stored.

### I/O Protection

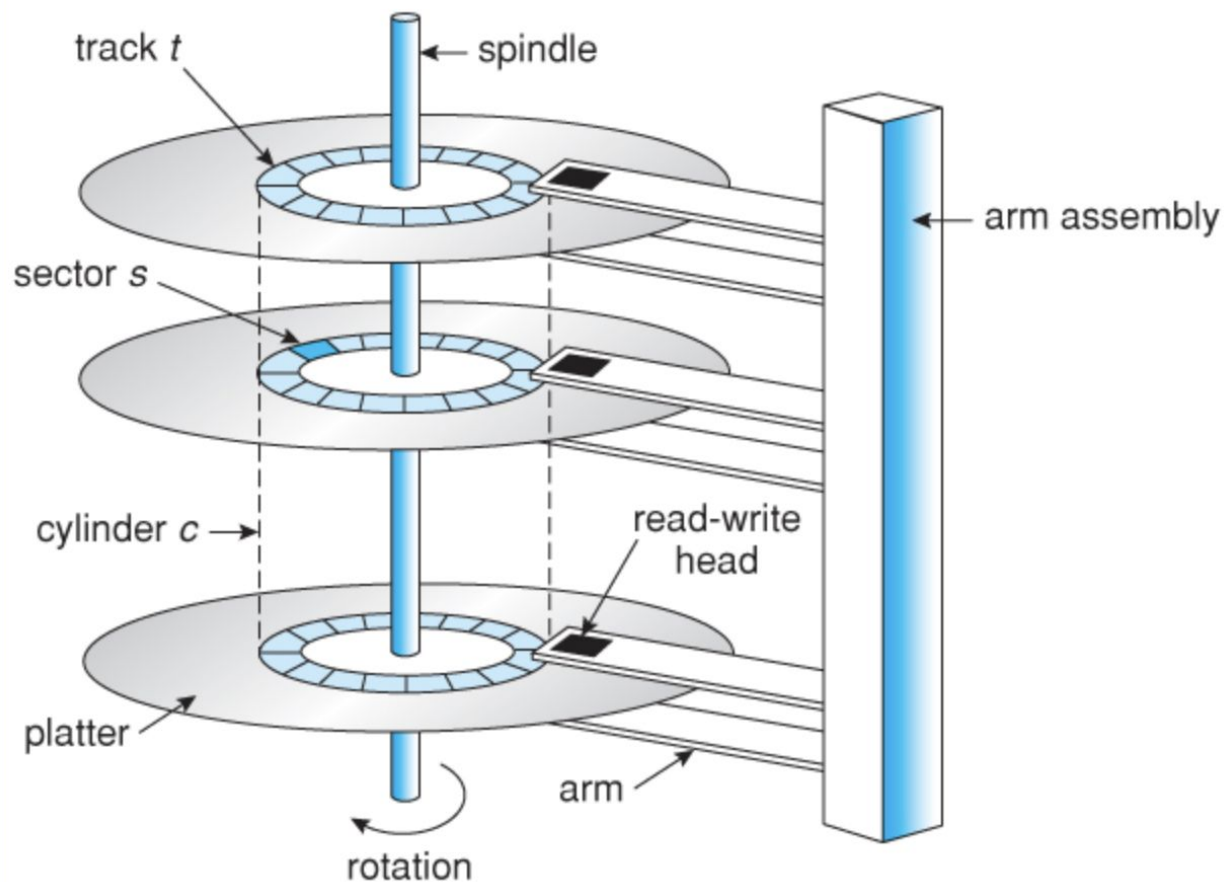
- The I/O system must protect against either accidental or deliberate erroneous I/O.
- User applications are not allowed to perform I/O in user mode - All I/O requests are handled through system calls that must be performed in kernel mode.
- Memory mapped areas and I/O ports must be protected by the memory management system, **but** access to these areas cannot be totally denied to user programs. ( Video games and some other applications need to be able to write directly to video memory for optimal performance for example. ) Instead the memory protection system restricts access so that only one process at a time can access particular parts of memory, such as the portion of the screen memory corresponding to a particular window.

### Magnetic Disks

- Traditional magnetic disks have the following basic structure:
  - One or more *platters* in the form of disks covered with magnetic media. *Hard disk* platters are made of rigid metal, while "*floppy*" disks are made of more flexible plastic.



- Each platter has two working **surfaces**. Older hard disk drives would sometimes not use the very top or bottom surface of a stack of platters, as these surfaces were more susceptible to potential damage.
- Each working surface is divided into a number of concentric rings called **tracks**. The collection of all tracks that are the same distance from the edge of the platter, ( i.e. all tracks immediately above one another in the following diagram ) is called a **cylinder**.
- Each track is further divided into **sectors**, traditionally containing 512 bytes of data each, although some modern disks occasionally use larger sector sizes. ( Sectors also include a header and a trailer, including checksum information among other things. Larger sector sizes reduce the fraction of the disk consumed by headers and trailers, but increase internal fragmentation and the amount of disk that must be marked bad in the case of errors. )
- The data on a hard drive is read by read-write **heads**. The standard configuration ( shown below ) uses one head per surface, each on a separate **arm**, and controlled by a common **arm assembly** which moves all heads simultaneously from one cylinder to another. ( Other configurations, including independent read-write heads, may speed up disk access, but involve serious technical difficulties. )
- The storage capacity of a traditional disk drive is equal to the number of heads ( i.e. the number of working surfaces ), times the number of tracks per surface, times the number of sectors per track, times the number of bytes per sector. A particular physical block of data is specified by providing the head-sector-cylinder number at which it is located.



- In operation the disk rotates at high speed, such as 7200 rpm ( 120 revolutions per second. ) The rate at which data can be transferred from the disk to the computer is composed of several steps:
  - The **positioning time**, a.k.a. the **seek time** or **random access time** is the time required to move the heads from one cylinder to another, and for the heads to settle down after the move. This is typically the slowest step in the process and the predominant bottleneck to overall transfer rates.
  - The **rotational latency** is the amount of time required for the desired sector to rotate around and come under the read-write head. This can range anywhere from zero to one full revolution, and on the average will equal one-half revolution. This is another physical step and is usually the second slowest step behind seek time. ( For a disk rotating at 7200 rpm, the average rotational latency would be  $1/2$  revolution / 120 revolutions per second, or just over 4 milliseconds, a long time by computer standards.
  - The **transfer rate**, which is the time required to move the data electronically from the disk to the computer. ( Some authors may also use the term transfer rate to

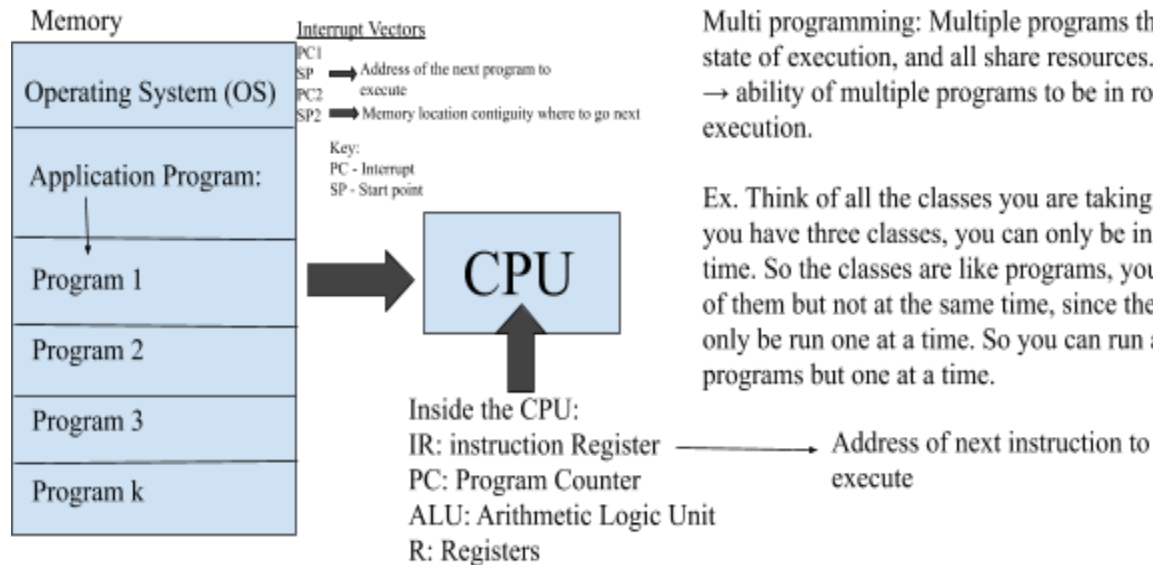
refer to the overall transfer rate, including seek time and rotational latency as well as the electronic data transfer rate. )

- Disk heads "fly" over the surface on a very thin cushion of air. If they should accidentally contact the disk, then a **head crash** occurs, which may or may not permanently damage the disk or even destroy it completely. For this reason it is normal to **park** the disk heads when turning a computer off, which means to move the heads off the disk or to an area of the disk where there is no data stored.
- Floppy disks are normally **removable**. Hard drives can also be removable, and some are even **hot-swappable**, meaning they can be removed while the computer is running, and a new hard drive inserted in their place.
- Disk drives are connected to the computer via a cable known as the **I/O Bus**. Some of the common interface formats include Enhanced Integrated Drive Electronics, EIDE; Advanced Technology Attachment, ATA; Serial ATA, SATA, Universal Serial Bus, USB; Fiber Channel, FC, and Small Computer Systems Interface, SCSI.
- The **host controller** is at the computer end of the I/O bus, and the **disk controller** is built into the disk itself. The CPU issues commands to the host controller via I/O ports. Data is transferred between the magnetic surface and onboard **cache** by the disk controller, and then the data is transferred from that cache to the host controller and the motherboard memory at electronic speeds.

## Class notes:

10/09/2019

# MULTI-Programming



Multi programming: Multiple programs that are in a state of execution, and all share resources.  
→ ability of multiple programs to be in rotate of execution.

Ex. Think of all the classes you are taking. Let's say you have three classes, you can only be in one class at time. So the classes are like programs, you can run all of them but not at the same time, since the CPU can only be run one at a time. So you can run all the programs but one at a time.

Operating System works as disciplinary ex. Cop or Parents...

Hardware interrupts is how the Operating System can limit/ control how much time each program has with the CPU and the interrupt gives back control to the OS so that it can pass access to another program to use the CPU for a limited amount of time, and the interrupt process occurs again after a certain amount of time.

10/29/2019

- Disk
- busy looping i/o
- interrupt driven i/o

Process: program, in execution

Process states:

- Running

- Ready
- Suspended
- Nascent
- Zombie

Create process - Nascent → get\_pID() - Running → I/O requested allocate CPU Suspended →  
I/O complete - Ready → Running

Running → exit - Zombie