# Operating System Notes

## By Garvit Singh

### Syllabus

#### Introduction

- Operating System Structure
- Types of Operating Systems
- Operating System Operations
- Process Management
- Memory Management
- Storage Management
- Protection and Security
- Special Purpose

## System Structure

- Operating System Services
- User Operating System Interfaces
- System Calls
- Types of System Calls
- System Programs Operating System Structure
- Virtual Machines
- System Boot



## Process

- Process Concept
- Process Scheduling Operations on Processes
- Inter-Process Communication
- Unix Pipes

## Multithreaded Programming

- Overview
- Multithreaded Models
- Thread Libraries
- Programs Using Pthreads

**Process Scheduling**

- Basic Concepts
- Scheduling Criteria

**Process Synchronization**

- Background
- Critical Section Problem Peterson’s Solution
- Synchronization Hardware
- Semaphores
- Classical Problems of Synchronization
- Programs Using Pthreads

## Deadlocks

- System Model
- Deadlock Characterization Methods for Handling Deadlocks
- Deadlock Prevention
- Deadlock Avoidance
- Deadlock Detection
- Recovery from Deadlock

## Memory Management

- Background - Address Binding, Logical vs Physical Address Space, Dynamic Loading, Dynamic Linking And Shared Libraries, Overlays
- Swapping
- Contiguous Memory Allocation
- PAGING

- Structure of Page Table Segmentation
- Demand Paging
- Page Replacement Policies Allocation of Frames
- Thrashing

**File System Interface and Implementation**

- File Concept
- Access Methods
- Directory and Disk Structure
- File System Mounting
- File System Structure
- Space Allocation Methods for Files - Contiguous, Linked, Indexed
- Free Space Management - Bit Vector, Linked List, Grouping, Counting

**Disk Management**

- Disk Scheduling Algorithms
- Disk Management
- Swap Space Management

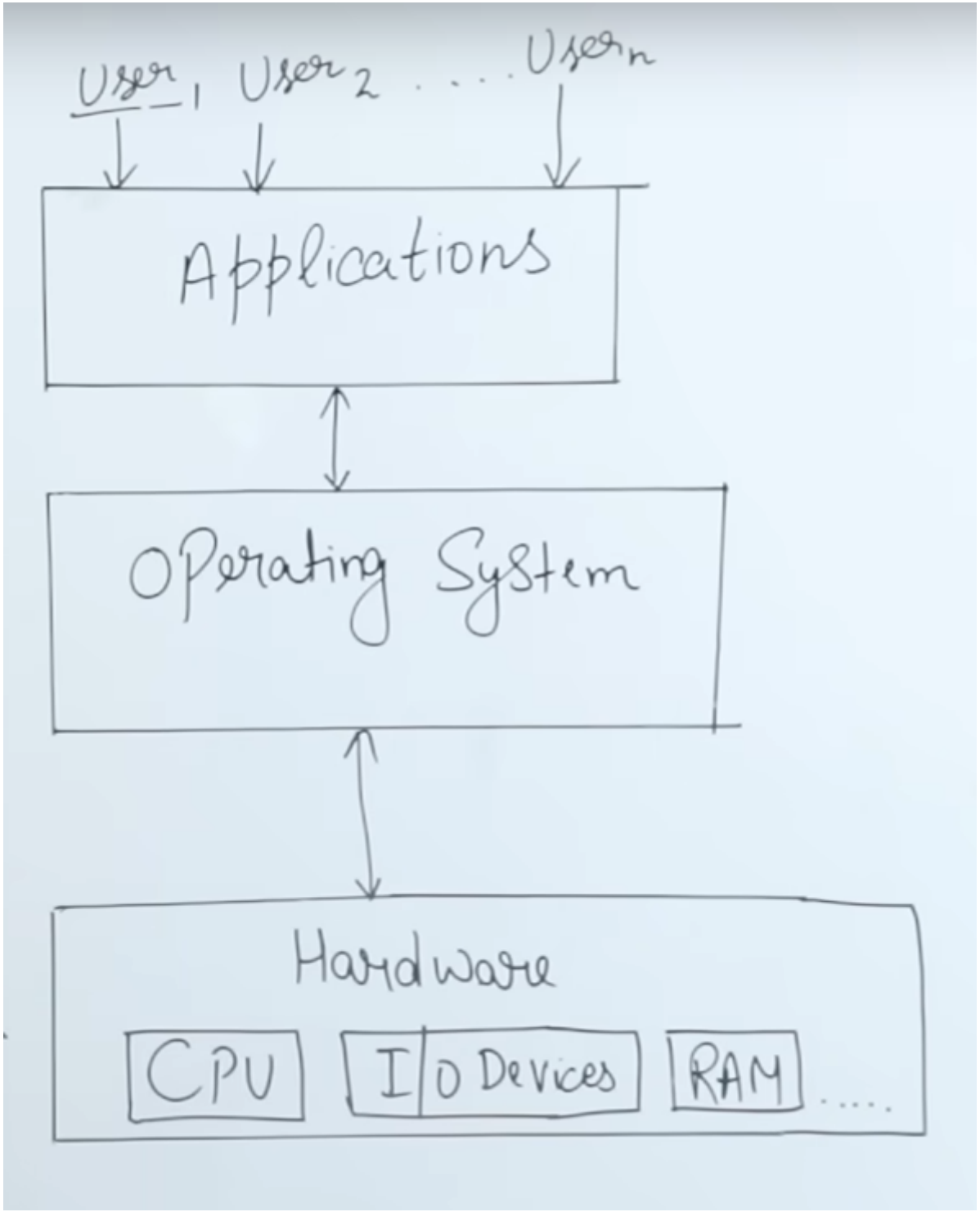
**Protection and Security**

- Goals of Protection
- Domain of Protection
- Access Matrix
- Implementation of Access Matrix
- The Security Problem
- User Authentication
- Program Threats
- System Threats Intrusion Detection

# Introduction To Operating Systems & It's Functions

Operating system is a software which works in kernel mode as an interface between computer hardware and the applications for end users. Examples include Windows, Linux, Android, MacOS etc.

Without an operating system, users would need to write commands or programs for every small task, making the interaction between user and hardware complex, tedious and time consuming.


User 1
User 2
User n
Applications
Operating System
Hardware
CPU
I/O Devices
RAM

**Primary goals of OS is to provide**

1. **Convenience** - Easy to use. Windows had acquired 95% market share in early 2000s because it was easy to use.
2. **High Throughput** - More number of tasks executed per unit time. Linux provides high throughput.

**Major Functionalities of OS include**

1. **Resource Management** - In servers, when multiple user access the server in parallel. the OS decides how much hardware needs to be allocated to each of them. Useful in parallel processing.
2. **Process Management** - CPU Scheduling algorithms to execute multiple processes.
3. **Storage Management** - Done through file system. Secondary storage devices like HDD, SSD etc are used by the OS to store data permanently.

4. **Memory Management** - A process that needs to be executed needs to be brought into the RAM or primary memory. Allocation and deallocation of processes in RAM is managed by OS.
5. **Security & Privacy** - Authentication, encryption, prevents processes from interfering other processes.

## Types of Operating Systems

1. Batch
2. Multi-programmed
3. Multitasking
4. Real Time OS
5. Distributed
6. Clustered
7. Embedded



### Batch Operating System

- Used in 1960s.
- A batch of similar kind of jobs is created and given to OS for execution.
- Punch cards, paper tape, magnetic tape were used to create the jobs.

- The jobs were given to an operator.
- Operator creates batches of similar kinds of jobs.
- Operator gives each batch one by one to the CPU for execution
- If the processes required I/O operations, the CPU remained idle during that time, which is a major drawback of Batch OS.
- CPU doesnt move to next job when it is idle due to I/O operations, which led to lot of time-consumption and inefficiency.
- Later on IBM came out with Fortran, IBSYS709X, which had monitors, and the jobs could be given to the CPU directly without needing an operator.

## Multiprogrammed Operating System

- Non-preemptive CPU scheduling of multiple processes within a RAM.
- Non-preemptive means CPU doesnt move to the next processes till it hasn't completely executed the current process, unless the process itself invokes an I/O operation or some other kind of interrupt.
- In the meantime, the CPU moves to the next process.
- This leads to better utilization of CPU, less CPU idleness, improved efficiency, higher throughput.

## Multi-Tasking/Time Sharing Operating System

- Preemptive CPU Scheduling of multiple processes within a RAM.
- Premptive means the CPU executes a process for a defined time and moves to the next process.
- If the process doesnt get executed in that time, it is scheduled to be executed again in future.
- Leads to better response times, better responsiveness by the CPU. Along with that, all the advantages of multiprogramming OS is also present.

## Real Time Operating System

- Two types - Hard & Soft Real Time OS.
- Involves time constraints
- There should be no delay between process invokation and execution by CPU.
- Hard Real Time OS - No time constraints or delays allowed. Used in high priority critical applications where the responses have to be immediate. Applications include military, alarm systems etc.
- Soft Real Time OS - Used in non-critical situations where some amount of delay can be tolerated. Applications are gaming, live video streaming etc.

## Distributed Operating System

- The execution environment is distributed to multiple devices spread across in different geographical locations.
- Each peripheral is loosely coupled with a central server through networking.
- Each peripheral has it's own memory, CPU etc and is connected with the entire network.
- If one device fails, that work can be done by other devices available.
- Availability and Fault tolerance is high.
- Load balancing can be done.
- Highly scalable.

## Clustered Operating System

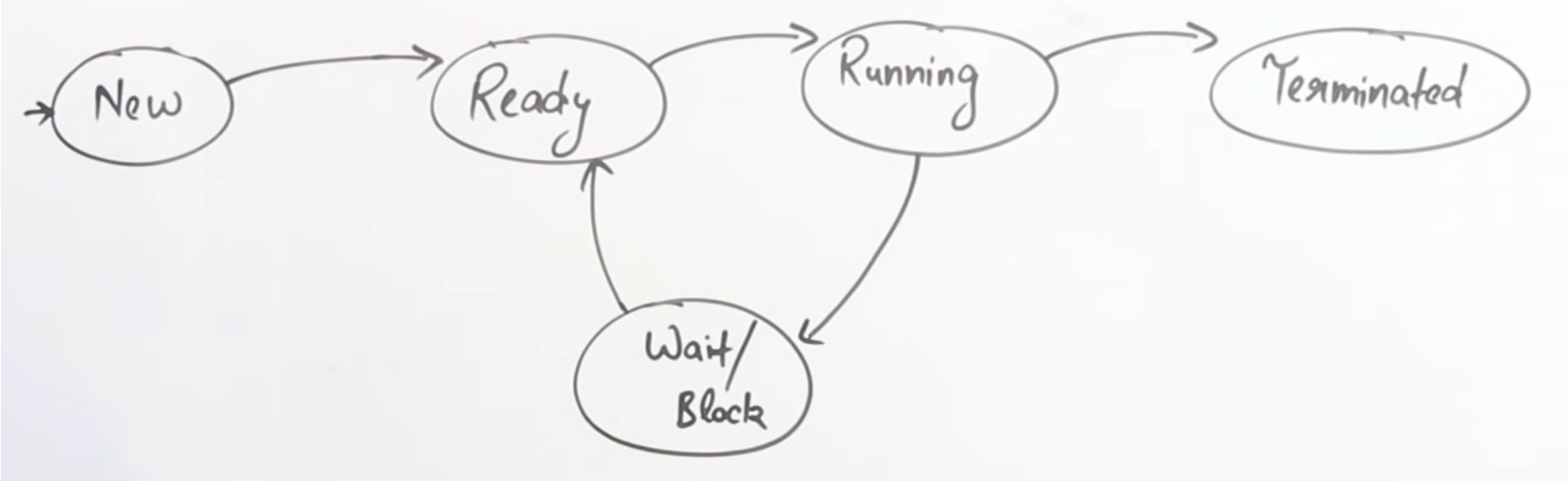
- Multiple devices connected to each other in a local network, creating a cluster.
- All devices act as one computer, and the computing power increases.
- Used in supercomputers.
- Availability and Fault tolerance is high.
- Load balancing can be done.
- Highly scalable.



## Embedded Operating System

- Made for a fixed functionality.
- Suitable for performing certain dedicated tasks repeatedly.
- Used in microcontrollers.

## Process States & Lifecycle






### New State

- New process is created and stored in secondary memory, and is ready to be invoked.

### Ready State

- The created process is brought into active state by user opening an application or executing some program.

- Ready Queue brings the process into the RAM.
- This is done by LTS or Long Term Scheduler.
- The job of LTS is to bring as many processes into the RAM as possible. Multiprogramming concept at play here.

**Running State**

- Process in the RAM is dispatched or scheduled to run in the CPU.
- The number of processes in running state will depend on the number of processors in the CPU. A uniprocessor CPU will execute one process at a time.
- The process is still in the RAM, but has been given a location for the CPU to work on.



**Terminated State**

- Process is deallocated from the RAM after it's execution is completed.

- Terminated state means the process is completely executed.
- The result of the computation will be stored in the secondary memory.

**Wait/Block State**

- 1st case is when a higher priority process is encountered, the current running process is put on hold and the high priority process is executed first. Multitasking at play here.
- The process is sent back to ready state as the high priority process is under execution.
- 2nd case is Time Quantum. The CPU runs a process for a fixed amount of time, and keeps moving to next process. The process which aren't executed completely, get scheduled to be executed in future.
- This is performed by Short Term Scheduler(STS). It helps in preemptive scheduling of processes.

- 3rd case is when a process has to perform an I/O operation, it is scheduled to be executed in future and CPU moves to the next process. This is where process is put into Wait/Block state and this queue is also present in the RAM itself.
- After the I/O operation is done, the process is put into Ready state, and is scheduled to be executed when its turn comes.
- 4th case, which is the worst case, is when all processes have to perform some I/O operation and are sent to the wait/block state.
- When this queue is filled, a new momentary state of Suspend/Wait is created, and this is where any new process that is sent to the Wait/Block state, it is put back into the Secondary memory.
- If memory in wait/block queue frees and the process in Suspend state has completed all I/O operation, then it is brought back to Wait/Block queue, otherwise it is kept in Suspend state till the time being.
- This work is done by Medium Term Scheduler(MTS).

- 5th case, when the Ready state queue is full, and a process with very high priority arrives, then processes are momentarily put into the Suspend/Ready state, and are resumed back when the high priority process is executed and space is created in the Ready State queue. This is also done by Mid Term Scheduler(MTS).
- Backing Store is done by MTS when Wait/Block queue is full and processes in Suspend/Wait state are waiting, so they are sent to the Suspend/Ready queue, and if the Ready Queue is also full, then the processes wait in the Suspend/Ready queue.

## System Calls

A programmatic way to shift from user mode to kernel mode.

1. File Related - For files. Open(), Read(), Write(), Close(), Create file etc.
2. Device Related - For hardware. Read, Write, Reposition, ioctl, fcntl.
3. Information - get Pid, attributes, get System time and data
4. Process Control - Load, Execute, abort, Fork(imp), Wait, Signal, Allocate etc.
5. Communication - Pipe(), Create/delete Connections, Shmget()
6. Protection & Security related - chmod etc

## Fork System Call

- Used to create a child process, which is a clone of the parent process, with a different ID.
- Fork() returns 0 for child, +1 for parent, -1 if child couldn't be created.
- The child process is executed parrallely and concurrently with the parent process.
- Total number of processes = 2^n, where n is the number of times fork() has been invoked.
- Total number of child processes = 2^n - 1, where n is the number of times fork() has been invoked.

## User Mode vs Kernel Mode

- User mode has mode bit = 1
- Kernel mode has mode bit = 0
- Applications used by the users run in user mode
- All core functionalities and drivers of OS run in kernel mode.
- Processor keeps switching between user and kernel mode.
- System calls are invoked to switch from user mode to kernel mode. A trap is generated by system call and shifts the process into kernel mode.
- The system call is executed in kernel mode and returns the result back into user mode.

## Process vs Threads in Operating System

| Process | Threads |
|---|---|
| 1. System calls involved in process. | 1. There is no system call involved. |
| 2. OS treats different processes differently. | 2. All user level threads treated as single task for OS. |
| 3. Different processes have different copies of data, file, code. | 3. Threads share same copy of code and data. |
| 4. Context switching is slower. | 4. Context switching is faster. |
| 5. Blocking a process will not block another process. | 5. Blocking a thread will block entire process. |
| 6. Independent. | 6. Interdependent. |

## User Level Thread vs Kernel Level Thread

| User Level Thread | Kernel Level Thread |
|---|---|
| 1. Managed by User level library. | 1. Managed by OS. |
| 2. Typically fast. | 2. Slower than user level. |
| 3. Context switching is faster. | 3. Context switching is slower. |
| 4. If one user level thread performs blocking operation, then entire process gets blocked. | 4. If one kernel level thread blocked, it doesn't affect other threads. |

## CPU Process Scheduling Algorithms

Two types - Preemptive & Non-Preemptive

| Pre-Emptive Algorithms | Non Pre-Emptive Algorithms |
|---|---|
| 1. SRTF(Shortest Remaining Time First) | 1. FCFS(First Come First Serve) |
| 2. LRTF(Longest Remaining Time First) | 2. SJF(Shortest Job First) |
| 3. Round Robin | 3. LJF(Longest Job First) |
| 4. Priority based | 4. HRRN(Highest Response Ratio Next) |
| | 5. Multilevel Queue |

## Timings in CPU Scheduling

1. Arrival Time - The time at which process enters the Ready queue or State.
2. Burst Time - Time required by a process to get executed on CPU.
3. Completion Time - The time at which process completes its execution.
4. Turn Around Time = { Completion Time - Arrival Time }
5. Waiting Time = { Turn Around Time - Burst Time }
6. Response Time = { Time at which a process gets CPU first time - Arrival Time }

## First Come First Serve(FCFS) CPU Scheduling Algorithm

1. Criteria : Arrival Time.
2. Mode : Non-Preemptive.
3. Use Gantt Chart to solve numericals.
4. Based on given Arrival and Burst Times of various processes, we find out their completion time, turnaround time, waiting time, response time using their formulas.
5. Also, find the average completion time, average turnaround time and average waiting time.
6. Response time will be equal to waiting time because FCFS is non-preemptive.

| Process No | Arrival Time | Execution Burst Time | Completion Time | TAT | WT | RT |
|---|---|---|---|---|---|---|
| ~~$P_1$~~ | 0 | 2 | 2 | 2 | 0 | |
| ~~$P_2$~~ | 1 | 2 | 4 | 3 | 1 | |
| ~~$P_3$~~ | 5 | 3 | 8 | 3 | 0 | |
| ~~$P_4$~~ | 6 | 4 | 12 | 6 | 2 | |

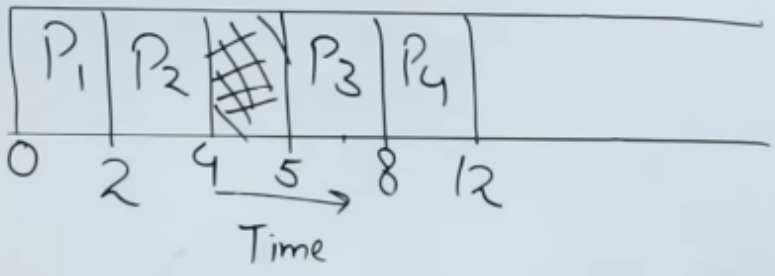
Criteria: "Arrival Time"

Mode: "Non-Preemptive"

CT - AT = TAT

TAT - BT = WT

Gantt Chart



## Shortest Job First(SJF) CPU Scheduling Algorithm

1. Criteria : Processes with least Burst Time.
2. Mode : Non - Preemptive.
3. Similar to FCFS, based on given arrival and burst times, we find out their completion time, turnaround time, waiting time, and response time.
4. Plot the Gantt Chart for processes being executed in CPU.
5. Calculate the average timings.
6. In non-preemptive algorithms, waiting time is always equal to response time.

| Process No | Arrival Time | Burst Time | Completion Time | TAT | WT | RT |
|---|---|---|---|---|---|---|
| ~~$P_1$~~ | 1 | 3 | 6 | 5 | 2 | |
| $P_2$ | 2 | 4 | 10 | 8 | 4 | |
| ~~$P_3$~~ | 1 | 2 | 3 | 2 | 0 | 0 |
| $P_4$ | 4 | 4 | 14 | 10 | 6 | |

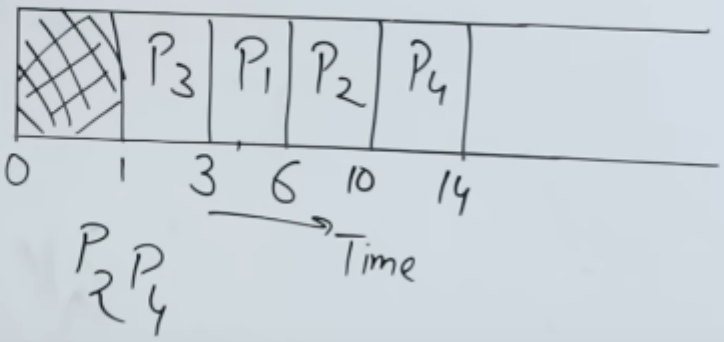
Criteria: "Burst Time"

Mode: "Non-Preemptive"

TAT = CT − AT

WT = TAT − BT

Gantt Chart



## Shortest Job First with Preemption(SJF + Preemption) CPU Scheduling Algorithm

1. SJF with Preemption is also known as **Shortest Remaining Time First(SRTF)** Scheduling Algorithm.
2. Criteria : Burst Time.
3. Mode : Preemptive.
4. More complicated numericals as you have to check the least burst time among all processes in the ready queue at every step.
5. If two processes have equal burst time, then consider their arrival times. The process with lower arrival time gets executed first.
6. Response time will be different than waiting time, because of preemption.

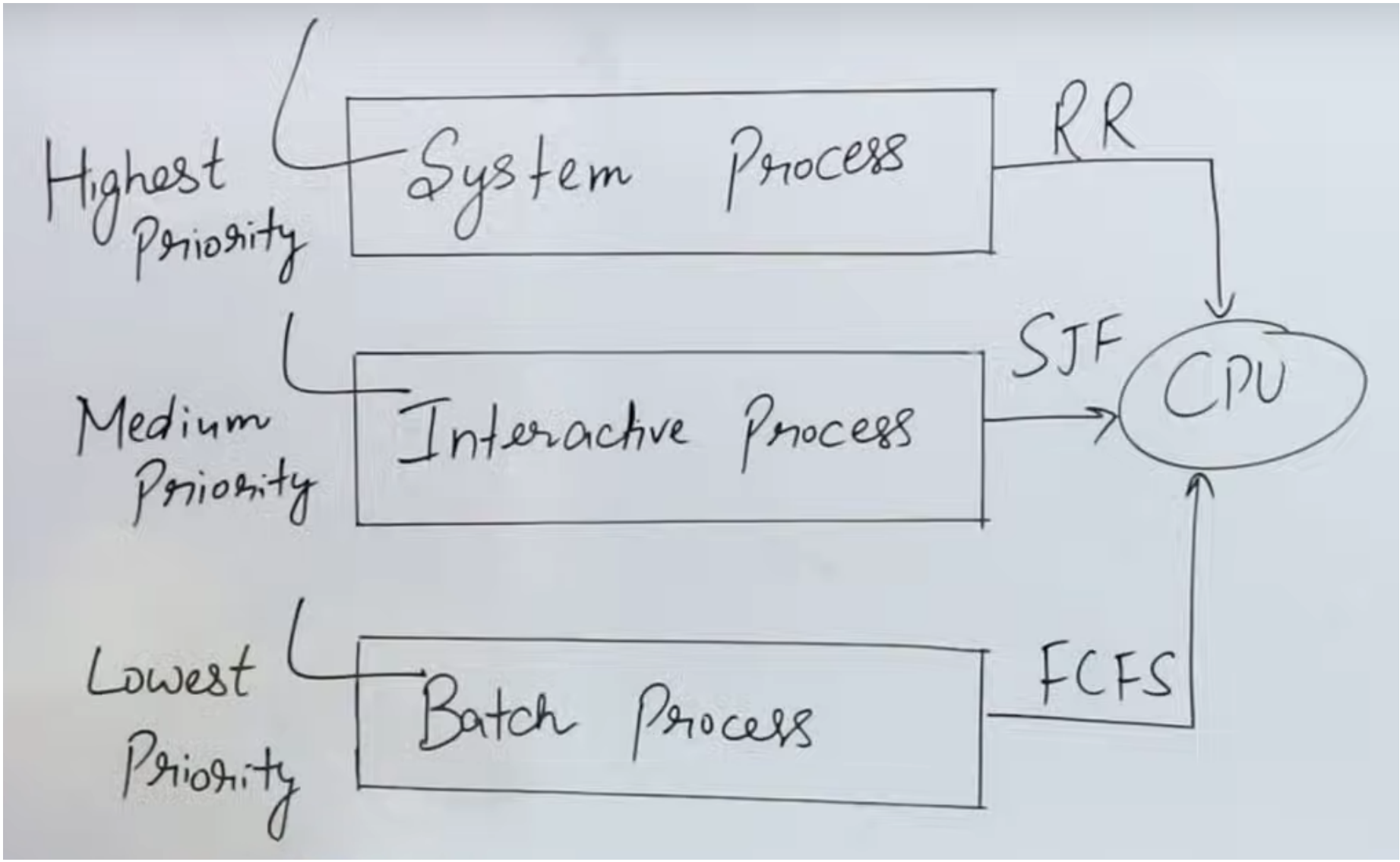
## Round Robin(RR) CPU Scheduling Algorithm

1. Criteria : Time Quantum
2. Mode : Preemptive
3. Use two Gantt charts for Ready Queue and Running Queue. Based on the given Time Quantum value, the arrival and burst times of processes, we calculate the CT, TAT, WT, RT.
4. **Context switching** happens when a running process is preempted back into the Ready Queue from the Running Queue when it has run for the specified Time Quantum value. The context of the Process Control Block of the running process is saved and the process is sent back to the Ready Queue and a new process is brought in to be executed.

## Pre-Emptive Priority CPU Scheduling Algorithm

1. Criteria : Priority.
2. Mode : Pre-Emptive.
3. Wether Higher Number = Higher Priority OR Lower Number = Higher Priority. Will be given in the question.
4. Based on the priority of processes, arrival and burst times, we calculate the CT, TAT, WT. Use Gantt chart for Running Queue.
5. In case of equal priority of two processes, use their arrival time. The one with lower arrival time gets executed first.

## Multilevel Queue Scheduling




1. Processes can be of different types, and there are different Ready

Queues for them.

2. System Processes have highest priority and are scheduled using Round Robin algorithm.
3. Interactive Processes like media players have medium priority scheduled using SJF algorithm and have a separate Ready queue.
4. Batch Processes are background processes, have lowest priority scheduled using FCFS algorithm.
5. Drawback is that the response times of batch processes will be very high, because the system processes will keep the CPU occupied leaving less time for interactive and batch processes to execute. This is the problem of starvation. This drawback is overcome by Multilevel Feedback Queue Scheduling.

## Multilevel Feedback Queue Scheduling

1. Lowest priority processes face the problem of starvation, which can be solved by feedback, where a low priority processes is progressively upgraded to a higher priority queue each time it is executed for a defined time quantum.
2. Processes which are already high priority do not face the problem of starvation.
3. The lowest priority process is executed for some time quantum, or it can be any algorithm for that matter. It is then given an upgrade to a queue having a higher priority and this process continues to the highest priority queue till the process has been executed completely.

## Process Synchronization

- Multiple processes can run in either serial mode, which means one process in execution at a time or in parallel mode, where multiple processes are executed parallelly at one moment.
- Two types of Processes - Cooperative Processes & Independent Processes.
- The execution of one cooperaive process has effects on other processes, because they share something in common. This can be variable, memory, code, resources like CPU, Printer etc.
- The execution of independent processes doesn't affect other processes running, as they do not share something in common.
- Cooperative processes need to be synchronized properly, otherwise it may lead to problems like race condition.
- Critical Section - It is a part of program where shared resources are accessed by various concurrent cooperative processes.

- If one program is using the critical section, then at the same time, no other program can use the critical section. If two programs use the critical section at the same time, then race condition will occur.
- If a program wants to use the critical section, then the code written in entry section has to be executed. If entry section executes without any problems, the program can enter into the critical section and execute the lines of code written there. When program finishes the critical section, an exit section is also coded. The stuff that is not common is put into the rest of section called the non-critical section. These help in preventing problems like race condition.
- Q. Do the producer consumer, printer spooler problem, which is the standard problem for multiple process synchronization.
- Q. Practice critical section solution using 'Lock' variable.
- Q. Practice critical section solution using 'Test_and_Set' instruction.

- Q. Practice critical section solution using 'Turn Variable(Strict alteration)' method.
- Q. Practice process synchronization solution using 'Semaphores' method.
- Q. Practice producer consumer problem using 'Semaphore'.
- Q. Practice Readers-writers problem using 'Binary Semaphore'.
- Q. Practice dining philosophers problem using 'Semaphore'.
- Q. Producer Consumer Problem Using Semaphores.
- Q. Readers Writers Problem Using Binary Semaphore.
- Q. Dining Philosophers Problem Using Semaphore.

## Producer Consumer Problem

C language program for consumer:

1. load Rc, m[count]
2. DECR Rc
3. Store m[Count], Rc

Consumer takes items from the buffer and processes/consumes them.

```
void consumer(void){
int itemc;
while(true){
        while(count == 0){
                itemc = Buffer(out);
                out = (out + 1) % n;
                count = count - 1;
                Process_item(itemc);
                }
```

```
        }
}
```

C language program for producer:

1. load Rp, m[Count]
2. INCR Rp
3. Store m[Count], Rp

Producer makes/produces an item and puts it into the buffer.



```
int count = 0;
void producer(void){
        int  itemp;
        while(true){
                Produce_item(itemp);
                while(count == n){
                        Buffer[in] = itemp;
                        in = (in + 1) % n;
```

```
                count = count + 1;
            }
        }
    }
```

Lets say n = 8

Three variables have to be tracked : in, out & count

The buffer and count variables are shared by both producer and consumer program.

| Buffer[0.....n-1] |
|---|
| 0 |
| 1 |
| 2 |
| 3 |
| 4 |
| 5 |

| Buffer[0.....n-1] |
| --- |
| 6 |
| 7 |

## Printer Spooler Problem

1. Load Ri, m[in]
2. Store SD[Ri], "F-N"
3. INCR Ri
4. Store m[in], Ri

Keep incrementing the 'in' variable and keep storing the processes in the spooler directory.



| Printer's Spooler Directory |
| --- |
| 0 |
| 1 |
| 2 |
| 3 |
| 4 |
| . |

| Printer's Spooler Directory |
| --- |
| . |

## Synchronization Mechanism

4 Conditions/Rules:

1. Mutual Exclusion
2. Progress
3. Bounded Wait
4. No assumption related to hardware, speed etc.



Rule 1 & 2 are primary conditions, they have to be fulfilled. Rule 3 & 4 are secondary conditions, which means they are not compulsory for synchronization, but we try to implement them.

## Critical Section Solution Using 'Lock'

```
do{
        acquire lock
        CS
        release lock
}
```

1. While(LOCK == 1) (Entry
2. LOCK = 1 Code)
3. Critical Section
4. LOCK = 0 (Exit Code)

Lock 0 means critical section is vacant

Lock 1 means critical section is full

# Critical Section Solution Using 'Test_and_Set' Instruction

```
while(test_and_set(&lock)){
CS
lock = false;
}
boolean test_and_set(boolean *target){
boolean r = *target;
*target = TRUE;
return r;
}
```

## Critical Section Solution Using 'Turn Variable(Strict Alteration)' Method

- 2 process solution
- Runs in user mode

| | **Process 'P0'** | **Process 'P1'** |
|---|---|---|
| Entry Code : | While(turn != 0); | while(turn != 1); |
| CS | critical section | critical section |
| Exit Code : | turn = 1; | turn = 0; |

## Process Synchronization Using Semaphores

Semaphore is an integer variable which is used in mutual exclusive manner by various concurrent cooperative processes in order to achieve synchronization. Two types - Counting(-Infinity to +Infinity) and Binary(0, 1).

Two Operations:

1. P(), Down, Wait
2. V(), Up, Signal, Post, Release

C Pseudocode for Down



```
Down(Semaphore S){
        S.Value = S.Value - 1;
        if(S.Value < 0){
        // Put Process (PCB) in Suspended List
        Sleep();
        }
```

```
        else{
        return;
        }
}
```

## C Pseudocode for Up

```
Up(Semaphore S){
        S.Value = S.Value + 1;
        if(S.Value <= 0){
        // Select a process from suspended list
        WakeUp();
        }
}
```

## Binary Semaphores

### C Pseudocode for Down

```
Down(Semaphore S){
        if(S.Value == 1){
                S.Value = 0;
        }
        else{
                //Block this process and place in suspened
list
                Sleep();
        }
}
```

### C Pseudocode for Up

```
Up(Semaphore S){
        if(Suspend list is empty){
```

```
            S.Value = 1;
        }
        else {
                //Select a process from suspend list and
                WakeUp();
        }
}
```

## Deadlocks

- If two or more processes are waiting for happening of some event, which never happens, then we say that these processes are involved in a state that is called deadlock.
- Two processes mutually waiting for an event, which never happens, this state is called deadlock.

4 necessary conditions for deadlock:

1. Mutual Exclusion.
2. No Pre-emption.
3. Hold & Wait.
4. Circular Wait.

**Resource Allocation Graph(RAG)**

To represent how the resources are allocated to processes, assigned multiple resources, RAG is used. Two types - Single Instance RAG & Multiple Instance RAG.

Methods to handle deadlocks:

1. **Deadlock Ignorance**(Ostrich method) - Trade off between deadlock ignorance and performance happens in this method. We just ignore the deadlock and do not write any code to prevent or manage it, so that it increases the performance of the OS. This is because the code for deadlock detection, prevention or correction is quite complex. Programmers tend to ignore it because deadlocks happen very rarely and they do not want to compromise on the performance of the OS becuase of this extra functionality.

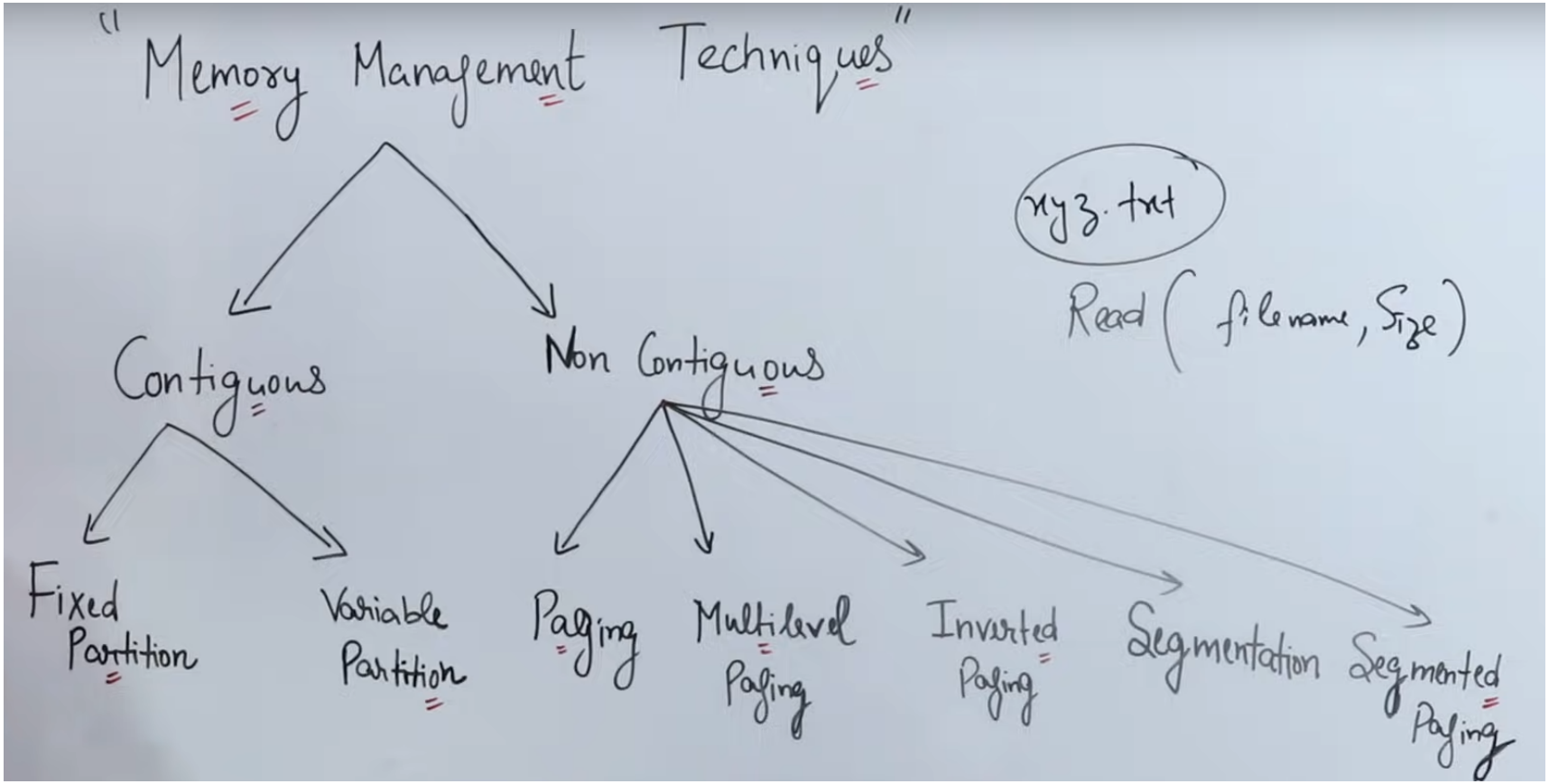
2. **Deadlock Prevention** - Prevent the 4 necessary conditions of deadlock formation from occuring, or atleast try to remove one of the condition, effectively preventing a deadlock. Prevent any four or all four.
3. **Deadlock Avoidance**(Banker's Algorithm) - Every time we allocate resources to a process, we check wether it is safe or unsafe to do it, and wether it will cause a deadlock. Banker's algo solves this problem.
4. **Deadlock Detection & Recovery** - We detect wether there is a deadlock, and if a deadlock is found, we try to recover the system from it.

## Memory Management

- Methods of managing primary memory for efficient utilization of memory.
- Anything that CPU works upon has to be brought into the RAM or the primary memory first.
- Degree of Multiprogramming focuses on bringing as many processes in the RAM as possible, so that the CPU is utilized properly, thereby increasing the performance of the system.
- Larger the number of processes in the RAM, the better CPU utilization is.
- This responsibility of bringing processes from secondary storage to the primary memory, preferably as many as possible is given to the Operating System.

## Memory Management Techniques

1. Contiguous - Fixed Partition(or Static), Variable Partition(or Dynamic)
2. Non-Contiguous - Paging, Multilevel Paging, Inverted Paging, Segmentation, Segmented Paging



## Contiguous Memory Management

### Fixed Size Partitioning | Internal Fragmentation

- No of partitions are fixed.
- Size of each partition may or may not be same.
- Contiguous allocation, so spanning is not allowed.
- Limit On Process Size : Processes can be allocated any partition, as long as the size of the partition is either greater than the process or equal to it.
- Internal Fragmentation : If a process is stored in one partition, then no matter how much space is left in that partition, it cannot be used to store any other process and goes wasted.
- Limited degree of multiprogramming : You cannot store processes that are more in number than the partitions, because of fixed paritions. If there are 4 partitions, you can store only 4 processes,

and their sizes are also limited by the maximum size allowed in their respective partitions.

- External Fragmentation : Although we have availability of space, we cannot store a process because the space is in multiple different slots.

## Variable Size Partitioning | Dynamic Partitioning

Partitions are not pre-made, memory is allocated to processes as and when they come. Variable sized partitions are made in runtime.

Advantages:

1. No chance for internal fragmentation.
2. No limitation on degree of multiprogramming or number of processes allowed to be stored.
3. No limitation on process size.

Disadvantages:

1. External fragmentation due to contiguous allocation.
2. Can be solved by 'Compaction' where the processes are shifted to fill up the empty spaces and make room for newer processes to be stored.

3. Compaction can be used but is undesirable because to shift a process, we first have to stop it from execution.
4. Allocation & Deallocation is complex because the number of partitions and processes is not fixed.

## Allocation Methods In Contiguous Memory Management

### First Fit

Allocate the first hole that is big enough.

Advantage

Simple and Fast.

Disadvantage

Can create big holes leading to fragmentation problems.

### Next Fit

Same as first fit but start search always from the last allocated hole.

Advantage

Search starts from previously left location

Disadvantage

Can create big holes leading to fragmentation problems.

**Best Fit**

Allocate the smallest hole that is big enough.

Advantage

Least internal fragmentation. Searches the entire least and then decides where to store the process.

Disadvantage

Can create small and tiny holes, that cannot be used to store any processes because of their small size. Very slow as well.

**Worst Fit**

Allocate the largest hole.

Advantage

Leaves maximum left over space, where other processes can be stored easily.

Disadvantage

Slow.

## Non-Contiguous Memory Management

### Paging

- Process can be divided and stored in multiple spanning memory locations.
- Solves the problem of external fragmentation.
- The division of a process is a costly process because holes in memory are created dynamically and their size keeps changing.
- Every time a process has to be stored in RAM, it has to be scanned for all the holes available, which makes it a time consuming process.
- Each division of a process is called a page. This paging is done when the process is in the secondary memory.
- Each slot or partition of a RAM is called frame.
- Page size is always equal to the Frame size.
- Number of pages = Process size/Page size

- Memory Management Unit(MMU) has the role of mapping the processes between CPU and RAM by converting the addresses generated by CPU to absolute addresses of RAM.
- MMU uses Page Table for this purpose. Page table contains frame numbers where that page is available in main memory.
- Every process has its own page table.
- The number of entries in the page table of a process will be equal to the number of pages a process has been divided into.
- CPU works on logical address. Logical address has page number and page offset. Offset means size of page. This logical address is converted to physical address by the MMU and then it performs mapping.

## Page Table Entries

The fields included in a page table are:

1. Frame Number
2. Valid(1)/Invalid(0) - Wether the page is present or absent in the specified frame number. Detects page fault.
3. Protection(RWX) - Used by OS for read, write, execute permissions.
4. Reference(0/1) - Least Recently Used(LRU) is a page replacement algorithm used while swap in and swap out of pages. A page which had been brought earlier in the frame, and is now being brought in again will be marked by 1, else 0.
5. Caching - Enable/Disable. Frequently used data is cached for better CPU access times.
6. Dirty/Modify - If data has been modified, but yet to be updated in the secondary memory, it is marked with 1, else 0.

Frame Number is a mandatory field while the rest others are optional.

**2-Level/Multilevel Paging**

Page table sometimes has to be divided further in order to accomodate it with the frame size. This is where multilevel paging helps.

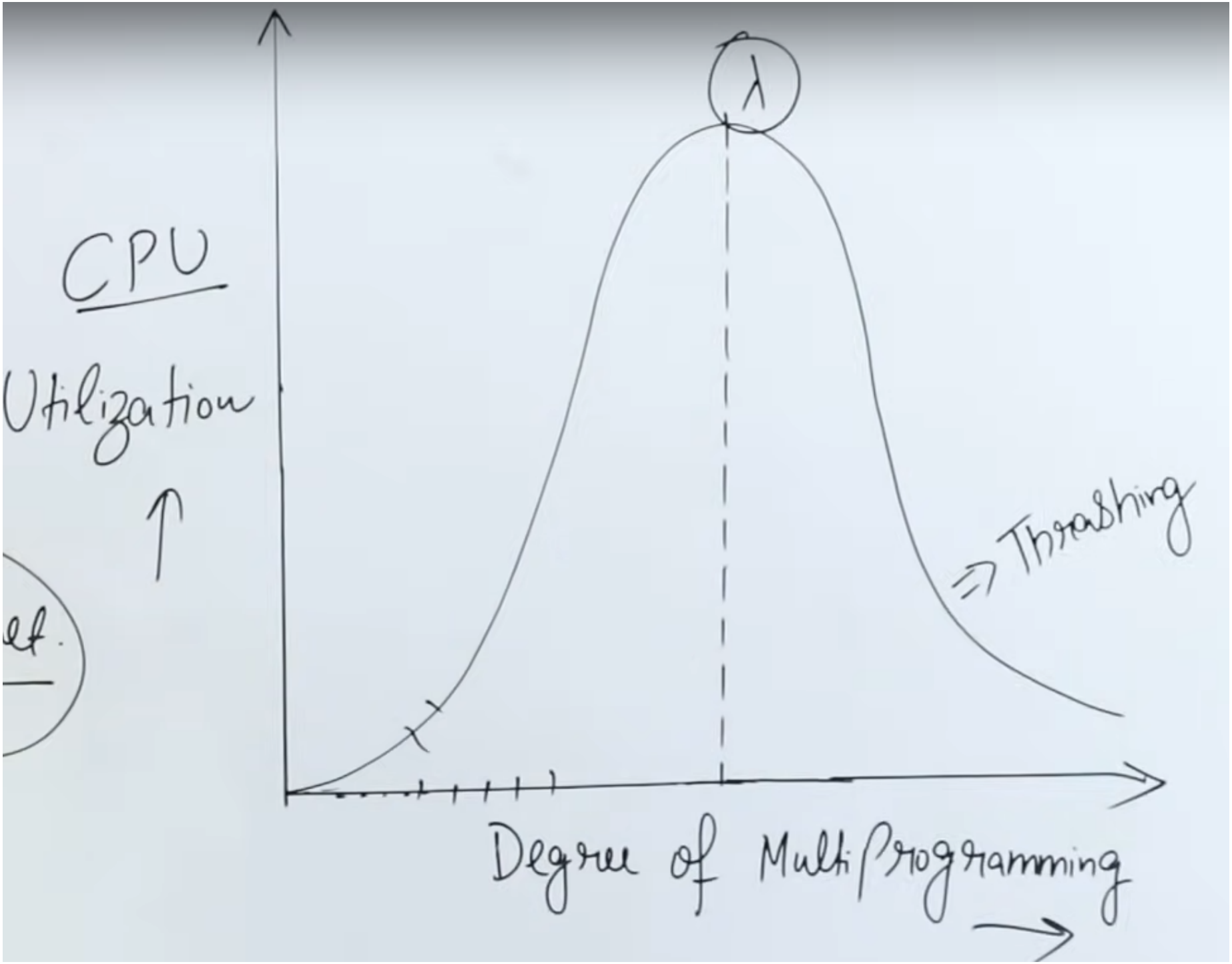
**Inverted Paging**

Instead of keeping a page table for all processes, a single global page table is created for all processes. The global page table contains frame number, page number and process ID.

Inverted page table has high searching times because of which it did not gain much popularity. Inverted page table saves memory but compromises on time. Nowadays, memory is getting cheaper and cheaper, but time is limited. Saving time is more preferable as compared to memory.

## Thrashing

When degree of multiprogramming in the RAM is too high, then it leads to increasing number of page faults, leading to page fault service times, which takes a page from the secondary memory and is very time consuming. This degrades the performance of the system. This scenario where degree of multiprogramming, instead of improving CPU utilization, it worsens it due to page faults is called thrashing. Solution is to either increase main memory, which may not be very viable. Another solution is to use a Long Term Scheduler(LTS).

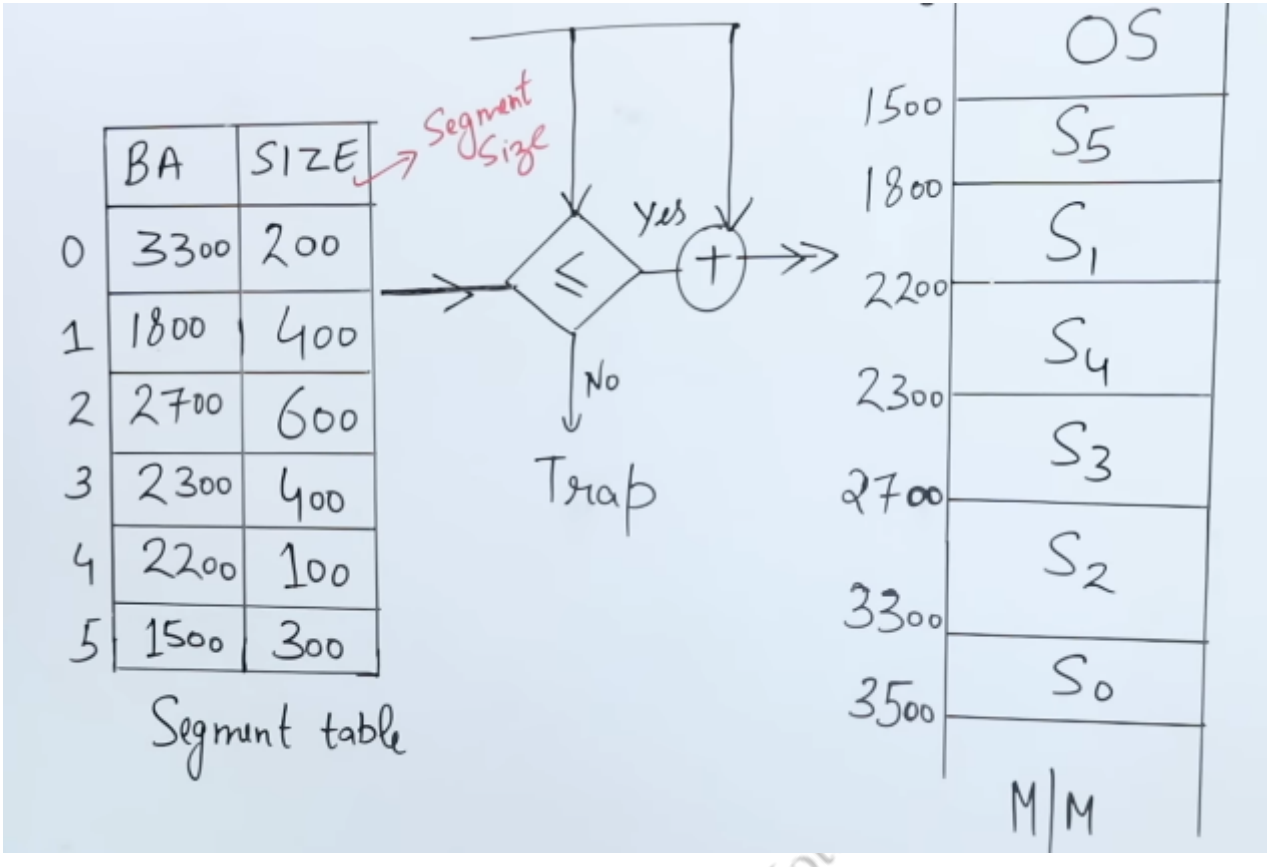

λ
CPU
Utilization
Thrashing
Degree of Multiprogramming

## Segmentation vs Paging

Paging divides processes in a way where half of the instructions can be in one page, and the rest in some other page. This leads to incompleteness in instructions. We divide the process into pages without caring about what is written in them.

Segmentation works on user's point of view. It doesnt divide a program directly like paging does. It creates segments of the program. These segments can be anything like functions etc. Size of each segment is of various sizes.

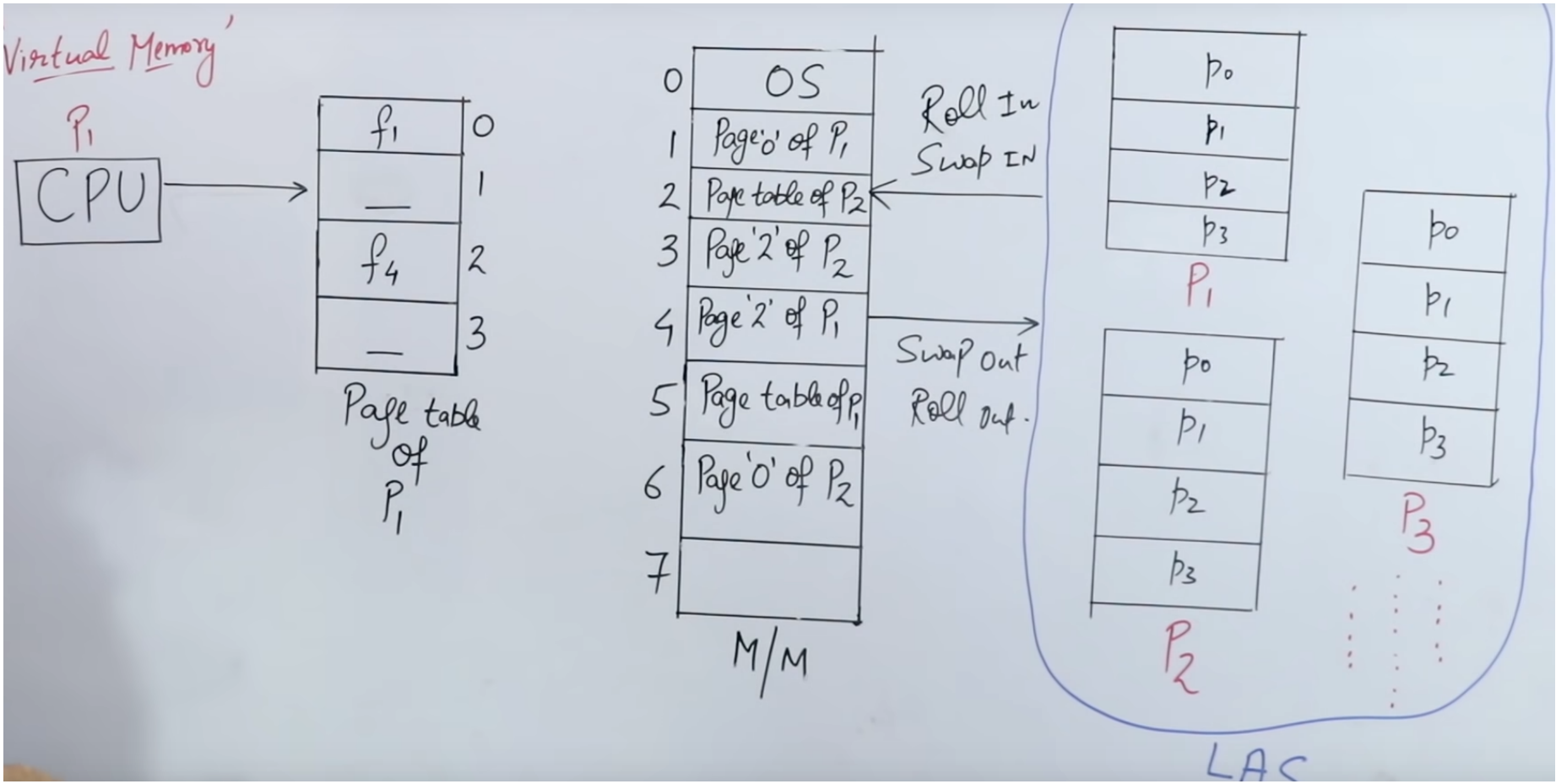
Segment Table contains the segment number, base address and the size of each segment.


BA
SIZE
Segment Size
0 3300 200
1 1800 400
2 2700 600
3 2300 400
4 2200 100
5 1500 300
Segment table
≤
Yes
No
Trap
+
OS
1500
S5
1800
S1
2200
S4
2300
S3
2700
S2
3300
S0
3500
M/M

## Overlay

- A process with a large size, which is larger than the main memory itself can be accomodated in the main memory with overlay.
- Operating systems do not provide any driver for overlay.
- The process is partitioned by the user itself, and each partition is brought into the main memory, executed by CPU and stored back in secondary memory, and the next process partition is brought into the RAM, and this continues till all partitions are done.
- The order at which partitions are going to be executed becomes very important.
- Overlay is used in embedded systems where the functionality is always fixed.
- Partitions created have to be independent. that means it should not be like one part of the code is in one partition and the other part is in second partition.

## Virtual Memory




In virtual memory, instead of storing all the pages of each process in the main memory, some pages of many processes are stored, and are constantly swapped in and out using page replacement algorithms as

and when they are needed by the CPU. Virtual memories are commonly used in modern day computers.

If a process page is not found in page table, a trap is generated, and context switching happens between the User and OS. Operating system now performs authentication for security purposes, and if authentication is passed, then operating system fetches the required page from the secondary storage and puts it into an empty space in main memory. The control goes back from OS to User.

Lets say 'p' is the probability that page fault occurs. Then,
Effective Memory Access Time (EMAT) = p(page fault service time) + (1-p)(main memory access time)

## Page Replacement Algorithms

1. FIFO - First In First Out
2. Optimal Page Replacement - Replace the page whose demand will be most late, which is not supposed to be used for longest dimension of time in future.
3. Least Recently Used(LRU) - Replace the least recently used page in past.
4. Most Recently Used(MRU) - Replace the most recently used page in past.

## Hard Disk Architecture In Operating Systems

- Multiple *platters* are connected together by a spindle and spin together in either clockwise or anticlockwise direction.
- Each platter has an upper surface and a lower surface.
- Each platter has a *read/write head* and all the heads are connected to an *Actuator Arm*. They are used for writing and retrieval of data and move back and forth on the platter.
- Each platter has many circular *tracks* on both the surfaces.
- The read/write head moves from one track to another.
- Each track has *sectors*.
- Data is stored in sectors.
- Platter -> Surfaces -> Tracks -> Sectors -> Data

## Disk Access Time

1. Seek Time - Time taken by R/W head to reach desired track.
2. Rotational Time - Time taken for one full rotation(360 degrees).
3. Rotational Latency - Time taken to reach desired sector(half of rotation time).
4. Transfer Time - Data to be Transfer/Transfer Rate
5. Transfer Rate(Data Rate) - No. of heads/surfaces X Capacity of one Track X No. of rotations in one second

## Disk Scheduling Algorithms

Goal : To minimize seek time.

1. FCFS(First Come First Serve)
2. SSTF(Shortest Seek Time First)
3. SCAN
4. LOOK
5. C-SCAN(Circular SCAN)
6. C-LOOK(Circular LOOK)

## FCFS

- The R/W head serves the requests in the request queue in a first come first serve manner.
- Direction is not necessary, the R/W head decides in which direction to move based on the request.
- Calculate the total number of R/W movement.

## SSTF

- The R/W head serves the requests which have the lowest seek time from its current position.
- The advantage is lower seek time and better response time, leading to better disk performance
- One problem can be starvation of certain requests.
- At every step, calculations have to be done for the shortest distance.
- Direction is not necessary, the R/W head decides in which direction to move based on which request is closest.
- Calculate the total number of R/W movement.

## SCAN

- Also called Elevator algorithm.
- Requires direction to be known.
- The R/W head moves till the last point in a direction.
- Serves all requests in a direction till the last request in that direction has been served and then serves requests in the other direction.
- The R/W after serving requests in a direction, moves till the last point, but when it serves requests in the second direction, it goes till the last request only.
- Disadvantage is that once the R/W head changes its direction, it will serve all the requests in that direction first, even if a request in the previous direction is much closer. Starvation of requests can occur.
- Calculate the total number of R/W movement.

## LOOK

- LOOK is similar to SCAN but the R/W head here doesnt need to move till the end of one direction.
- The R/W head serves all directions in one direction, goes till the last request and changes its direction from there.
- The extra distance covered by SCAN is not there in LOOK.
- Calculate the total number of R/W movement.

## C-SCAN(Circular SCAN)

- Direction must be known.
- The R/W head serves all requests in the direction specified, and goes till the end of that direction. This part is similar to SCAN.
- The R/W head travels to the last point of the other side. No requests are fulfilled in this movement.
- The R/W head now serves all the pending requests by starting from the extreme position, be it start or end.
- Calculate the total number of R/W movement.

## C-LOOK(Circular LOOK)

- Direction must be known.
- The R/W head serves all request in the specified direction, and changes direction from the last request and goes till the last request of the other end.
- When the R/W head is coming back, no request is served.
- The R/W head after reaching the last request of the other end, now serves all the remaining requests.
- Calculate the total number of R/W movement.

## File System In Operating System

File system manages how data is to be stored or fetched. Files are logically divided into blocks and these blocks are then mapped onto the sectors in a hard disk. This mapping may or may not be contiguous.

### File Attributes & Operations

| File Operations | File Attributes |
|---|---|
| 1. Creation | 1. Name |
| 2. Reading | 2. Extension(type) |
| 3. Writing | 3. Identifier |
| 4. Deletion | 4. Location |
| 5. Truncating | 5. Size |
| 6. Repositioning | 6. Modified Date, Created Date |
| | 7. Protection/Permission |
| | 8. Encryption, Compression |

## File Allocation Methods

1. Contiguous Allocation
2. Non-Contiguous Allocation - Linked List Allocation, Indexed Allocation

   These methods aim for efficient disk utilization and faster access times.



### Contiguous File Allocation

Advantages

1. Easy to implement.
2. Excellent read performance.

Disadvantages

1. Disk will become fragmented.
2. Difficult to grow file.

## Linked List File Allocation

### Advantages

1. No external fragmentation.
2. File size can increase.

### Disadvantages

1. Large seek time.
2. Random acess/Direct acess difficult.
3. Overhead of Pointers.

## Indexed File Allocation

Advantages

1. Supports direct access.
2. No external fragmentation.

Disadvantages

1. Pointer overhead(high number of pointers used).
2. Multilevel Index(for very large files).

## Unix Inode Structure

Unix Inode is a data structure that contains File attributes, direct blocks, single indirect blocks, double indirect blocks and triple indirect blocks. It is a hybrid approach that overcomes the shortcomings of the previous file allocation methods. 'I' is Index, 'Node' is the block in Inode.

Q. A file system uses Unix Inode data structure which contains 8 direct block addresses, one indirect block, one double and triple indirect block. The size of each block is 128 Bytes and size of each block address is 8 Bytes. Find the maximum possible file size?

# Protection & Security in Operating Systems

## Introduction

In the digital world, operating systems (OS) act as the foundation for secure computing. Protection and security are crucial aspects of an OS, safeguarding its resources and ensuring seamless, reliable operation.



## Understanding the Terms

1. **Protection:** refers to controlling access and usage of system resources (CPU, memory, disk, etc.) by processes and users. It focuses on internal threats, preventing interference and errors caused by unauthorized access or program malfunction.
2. **Security:** encompasses broader aspects, protecting the OS from external threats like malware, hackers, and unauthorized physical access. It ensures the system's **confidentiality, integrity, and availability (CIA triad):**

3. **Confidentiality:** Data should be accessible only to authorized users.
4. **Integrity:** Data and system resources should remain unaltered and consistent.
5. **Availability:** Authorized users should have uninterrupted access to system resources.

## Protection Mechanisms

- **User Authentication:** Mechanisms like usernames/passwords, biometrics, or multi-factor authentication verify user identity before granting access.
- **Resource Access Control:** Access permissions define which users and programs can access specific resources (files, devices, etc.).
- **Memory Management:** Techniques like memory isolation and virtualization prevent processes from interfering with each other's memory space.
- **System Integrity Protection:** Features like code signing and secure boot verify the authenticity and integrity of system software.

## Security Measures

- **Firewall:** Monitors incoming and outgoing network traffic, blocking unauthorized access attempts.
- **Anti-malware Software:** Detects and removes malware like viruses, worms, and trojans.
- **System Updates:** Regular updates patch vulnerabilities and address security flaws.
- **Physical Security:** Measures like access control lists and surveillance cameras prevent unauthorized physical access to the system.

## Key Points

- Protection and security are essential for reliable and secure computing.
- They address both internal and external threats.
- The CIA triad (confidentiality, integrity, and availability) serves as a guiding principle.
- Various mechanisms and measures work together to achieve protection and security.

## Goals of protection

- In one protection model, computer consists of a collection of objects, hardware or software.
- Each object has a unique name and can be accessed through a well defined set of operations.
- Protection problem - ensure that each object is accessed correctly and only by those processes that are allowed to do so.

## Security Violation Categories

- Breach of confidentiality - Unauthorized reading of data.
- Breach of Integrity - Unauthorized modification of data.
- Breach of Availability - Unauthorized destruction of data.
- Theft of Service - Unauthorized use of resources.
- Denial of Service(DOS) - Prevention of legitimate use.

## Further Exploration

- Explore specific protection and security features in different operating systems (Windows, Linux, macOS).
- Research emerging security threats and how OSes adapt to combat them.
- Understand the role of encryption in secure communication and data storage.