Operating System

Created by	Ⓑ Borhan
Last edited time	@November 11, 2024 9:22 PM
Tag	Year 3 Term 1

Resources

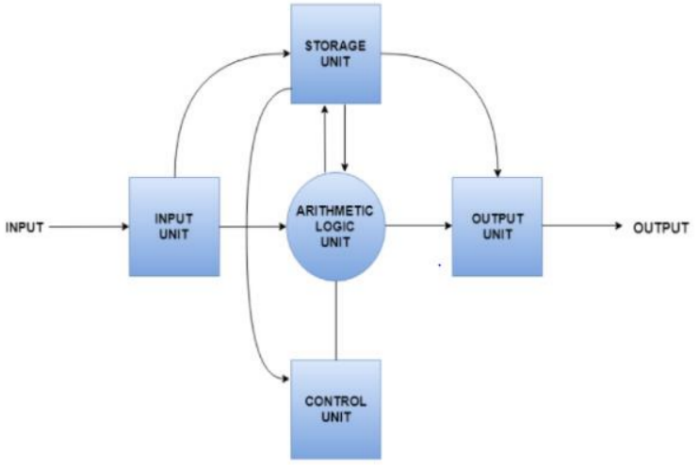
- https://medium.com/@algorhythm2411/operating-system-os-learning-resources-515d685ad1c3#:~:text=Books,Galvin%2C%20and%20Greg%20Gagne
- https://www.youtube.com/playlist?list=PLG9aCp4uE-s17rFjWM8KchGlffXgOzzVP
- https://www.youtube.com/watch?v=xw_OuOhjauw&list=PLmXKhU9FNesSFvj6gASuWmQd23UI5omtD&index=1&t=1527s&ab_channel=KnowledgeGATE
- https://www.poriyaan.in/paper/introduction-to-operating-systems-81/

Introduction & Basics of OS

Computer System Architecture

Computer System Architecture refers to the design and organization of the components of a computer system, including the hardware and software architecture that allows the system to function effectively.

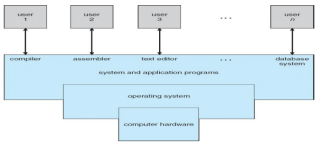
Components in Computer Architectures

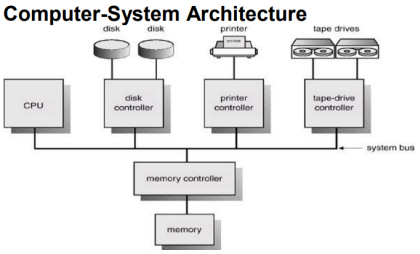


- Input Unit : It takes data from the input devices, converts it into machine language and then loads it into the computer system.
- Storage Unit: Storage unit contains many computer components that are used to store data. It is traditionally divided into primary storage and secondary storage.
- Arithmetic Logic Unit: All the calculations related to the computer system are performed by the arithmetic logic unit. It can perform operations like addition, subtraction, multiplication, division etc.
- Control Unit: This unit controls all the other units of the computer system and so is known as its central nervous system. It transfers data throughout the computer as required including from storage unit to central processing unit and vice versa.
- Output unit: This unit takes the processed data from the computer system and converts it into a format that can be understood by the user. It then sends this data to output devices such as monitors, printers, or speakers.

Computer System Structure/Components: Four Components

- Hardware
- Operating System
- Application system
- Users





Operating System

An Operating system is a program that **controls the execution of application** programs and acts as an **interface between the user of a computer and the computer hardware.**

More definition

- Software abstracting hardware
- Interface between user and hardware
- Set of utilities to simplify application development/execution
- Control program: controls the execution of user programs and operations of I/O devices
- Acts like a government
- Resource allocator: manages and allocates resources
- Kernel: The one program running at all times

Components of OS

- **Kernel :** Kernel is an active part of OS, running all times and can interact with the hardware.
- **Shell:** Shell is a computer program that exposes an operating system's services to a human user or other programs. The shell is nothing more than a program that carries the user typed commands or instructions from the terminal and converts them into something that the kernel can understand.
  - GUI (graphical user interface) : A graphical user interface (GUI) provides means fo**r manipulating programs graphically**, by allowing for operations such as opening, closing, moving and resizing windows,
  - CLI (command-line interface) : A command-line interface (CLI) is an operating system shell that **uses alphanumeric characters typed** on a keyboard to provide instructions and data to the operating system, interactively.

Functions of Operating Systems

- **Process Management**
  - creation, deletion, suspension and resumption
  - Provision
    - process synchronization, process communication
- **Memory Management (RAM)**
  - Keep track of which parts of memory are currently being used and by whom

    - Decide which process to load when memory space becomes available
    - Allocate and deallocate memory space as needed
- **Secondary storage**
    - Free space management
    - Storage allocation
    - Disk scheduling
- **File Management**
    - File creation and deletion
    - Directory creation and deletion
    - Support of primitives for manipulating files and directories
    - Mapping files onto secondary storage
    - File backup
- **Networking (Distribution systems)**
    - Computation Speed up
    - Increased data availability
    - Enhanced reliability
- **I/O System Management**
    - A buffer caching system
    - A general device-driver interface
    - Drivers for specific hardware devices
- **Protection System**
    - Protection refers to a a mechanism for controlling access by programs, processes or users system and resources
    - Distinguish between authorized and unauthorized usage
    - specify the controls to be imposed
    - provide a means of enforcement
- **Command-Interpreter System**

Goals of Operating System

- Convenience (User-friendly)
    - User Interface
    - Ease of Use
- Efficiency (Best using all hardware)
    - Resource Utilization
    - Performance Optimization
- Portability
    - Cross-platform Compatibility
    - Hardware Independence
- Reliability
    - System Stability
    - Consistent Performance
- Scalability (Updating)

    - System Upgrades
    - Expandable Capacity
- Robustness (Tackling error)
    - Error Detection
    - Error Recovery

This text representation outlines the main goals of an operating system and their sub-components, as shown in the original diagram.

```
%%{init: {'theme': 'base', 'themeVariables': { 'fontSize': '16px' }}}%%
graph LR
    OS["Operating System Goals"]
    OS --> Convenience["Convenience (User-friendly)"]
    OS --> Efficiency["Efficiency (Best using all hardware)"]
    OS --> Portability["Portability"]
    OS --> Reliability["Reliability"]
    OS --> Scalability["Scalability (Updating)"]
    OS --> Robustness["Robustness (Tackling error)"]

    Convenience --> UI["User Interface"]
    Convenience --> EasyUse["Ease of Use"]

    Efficiency --> ResourceUtil["Resource Utilization"]
    Efficiency --> PerformanceOpt["Performance Optimization"]

    Portability --> CrossPlatform["Cross-platform Compatibility"]
    Portability --> HardwareIndep["Hardware Independence"]

    Reliability --> Stability["System Stability"]
    Reliability --> Consistency["Consistent Performance"]

    Scalability --> SysUpgrades["System Upgrades"]
    Scalability --> ExpandCapacity["Expandable Capacity"]

    Robustness --> ErrorDetection["Error Detection"]
    Robustness --> ErrorRecovery["Error Recovery"]
```

Operating System Services:

- Program Execution
    - Load and execute programs
    - End or abort execution
- I/O Operations
    - Device control
    - Data transfer
- File System
    - File management
    - Directory management
- Communications
    - Inter-Process Communication (IPC)
    - Network communication

- Error Detection
  - Hardware errors
  - Software errors
- Resource Allocation
  - CPU allocation
  - Memory allocation
- Accounting
  - Usage tracking
  - User billing
- Protection
  - Access control
  - Data security

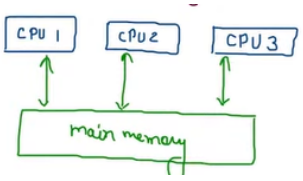
Types of OS

- **Uni-Programming OS**
  - OS allows only one process to reside in main memory (MM)
  - Single process cannot keep CPU and I/O devices busy simultaneously
  - Not a good CPU utilization
- **Multi Programming OS**
  - multiple processes to reside in MM
  - degree of multiprogramming : no. of running process in mm
  - degree of multiprogramming increase, CPU utilization increase but up to a certain limit.
  - Better CPU utilization than uni programming.
  - Concepts
    - All the jobs that enter the system are stored in the job pool. The operating system loads a set of jobs from job pool into main memory and begins execute.
    - During execution, the job may have to wait for some task, such as I/O operation, to complete. In multiprogramming system, the OS simply switches to another job and executes. When the job needs to wait, the CPU is switched to another job and so on.
    - As long as, at least one job needs to execute.
  - Types
    - Preemptive : process can be forcefully taken out of the cpu
    - Non-preemptive: process runs of CPU will to wish
      - Either process terminates
      - or goes for I/O operation
- **Multi-tasking OS/Time-Sharing OS**
  - Extension of multi-programming OS in which processes execute in round robin fashion
  - Fastest switching between multiple jobs to make processing faster
  - Allows multiple users to share computer system simultaneously
  - The users can interact with each job while it is running
- **Multi-User OS**

- allows multiple users to access single system simultaneously

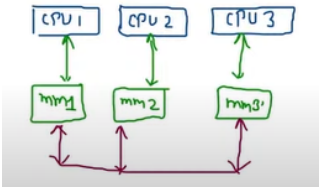
- **Multi-processing OS/Multiprocessor OS**
  - known as parallel OS or tightly coupled OS
  - computer system has multiple CPUs/processors
  - have more than one processor in close communication that sharing the computer bus, the clock and sometimes memory and peripheral devices.
  - It executes multiples job at the same time
  - Not in windows, available on Linux
  - **Categories**
    - **Symmetric multiprocessing system:** each processor runs an **identical copy** of the OS and these copies **communicate** with one another as needed
    - **Asymmetric Multiprocessing system:** a processor is called master processor that controls other processors called slave processor, establishes master-slave relationship, master processor schedules the jobs and manages the memory for entire system
  - Types
    - **Tightly coupled/shared memory :**

      All CPU's sharing single MM



    - **Loosely coupled/distributed system:**

      Each CPU's has its own MM



- **Embedded OS**
  - An OS for embedded computer systems
  - Designed for a specific purpose, to increase functionality and reliability for achieving a specific task
  - User interaction with OS is minimum
- **Distributed OS**
  - the different machines are connected in a network and each machine has its own processor and own local memory
  - OS of all machines work together
  - **Types**
    - Client-server system
    - Peer-to-peer system
- **Desktop System/Personal Computer System**
  - maximizing user convenience and responsiveness

- it is neither multi-user nor multi tasking

- **Real-Time OS**
  - Real time operating system (RTOS) are multi tasking OS, used in environments where a large number of events, mostly external to the computer system, must be accepted and processed in a short time or within certain deadlines
  - OS used for rocket launching
  - Every process has a deadline
  - Types
    - Hard RTOS : Strict about deadlines, example: air bags on cars
    - Soft RTOS: some relaxation in deadline, example : online games
- **Hand-held Device OS**
  - OS used in hand-held devices
  - Android, IOS etc

System Call

- A system call is a way for programs to interact with operating system.
- provides an interface between the process and the OS
- For example, for I/O a process involves a system call telling the operating system to read or write particular area and this request is satisfied by the operating system.

Types of system call

Process Control

- end, abort
- load, execute
- create process, terminate process
- get process attributes, set process attributes
- wait for time
- wait event, signal event
- allocate and free memory

Device Management

- request device, release device
- read, write, reposition
- get/set device attributes
- logically attach or detach devices

File management

- create, delete file
- open, close
- read, write, reposition
- get file attributes, set file attributes

Information maintenance

- get/set time or date
- get/set system data
- get/set process attributes, file attributes, devices attributes

Communications

- create, delete communication connection
- send, receive message
- transfer status information
- attach or detach remote device

Dual Modes of Operation

Used to implement protection

- User mode (mode bit = 1)
- Kernel/System/Supervisor/Privileged Mode (mode bit=0)

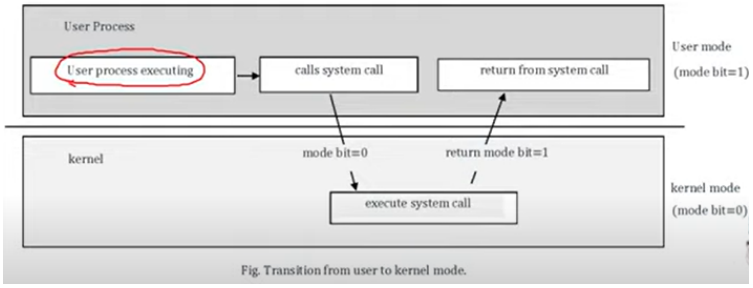


Fig. Transition from user to kernel mode.

Questions

- What are the computer system architecture? Briefly describe multiprocessor system.
- Define Operating system? What are the services of operating system?
- What are the Operating system services? Briefly describe with the figure
- Define Time sharing and Real-Time operating systems.

- Write the fundamental approaches for users to interface with the operating system and explain them briefly.
- What is system call? Mention the name of the system calls involved in Device management.
- What is system call? Mention different types of system call.
- What does the kernel do when a context switch occurs?
- In what ways is the modular kernel approach similar to the layered approach? In what ways does it differ from a layered approach?
- What is meant by context switching? - explain with the proper figure.
- What is the advantage of many-to-many relationships between user threads and kernel threads over other relationship models?

Process

- **What are the differences between a process and a program? Explain.**

A process is **sequential program in execution**. A process defines the **fundamental unit of computation** for the computer. Components of process are :

1. Object Program
2. Data
3. Resources
4. Status of the process execution

Object program i.e. code to be executed. Data is used for executing the program. While executing the program, it may require some resources. Last component is used for verifying the status of the process execution. A process can run to completion only when all requested resources have been allocated to the process.

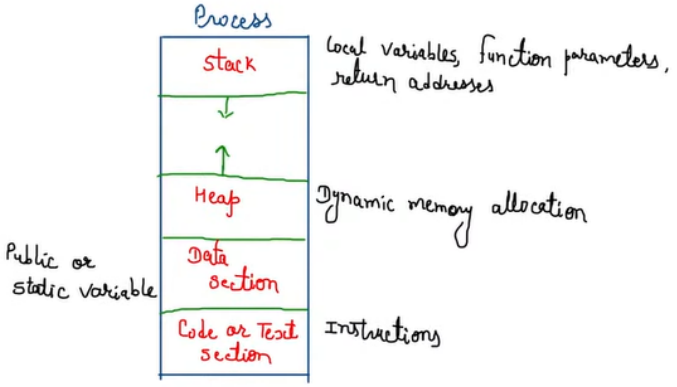
Process = Program (Code) + Running environment (operands and other information)

Aspect	Program	Process
Nature	Static object	Dynamic object
Storage Location	Resides in secondary storage (e.g., hard drive)	Resides in main memory (RAM)
Execution	Inactive until executed	Active, executing at any given time
Lifetime	Span time is unlimited	Span time is limited (ends when execution completes)

Aspect	Program	Process
Entity Type	Passive entity (just a set of instructions)	Active entity (involves execution)
Representation	Expressed in a programming language	Expressed in assembly or machine language (for execution)

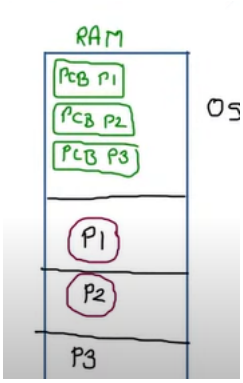
Process as a Data Structure

- **Definition :** code or instruction or program
- **Representation/Implementation :**
  - How process stored in memory



- **Operations**
  - Create (Resource Allocation)
  - Schedule, run
  - wait/block
  - Suspend, resume
  - Terminate (Resource Deallocation)
- **Attributes**
  - PID : process id, uniquely identify each process
  - PC : Program counter
  - GPR : General process register
  - Lists of Device
  - Type
  - Size
  - Memory Limits
  - Priority
  - State
  - List of files

  (These attributes are maintained by a DS called PCB)



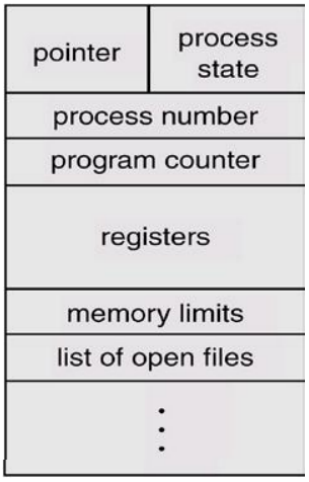
PCB

- **Draw the block diagram of the Process Control Block (PCB).**
- **Mention five components of a process's PCB.**

A Process Control Block (PCB) is a data structure used by the operating system to store information about a specific process.

- Each process **contains** the process control block (PCB) or process descriptor.
- PCB is the **data structure** used by the OS.
- OS groups all information that needs about a particular process.

Components of PCB



- Pointer: Pointer points to another PCB to maintain the scheduling list.
- Process state
- Program Counter : The address of the next instruction
- CPU Registers : Registers like AC, GPR, IR etc.
- CPU Scheduling information: process priority, pointer to scheduling queues and any other scheduling parameters.
- Memory-management information: base and limit register, page or segments table etc.
- Accounting information: amount of CPU & real time uses, time limits, account numbers, job or process numbers.
- I/O Status information: List of I/O Devices allocated to the process, a list of open files etc.

Context

- **What is context switching?**
- **Explain context switching with an appropriate figure.**

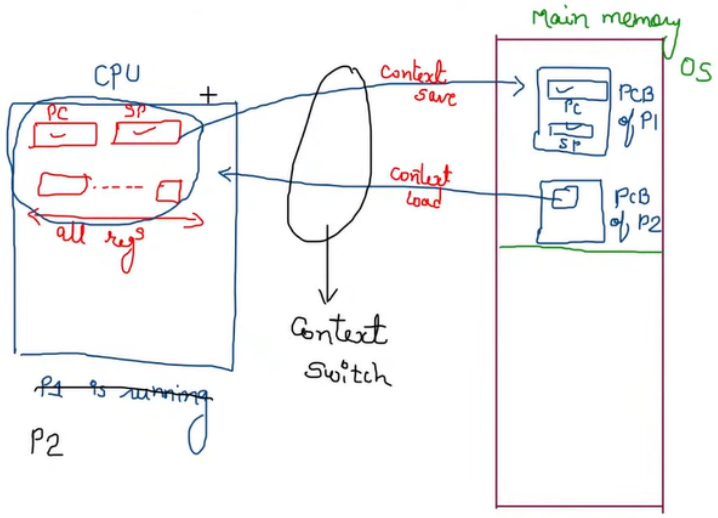
The content of PCB in a process are collectively known as “Context” of that process

- **Context switching :**

  Context Switching is the process by which an operating system saves the state of a currently running process or thread and loads the state of a different process or thread.

  - Stop a running process and start another
  - Context switch is done by dispatchers
  - Context switch time is pure overhead.

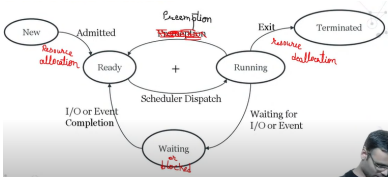
- When the scheduler switches the CPU from executing one process to executing another, the context switcher saves the content of all processor registers for the process being removed from the CPU in its process being removed from the CPU in its process descriptor
- Work
  - Context Save
  - Context Load



Process States

- **Draw the process state diagram and label its states.**
- **Discuss the various process states.**

When process executes, it changes state. Process state is defined as the current activity of the process.



New: A process that just been created.

Ready: A process said to be ready if it needs a CPU to execute. A ready process is runnable but temporarily stopped running to let another process run.

Running: A process that is currently being executed/ a process which has the CPU to run. A running process possesses all the resources needed for its execution, including the processor.

Terminated : The process has finished execution.

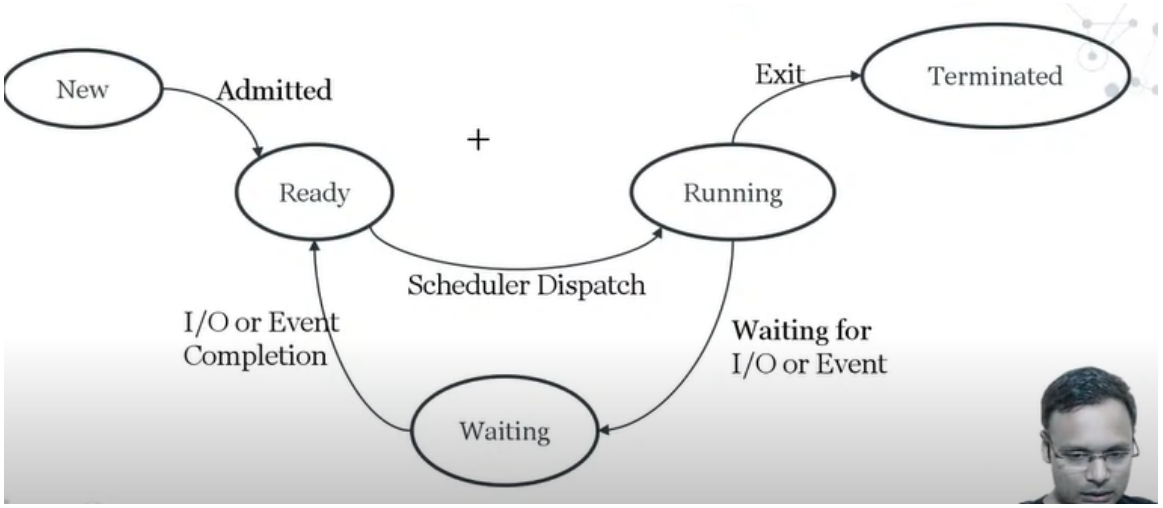
Blocked/Waiting : A process which is waiting for some event to happen such that as an I/O completion before it can proceed.

Transition

- New to ready : when process is admitted by OS, done by OS
- Ready to running: when process is dispatched to CPU, done by OS

- Running to Terminated: When a process is completed, done by Process
- Running to blocked: When a process goes for a IO or event, done by process
- Running to read : When a process is preempted, done by OS
- Blocked to ready : When a process completes IO or event, done by OS

Process states : Non-preemptive



CPU vs IO Bound Process

- CPU Bound: If the process is intensive in terms of CPU operations, spends more time doing computations, few very long CPU bursts
- IO Bound:: If the process is intensive in terms of IO Operations, spends more time doing I/O than computations ,many short CPU bursts

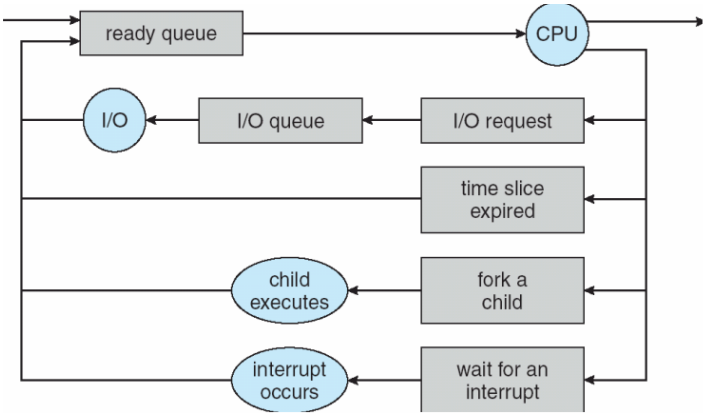
Process Scheduling

The scheduling mechanism is the part of the process manager that handles the **removal of the running process** from the CPU and the **selection of another process basis** of particular strategy.

- Need for better resource utilization

Process Queues

- Job Queue : All process in the system which are entered to the system as new processes,
- Ready queue: Processes that are residing in MM and are ready and waiting to execute by CPU. This queue is stored as a linked list. Each PCB includes a pointer field that points to the next PCB in ready queue.
- Device queue: process are waiting for a specific i/o device, each device has its own device queue



Schedulers

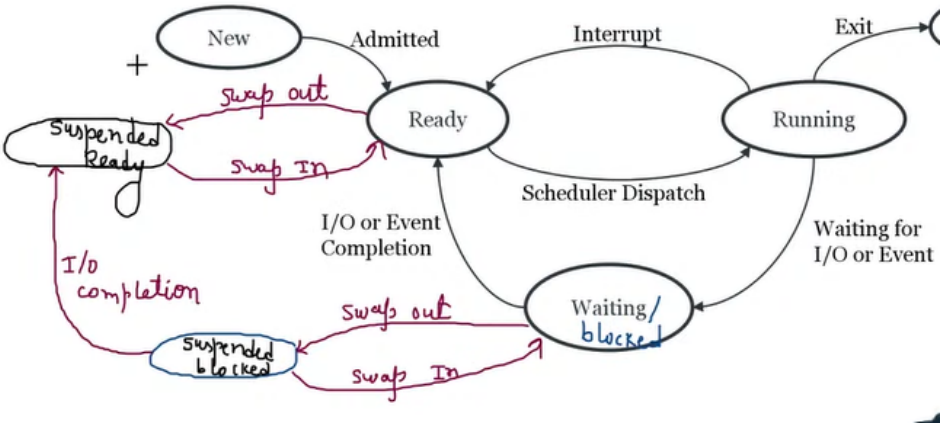
- **What are the schedulers in an operating system?**
- **Differentiate between short-term and long-term schedulers.**

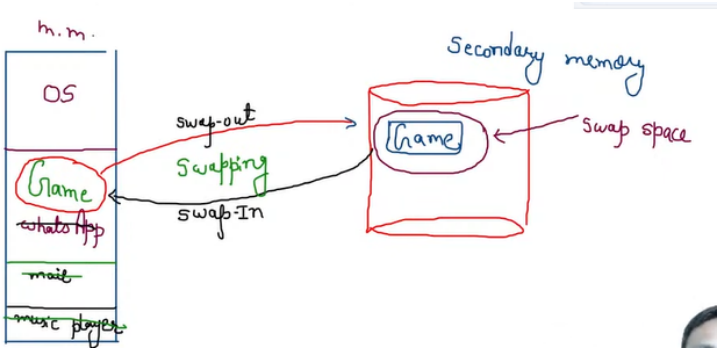
A scheduler is a decision maker that selects the processes from one scheduling queue to another or allocates CPU for execution.

Types of schedulers

Aspect	Long Term Scheduler	Short Term Scheduler	Medium Term Scheduler
Function	Job scheduler	CPU scheduler	Swapping scheduler
Speed	Slower than short-term scheduler	Very fast	Moderate speed, between long and short-term
Control over Multiprogramming	Controls the degree of multiprogramming	Minimal control over multiprogramming	Reduces the degree of multiprogramming
Presence in Time-Sharing Systems	Usually absent or minimal	Minimal role	Commonly used
Selection	Chooses processes from a pool and loads them into memory for execution	Selects processes ready for CPU execution	Reintroduces a process into memory for resumed execution
Process State Transition	Transitions processes from New to Ready state	Transitions processes from Ready to Running state	Does not directly involve state transitions
Process Mix	Selects a balanced mix of I/O-bound and CPU-bound processes	Frequently selects a process for the CPU	-

- Long-term-scheduler (job) :
  - selects processes from discs and loads them into Main memory
  - new state to ready state
  - resource allocation happens
- Short term scheduler (CPU) :
  - selects one of all ready processes to run on CPU
- Mid-term Scheduler (Medium-term)
  - Does Swapping
    - Swap out : suspended state





Operations

- **What are the reasons that a parent process may terminate the execution of one of its child processes?**

- **UNIX Examples**
- `fork`: Creates a new process.
- `exec`: Replaces process memory with a new program after `fork`.

- **Following are the resources for terminating the child process by parent process.**

1. The task given to the child is no longer required.
2. Child has exceeded its usage of some of the resources that it has been allocated.
3. Operating system does not allow a child to continue if its parent terminates.

CPU Scheduling

- **Explain the difference between preemptive and non-preemptive scheduling.**
- **Describe how to address the starvation problem in a priority scheduling algorithm.**
- **Explain the effect of increasing or decreasing the time quantum to an arbitrary small number for the Round-Robin scheduling algorithm with a suitable example**
- **What are the scheduling criteria?**
- **Draw Gantt charts illustrating FCFS, preemptive SJF, non-preemptive priority, and Round-Robin scheduling.**

CPU scheduling refers to **the switching between processes that are being executed**. It forms the basis of multiprogrammed systems.

Scheduling Criteria

- CPU Utilization (max) : Keeping the CPU busy
- Throughput (max) : no. processes that completes their execution per time unit
- Turnaround time (min) : amount of time to execute a process
- Waiting time (min): amount of time process has been waiting in the ready queue
- Response time:(min) amount of time it takes from when a request was submitted until the first response produced

[Between the parenthesis : Optimization Criteria]

CPU Types

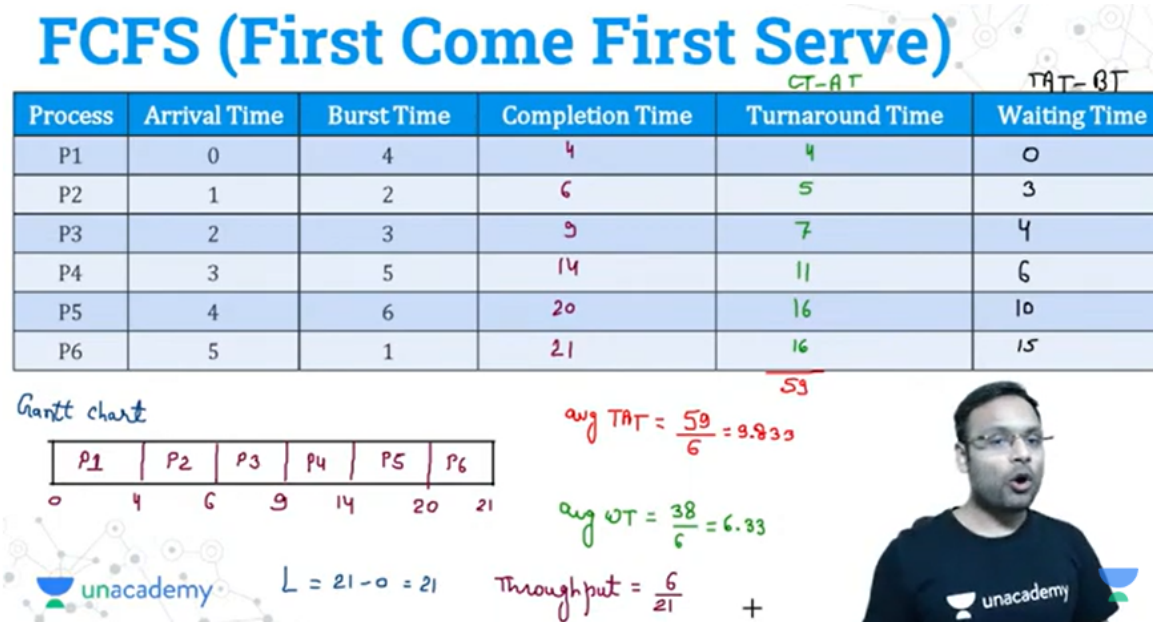
- Preemptive
- Non-preemptive

Aspect	Preemptive Scheduling	Non-Preemptive Scheduling
Definition	The CPU can be taken away from a running process before it finishes its execution.	Once a process starts executing, it runs to completion or until it voluntarily releases the CPU.
Control	The operating system has control and can interrupt processes.	The process controls when it gives up the CPU.
Interruptions	A process can be interrupted by the scheduler to give CPU time to another process.	A process cannot be interrupted; it must finish its execution.
Context Switching	Frequent context switching due to preemption.	Less frequent context switching as processes are not preempted.
Example	Round-robin, Shortest Remaining Time First (SRTF).	First-Come, First-Served (FCFS), Priority Scheduling (non-preemptive).
Advantages	Better responsiveness and fairness in handling processes.	Simpler to implement, no need for frequent context switching.
Disadvantages	Increased overhead due to context switching.	Can lead to poor response times, especially with long-running processes.

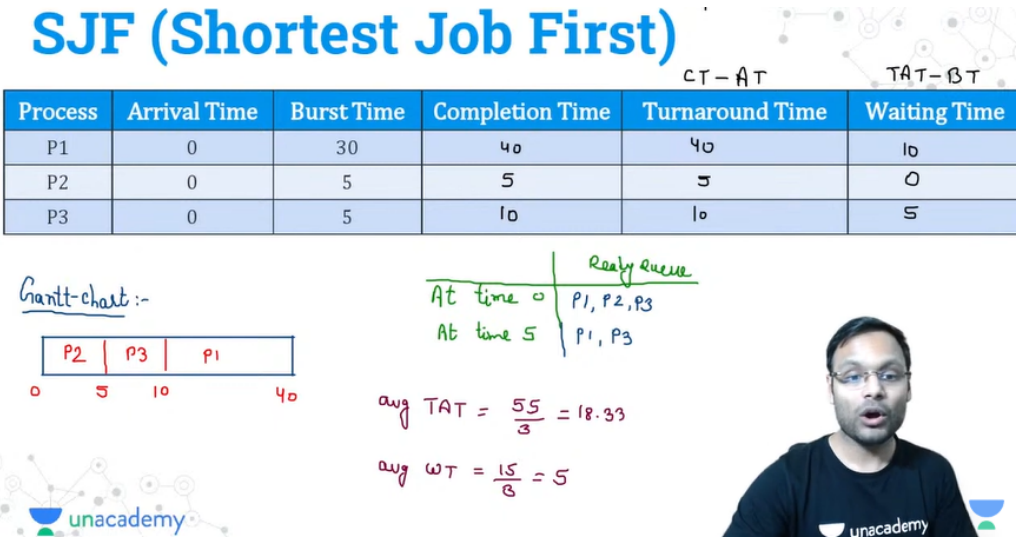
[Note : Every process has no any I/O operation. (Assumption)]

Algorithms

- FCFS : First Come First Serve
  - Criteria : Arrival Time (AT)
    - Tie-breaker : Smaller process id first
  - Type: Non-preemptive
  - Gannt Chart:
    - From when to when
    - always start from 0
  - **Disadvantages**
    - **Convoy Effect :** If a large process is scheduled first than it slows down system's performance.

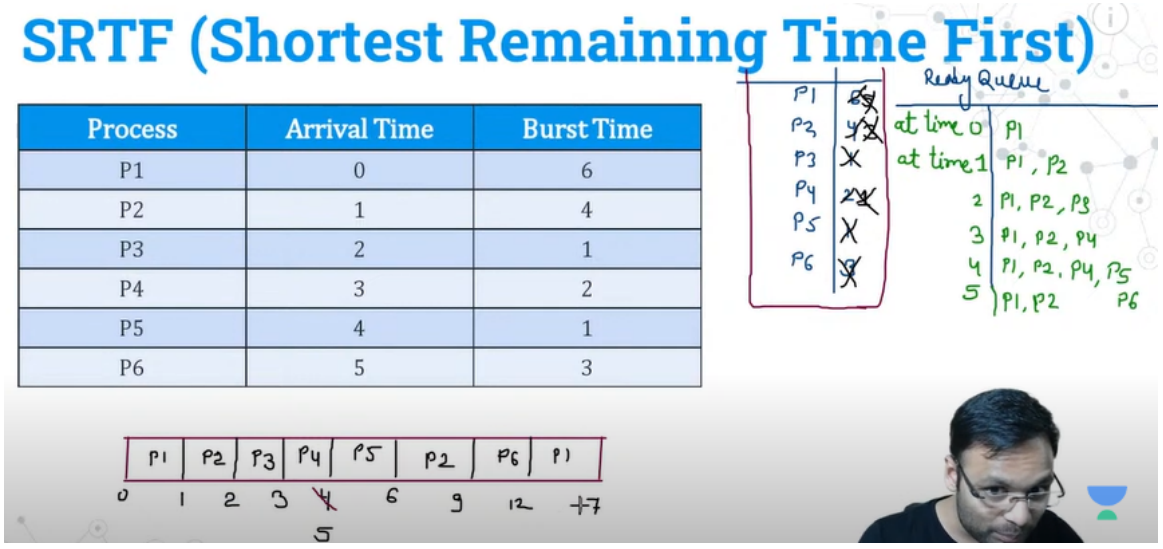


- **SJF (Shortest Job First)**
  - **Criteria :** Smallest Burst Time process first
    - **Tie-breaker:** FCFS (Arrival Time)
  - **Type:** Non-preemptive /**preemptive**
  - **Advantages**
    - Less response time
  - **Disadvantages**
    - Not practical
    - Starvation : indefinite waiting
    - No fairness



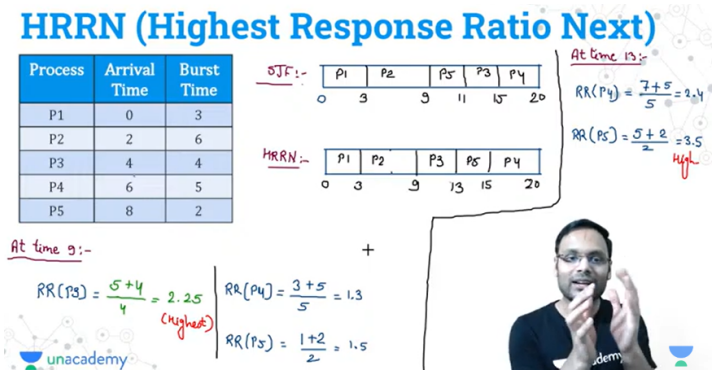
- **SRTF (Shortest Remaining Time First)**
  - **Criteria :** Burst Time

    - Tie-breaker : FCFS
  - **Type:** Preempted
  - **Tricks**
    - Write the process, BT by side
  - **Disadvantages**
    - Not practical
    - Starvation : indefinite waiting
    - No fairness



- **HRRN : Highest Response Ratio Next**
  - **Objective:** Not only favors short jobs but decreases the WT of longer jobs
  - **Criteria:** Response Ration (High to low)
    - Tie-braker: BT
  - Type: Non-preemptive

$RR = \frac{W+S}{S}$, W = wait time, S=Service/Burst Time

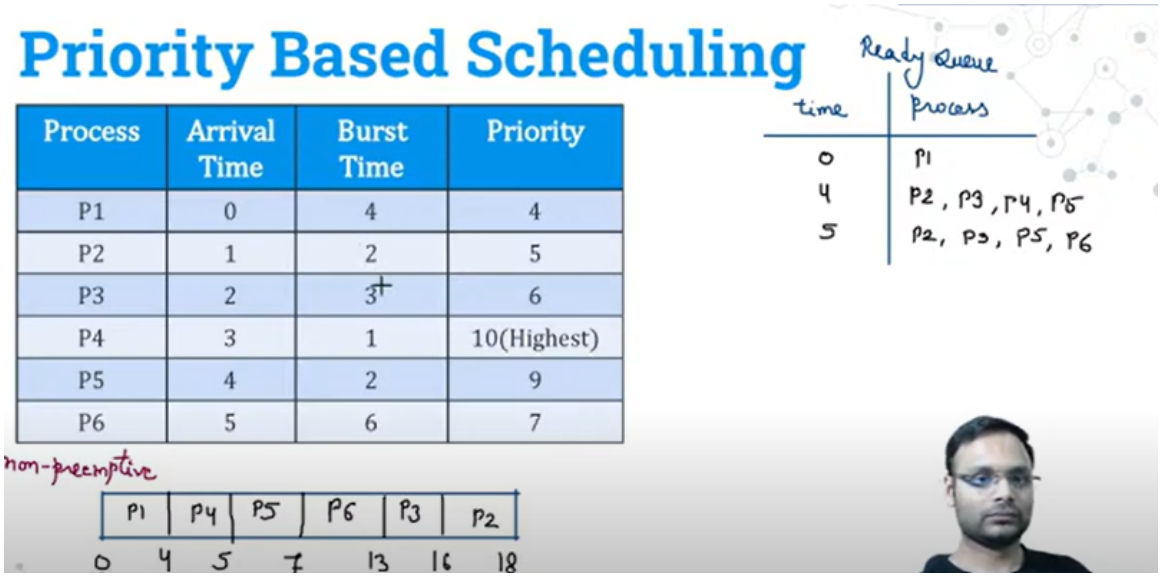


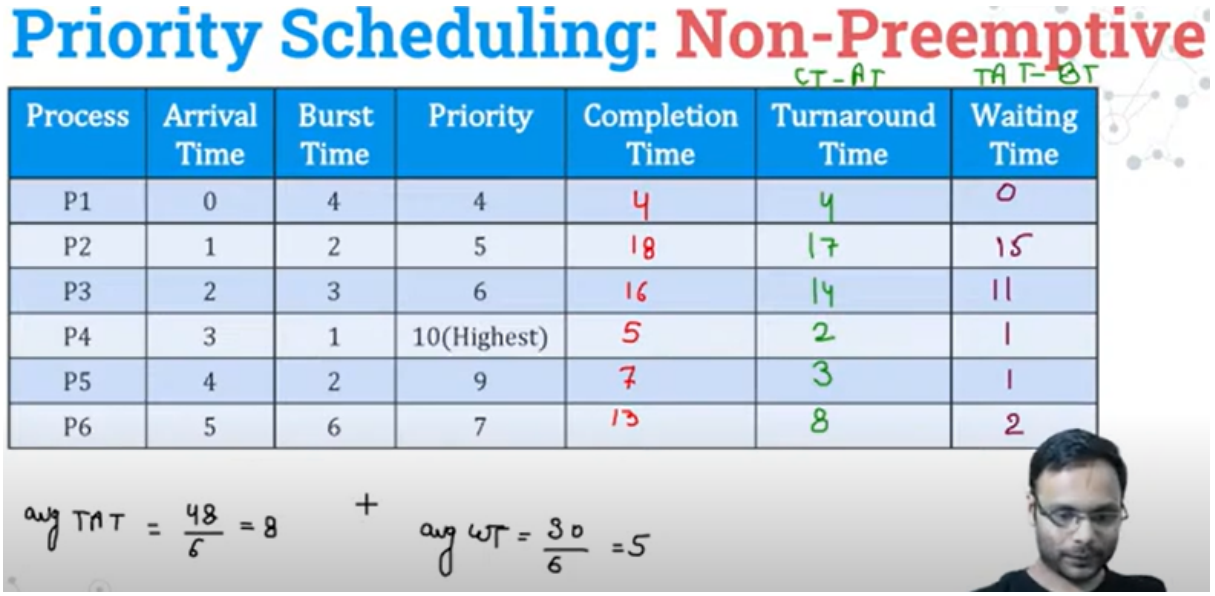
- **Priority Based Scheduling**
  - **Criteria :** Priority
    - **Tie- breaker:** FCFS
  - **Type:** Non-preemptive, Preemptive both
  - **Disadvantages**
    - Starvation: If higher priority processes keep arriving then low priority processes may wait untill indefinite time

Priority

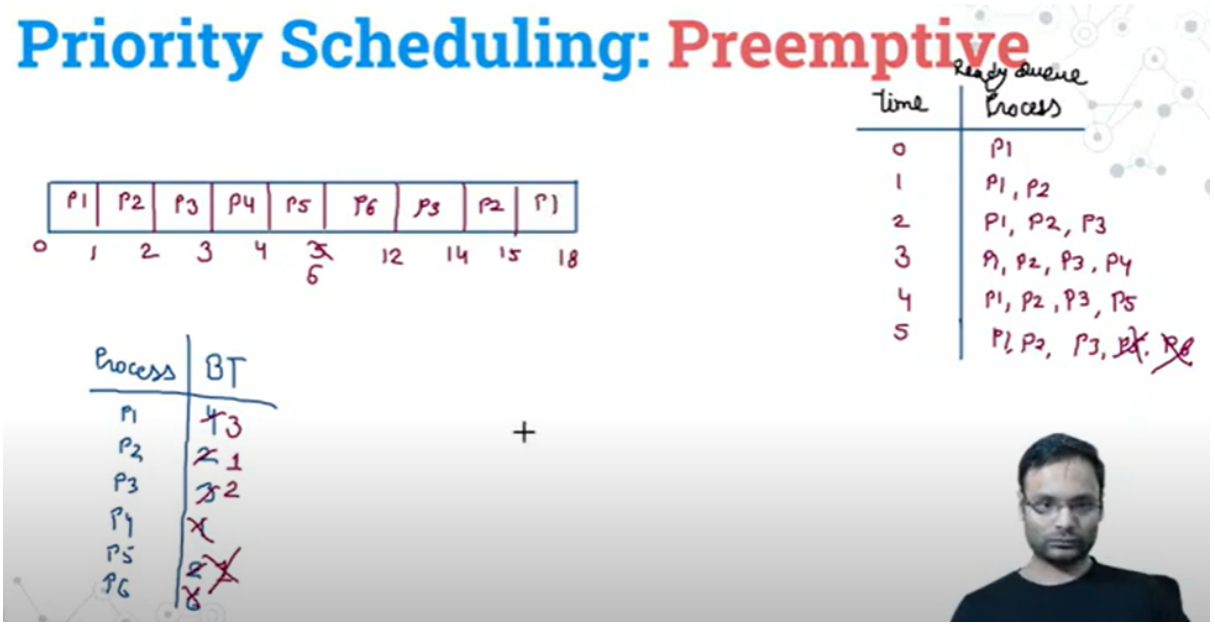
- **Type**
  - Static : Fixed
  - Dynamic: may increase or decrease

Non preemptive





Preemptive



- **Round-Robin**
  - **Objective:**
    - provides instructiveness
    - fairness
  - **Criteria :** AT + Q, Q : Time Quantum
    - **Tie-breaker:** Process ID
  - **Type:** Preemptive

Memory Management

- **What is the fragmentation problem in memory management? Define internal and external fragmentation.**
- **Describe first-fit, best-fit, and worst-fit strategies with examples.**
- **Explain swapping and describe the standard swapping process.**
- **What is paging? Explain paging with an example.**
- **Discuss the paging method and show with an example**
- **Define paging method and demonstrate with an example.**

Memory Management is the process of efficiently handling the computer's memory, which includes the allocation, deallocation and organization of memory during execution.

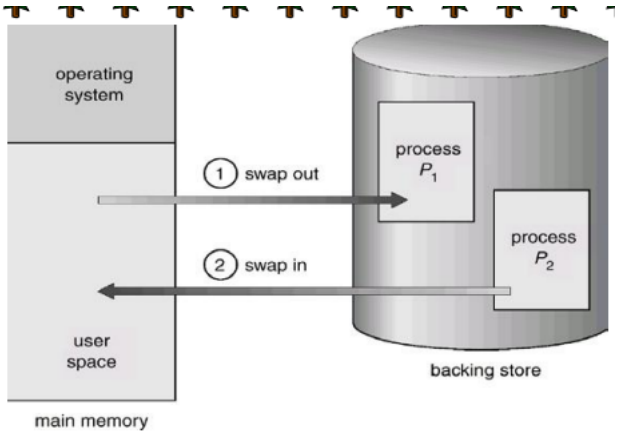
- It increases utilizing of CPU by increasing degree of multiprogramming
- Protect process memory from unauthorized access and prevent process interfering with each other.
- Allocation and deallocation. process isolation, efficient memory use, memory sharing, memory hierarchy management c

Binding of Instruction Data to Memory

1. **Compiler Time :** If it is known at compiler time where the process will reside in memory, then absolute code can be generated**.**
2. **Load time :** It is the time taken to link all related program file and load into the main memory. It must generate relocatable code if memory location is not known at compiler time.
3. **Execution:** Time taken to execute the program by processor. If the process can be moved during its execution from one memory segment to another, then binding must be delayed until run time.

Technique to optimize use and increase system efficiency

1. Dynamic Loading
    a. a programs code and data are loaded into memory only when they are needed during execution
    b. improve memory utilization, save memory, No OS supper required
2. Dynamic Linking
    a. postpones the linking of libraries to a program until runtime, instead of linking them at compiler time
    b. Small piece of code → stub, used to find library in memory
    c. Save memory by sharing common libraries, Need support from OS
3. Overlays
    a. process is larger than available memory → the necessary parts (overlay) are loaded into memory while other parts are swapped in/out
    b. designed and managed by programmer. no need of OS support
4. **Swapping**
    a. temporality moves a process (or a part) out of memory (RAM) to secondary storage to free up memory for others
    b. when the process is needed again it is swapped back into memory
    c. Process : Program Execution → Memory Full or Process Waiting → Swapping out (Roll out) a selected process MM to secondary (swap space) → Swapping In the process needs CPU or ready for execution, secondary to MM → Process resumption, process resume its execution from where it left of

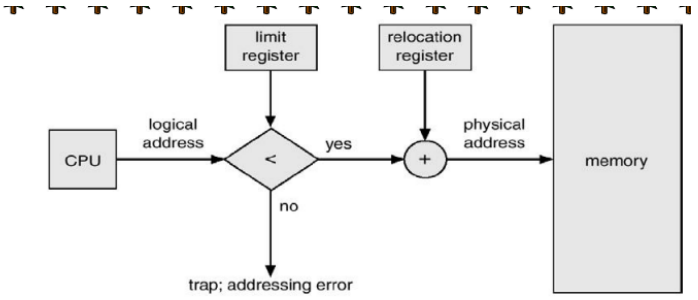


(Schematic View of Swapping)

Basis of comparison	Logical Address	Physical Address
Basic	Virtually generated by CPU	Exists within the MMU
Visibility	Viewable.	Not viewable
Address Space	logical address space	physical address space
Access	Used to access physical address	Not directly accessed
Generation	Generated by the central processing unit.	Computed by the memory management unit.
Variation	variable	constant

Memory Management Technique

1. **Contiguous**
   - each process in the system is assigned in a single continuous of memory during its execution



(Hardware support for relocation and limit registers)

- **Types**
  - Single Partition Scheme
  - Multiple Partitions Scheme : main memory is divided into a number of fixed-sized partitions where each partition should contain only one process

Feature	Multiple Fixed Partitions	Multiple Variable Partitions
Partition Size	Fixed partition sizes defined during system generation.	Variable partition sizes based on process requirements.
Memory Allocation	Any process can fit into an available partition if its size ≤ partition size.	Processes are allocated exactly the memory they require.
Efficiency of Memory Usage	Inefficient due to internal fragmentation.	More efficient, with no internal fragmentation.
Max Active Processes	Fixed, limited by the number of partitions.	Flexible, limited by the available memory.
Swapping	Processes can be swapped in and out of partitions.	Compaction is needed to manage memory, especially due to external fragmentation.
Operating System Overhead	Low overhead.	Higher overhead due to memory compaction.
External Fragmentation	Does not occur, as memory is statically partitioned.	Can occur, requiring compaction to manage free memory.
Implementation Complexity	Simple and easy to implement.	More complex due to dynamic memory allocation and compaction.

- **Fragmentation:** a situation in memory management when memory space is used inefficiently, leading to wasted or unusable memory

  Types

Feature	Internal Fragmentation	External Fragmentation
Definition	Wasted memory within a partition that is not utilized by a process.	Wasted memory outside the allocated memory regions, leaving gaps between blocks.
Cause	Occurs when a process is allocated more memory than it needs, causing unused space within the partition.	Occurs when free memory is scattered in small blocks, but no contiguous space is large enough for a new process.
Location of Wasted Space	**Inside** the allocated partition.	**Between** allocated memory blocks.
Memory Allocation	Memory is allocated in fixed-size blocks, leading to unused space if the process is smaller than the block size.	Memory is allocated dynamically, and free space is fragmented.
Impact on Performance	Leads to inefficient use of memory, but doesn't require extra work to manage.	Causes difficulty in allocating memory, potentially leading to delays or inability to allocate memory even if total free space is sufficient.
Management Complexity	**Simple** to manage; internal fragmentation is a result of fixed-size partitions.	More **complex** to manage; requires techniques like **compaction** to reduce fragmentation.

Feature	Internal Fragmentation	External Fragmentation
Solution	Cannot be fully avoided, but can be minimized by using smaller partition sizes.	Can be resolved by techniques such as memory compaction or using more efficient allocation strategies.
Example	A partition of 1 KB allocated to a process requiring 900 bytes results in 100 bytes of internal fragmentation.	Free memory blocks of 100 KB, 50 KB, and 25 KB scattered across memory might prevent allocation of a 120 KB process.
Advantages	- Simple to manage. - Low overhead in allocation. - Memory is allocated quickly and predictably.	- Flexible memory usage, as memory can be allocated dynamically. - No need to allocate more space than required by the process.
Disadvantages	- Inefficient use of memory due to unused space within partitions. -Can lead to significant wastage if partitions are large and processes are smaller.	- Inefficient memory use due to scattered free space. - May cause memory allocation failures even when there is sufficient total free memory. - Requires memory compaction, which adds overhead.

- **Partition Selection Policy :** request memory → the OS must decide which free memory partition to allocate → guided by Partition Selection Policy
  - **First Fit:** memory manager scans the list of free block from beginning → allocate the first block that is big enough
  - **Next Fit :** starts from the last block allocated → the current process allocated to the next block which is big enough
  - **Best Fit :** searches the entire list of blocks → find the smallest block which is big enough for the process
  - **Worst fit:** searches the entire list of blocks→ find the largest block that is big enough size than the size of process

2. **Non contiguous:** it is allowed to store the processes in noncontiguous memory locations.

   Types

   i. **Paging**

   - **Frames :** Main memory is divided into a number of equal size blocks
   - **Pages :** Each process divided into number of equal size block of the same length as frames
   - From logical to physical addresses
     - CPU generated/logical address : page number and page offset
     - Page table contains the base address of each page in physical memory
     - Logical address space 2^m and page size 2^n addressing units, then (m-n) bits = page number and n bits = page offset.

     Example

     Logical memory = 4B = 2^2, $m = 2$
     Page size = $2^1, n = 1$

     Logical address = 2^2, 2 bits

     Page number bit = $m - n = 2 - 1 = 1$

     Offset = $n = 1$

     Primary memory = 8 B = $2^3, m = 3$

     Frame size = $2^1, n = 1$

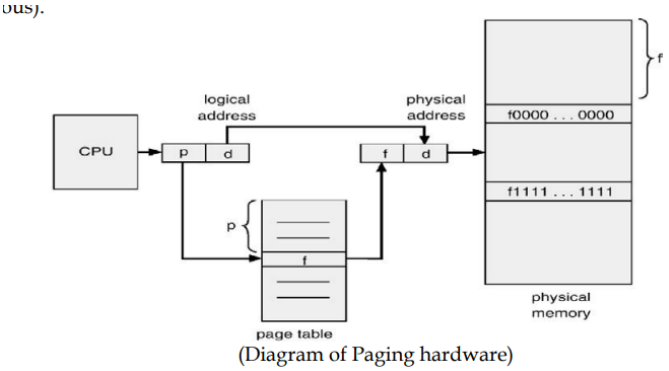
     Physical address = $2^3 = 3$ bits

     Frame number bit = $m - n = 2$

Offset = n = 1

- Find physical address using decimal value

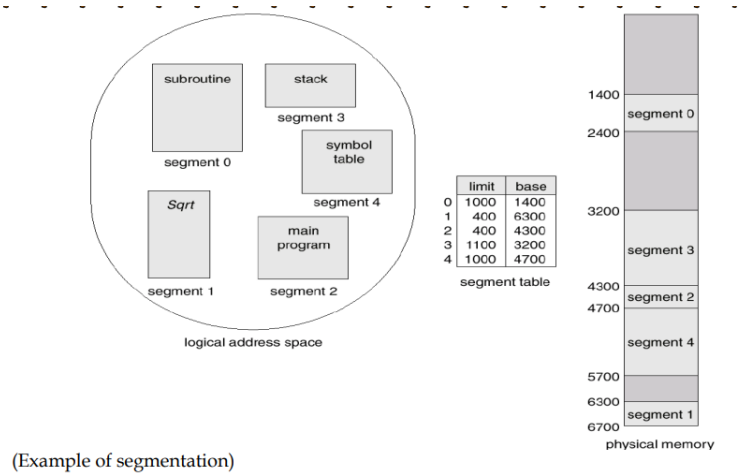
  Physical address = (frame number * frame size) + offset



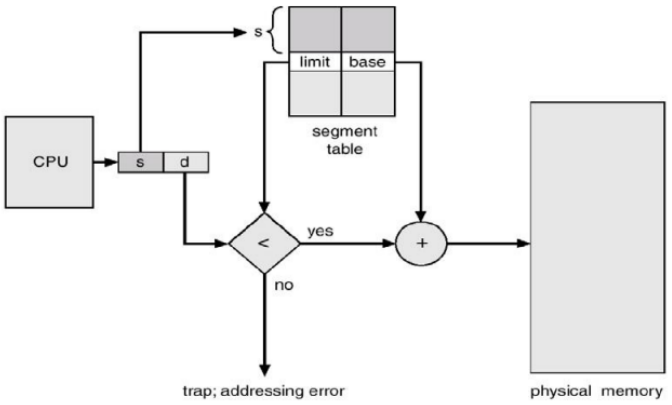
(Diagram of Paging hardware)

ii. **Segmentation :** Segmentation is a memory management technique that divides a process memory into variable sized blocks called segments, based on the logical division of program. Segments are logical unit such as function, array, method etc.

- base : contains the starting physical address
- limit : specifies the length of the segment
- STBR : segment table base register , points to the segment table's location in memory
- STLR: segment table length register, indicates number of segments used by a program
- A logical-address space is a collection of segments.. Logical address consists of a two tuple: <segment-number, offset>
- The segment number is used as an index into the segment table. The offset d of the logical address must be between 0 and the segment limit



(Example of segmentation)



Feature	Paging	Segmentation
Definition	A memory management technique that divides the process into fixed-sized blocks, called pages, which are mapped to physical memory frames.	A memory management technique that divides the process into variable-sized segments, each representing logical units of the program.
Division of Program	Divides the program into fixed-sized pages.	Divides the program into variable-sized segments.
Responsibility	Managed by the Operating System.	Managed by the Compiler.
Size Determination	Page size is fixed and determined by hardware.	Segment size varies and is determined by the user.
Speed	Generally faster than segmentation.	Generally slower than paging.
Fragmentation Type	Internal Fragmentation.	External Fragmentation.
Mapping Table	Uses a **Page Table** to map logical pages to physical memory frames.	Uses a **Segment Table** with base and limit addresses for each segment.

File System

- **What are the file access methods? Briefly describe them.**
- **Explain the difference between sequential and direct file access methods.**
- **Discuss file sharing methods.**
- **In Unix, Linux, and Windows file systems, describe the purpose of multiple timestamps associated with files.**

A file system is a **method and data structure** that an operating system uses to manage, organize files on storage devices.

File system consists of two parts

- collection of files
- a directory structure

- A file is a **collection of related information** that is recorded on secondary storage. A file is a collection of similar record.
- Common terms related to file : field (basic element of data), record (collection of related fields), file (collection of similar records), database(collection of related data)
- File attributes : name, identifier , type, location, size, protection, time, date and user identification
- File operation: read, write, create, reposition, delete, truncating
- **Access Method:** File access method defines the way processes read and write files.

1. **Sequential Access Method:** Simple method, the information in a file is accessed sequentially one record after another
    a. a process could read all the records in a file in order, starting at the beginning. It cannot skip any records and cannot read out of order
    b. Batch Application uses. Sequential file organization easily stored on tape and hard disk.
    c. **Disadvantages:** poor performance, more efficient search technique is required

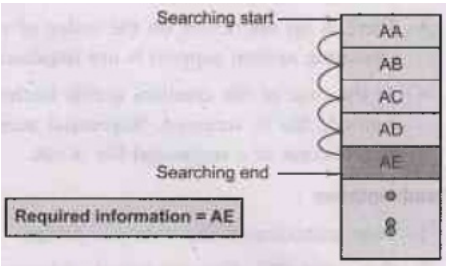


Fig. 6.2.1 Sequential file access

2. **Direct Access method:** Random Access Method, allows a user to position the read/write mark before reading or writing.
    a. It provides accessing the records directly. It is based on hard disk that is a direct access device. It allows random access of any file block.
    b. Each records has its own address on the file with by help of which it can be directly accessed for reading or writing. This feature is used by editors.
    c. There is no restriction on the order of reading or writing for a direct access file. OS support is not needed.
    d. **Disadvantages:** Poor utilization of i/o device, consumes CPU times for address calculation
3. **Index Sequential Access (Extra)**
    a. it is a combination of direct and sequential access method.

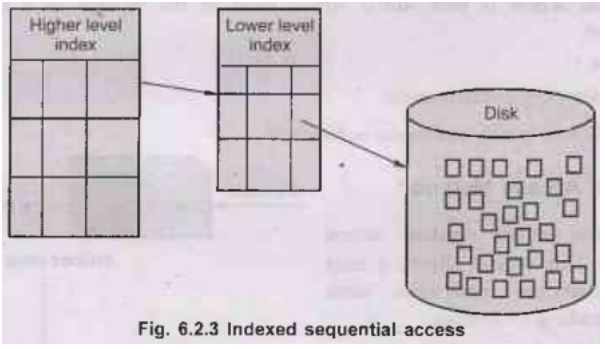
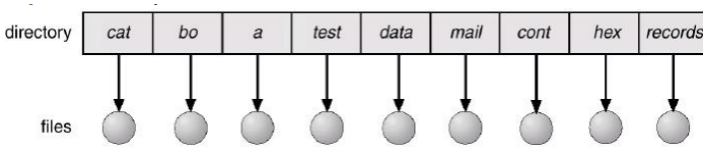


Fig. 6.2.3 Indexed sequential access

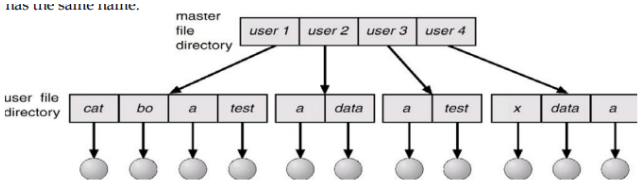
Aspect	Sequential Access	Direct Access
Access Pattern	Data is accessed in a fixed, linear sequence.	Data can be accessed in any order.
Navigation	Step-by-step; must go through preceding data.	Can jump directly to any data block.
Efficiency	Efficient for linear access (e.g., reading from start to end).	Efficient for quick access to specific data points.
Complexity	Simple to implement.	More complex to implement.
Best Use Case	Text files, logs, sequential data processing.	Databases, index files, applications needing random access.
Advantage	Minimal overhead, straightforward access.	Fast access to specific data.
Disadvantage	Inefficient for random access.	Less efficient for reading large files sequentially.

- **Directory:** A directory is an object that contains the names of file system objects.

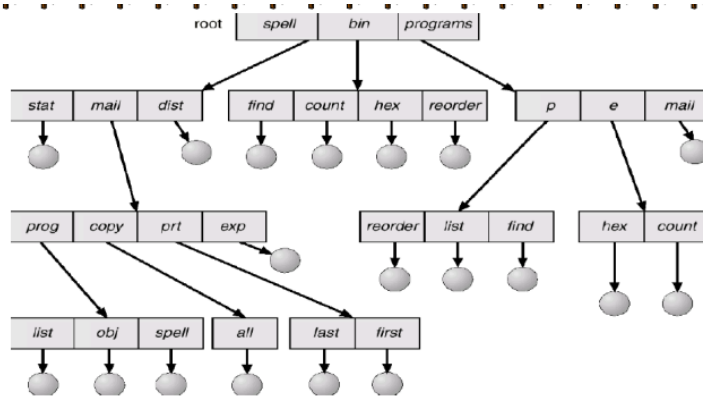
- **Single-Level Directory:** simplest, files contained in the same directory, easy to support and understand, limitations when number of files increases, files must have unique name



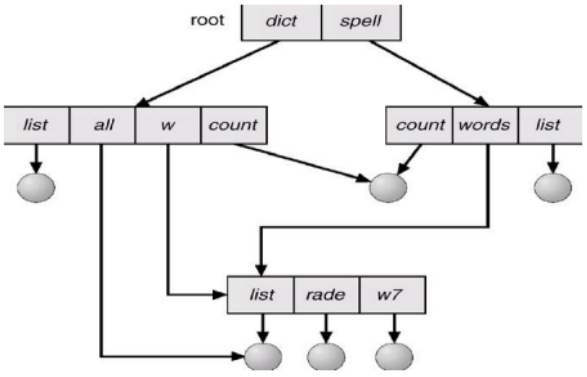
- **Two-level directory:** each user has own user file directory (UFD), each UFD has similar structure, when a user job starts, master file directory is searched, MFD is indexed by username
  - when user refers to a particular file, only his own UFD is searched
  - to create, delete a file for a user, the OS confines its search to the local UFD



- **Tree-structured directories:** powerful and flexible approach to organize files in hierarchical , there is a master directory which has under it a number of user directories, each of users may have sub directories



- **Acyclic-graph directories :** shares subdirectories and subordinates files, same file or subdirectory may be in two different directories,
  - a shared file is not the same as two copies of the file, with two copies each programmer can view the copy rather than the original.. but if one changes, it won't change another in the others copy



File Allocation: the strategies employed by computer OS for the efficient distributing of storage space on disks or others

Aspect	Contiguous File Allocation	Linked File Allocation	Indexed File Allocation
Storage Pattern	Files stored in a single, continuous block	Files stored in non-contiguous blocks with pointers	Files stored in non-contiguous blocks with an index block
Access Speed	Fast access (sequential)	Slower access (following pointers)	Fast access (uses index block)
Fragmentation	Prone to fragmentation	Low fragmentation	Low fragmentation
File Size Flexibility	Limited by contiguous free space	Flexible, any size	Flexible, any size
Disk Space Efficiency	Efficient for continuous storage	Less efficient due to pointer storage	Slightly less efficient due to index block
Risk of Data Loss	Low, as entire file is in one block	Higher, as broken pointers can lose access to data	Low, as the index block can be duplicated
Best for	Large files with sequential access needs (e.g., video files)	Files of varying sizes in a fragmented disk space	Files needing random access or large files with redundancy needs

Thread

- Component of process or lightweight process
- provide a way to improve application performance through parallelism

Aspect	User-Level Threads	Kernel-Level Threads
Definition	Threads managed at the user level without kernel intervention	Threads managed directly by the operating system kernel
Creation & Management Speed	Faster to create and manage	Slower to create and manage
Implementation	Implemented by a thread library at the user level	Directly supported by the operating system
Operating System Dependency	Can run on any operating system	Specific to the operating system
Support Level	Support is provided at the user level, known as user-level threads	Support is provided by the kernel, known as kernel-level threads
Multiprocessing Capability	Limited multiprocessing due to lack of direct kernel support	Kernel routines can be multithreaded, enabling better multiprocessing

Aspect	Process	Thread
Definition	Called a "heavyweight process" that operates independently	Called a "lightweight process" that operates within a process
Switching Requirement	Process switching requires interaction with the operating system	Thread switching doesn’t need an OS call or kernel interrupt
Memory & Resource Allocation	Each process has its own memory and file resources	Threads share the same memory and file resources
Blocking Behavior	If one process is blocked, no other process can execute until it’s unblocked	If one thread is blocked, other threads in the same process can still execute
Resource Usage	Multiple processes use more resources compared to threads	Multiple threads use fewer resources than processes
Interaction & Independence	Processes operate independently from each other	Threads can access and modify each other's stacks within the same process