Operating System

The operating system is a software program that facilitates computer hardware to communicate and operate with the computer software. It is the most important part of a computer system without it computer is just like a box.

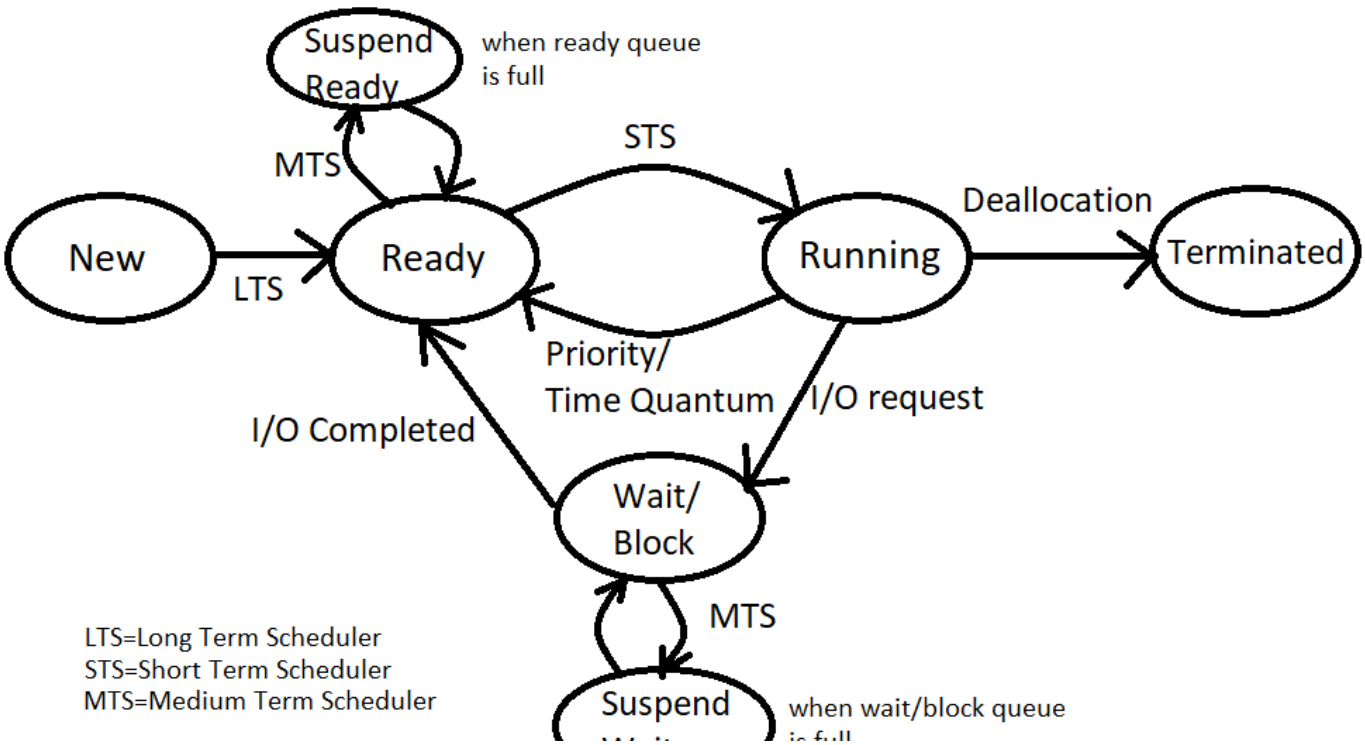
Responsibilities of OS:

1. Resource Management
2. Process Management (CPU Scheduling)
3. Storage Management (Hard disk)
4. Memory Management (RAM)
5. Security

Types of Operating System:

1. Batch: The users do not interact with the computer directly. Each user prepares its job on an off-line device like punch cards and submits it to the computer operator.
2. Multi-programmed: Adds maximum no. of processes in RAM. It's non-preemptive i.e. finishes one process before starting another unless the first process goes for I/O operations
3. Multi-tasking: Adds maximum no. of processes in RAM. It is preemptive i.e processes will be executed for a particular amount of time and then next process will be executed. Reduces response time
4. Real-time: can be hard or soft. Basically it has huge time constraint
5. Distributed: Machines are distributed geographically and connected through a network.
6. Clustered: Machines are clustered in a local network. Kind of makes a super computer by combining many machines to increase computational power
7. Embedded: It is designed to perform a specific task for a device that is not a computer. Example: Microwave, Dish washer etc

Process States:



System call: Used to shift from user mode to kernel mode to access functionalities

Types:

1. File Related: Open(),Read(), Write(),Close(), etc.
2. Device Related: Read, Write, Reposition, ioctl, fcntl, etc.
3. Information: getPid, attrubutes, get System time, etc.
4. Process Control: Load, execute, abort, fork, wait, signal, allocate, etc.
5. Communication: create/delete connections, pipe(), shmget()

Fork: Creates a copy of itself.

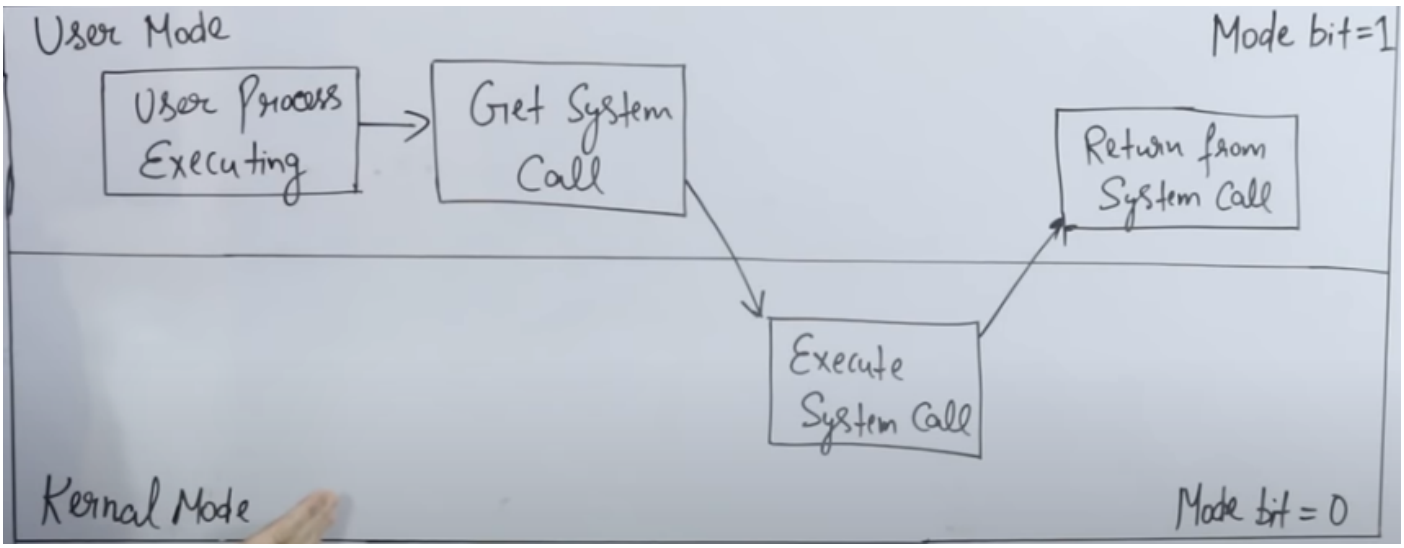
Number of forks in each step: 2^n-1

Number of times executed:2^n

YouTube Video

L-1.9: Questions on Fork System Call With Explan...

User mode vs kernel mode:



Threads:

Process	Threads
System calls	No system calls
Os treats different processes differently	All user level threads treated as single task for OS
Different processes have different copies of data, file and code	Threads share same copy of code and data only stack registers are separate
Context Switching is slower	Context Switching is faster
Independent	Interdependent
Blocking a process will not block another	Blocking a thread will block entire process

User level Thread	Kernel Level Thread
managed by user level library	managed by OS
faster	slower
Context Switching is faster	Context Switching is slower
Blocking a user level thread will block entire process	Blocking a kernel level thread will not block others

---

Scheduling Algorithms: Two types- Preemptive and Non-preemptive

Preemptive:

1. SRTF (Shortest Remaining Time First)
2. LRTF (Longest Remaining Time First)
3. Round Robin (Time Quantum)
4. Priority Based

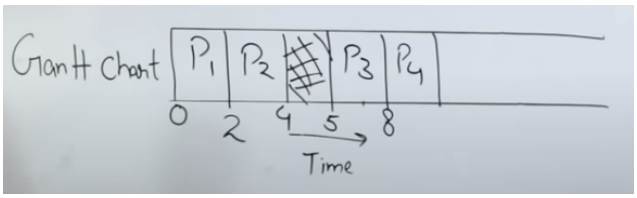
Non-preemptive:

1. FCFS (First Come First Serve)
2. SJF (Shortest Job First)
3. LJF (Longest Job First)
4. HRRN (Highest Response Ratio Next)
5. Multilevel Queue

Important Terminologies:

1. Arrival time: The time at which process enters the ready queue

1. Arrival time: The time at which process enters the ready queue
2. Burst time: Time required by a process to get executed on the 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: The time at which a process gets CPU for first time - Arrival time
7. Gantt Chart: Made for solving problems. Example:



YouTube Video

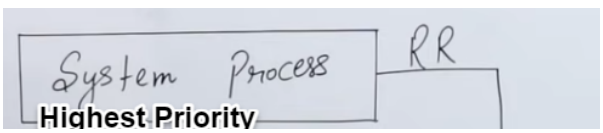
L-2.9: Example of Mix Burst Time(CPU & I/O both...

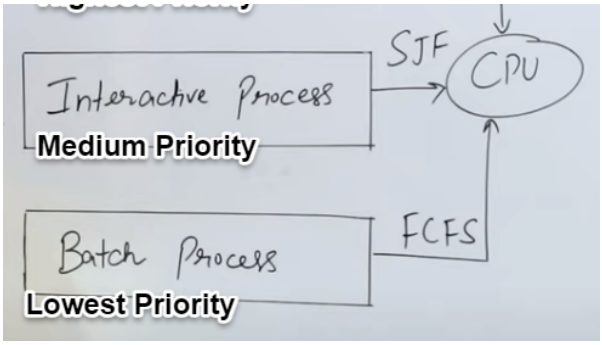
YouTube Video

L-2.7: Round Robin(RR) CPU Scheduling Algorith...

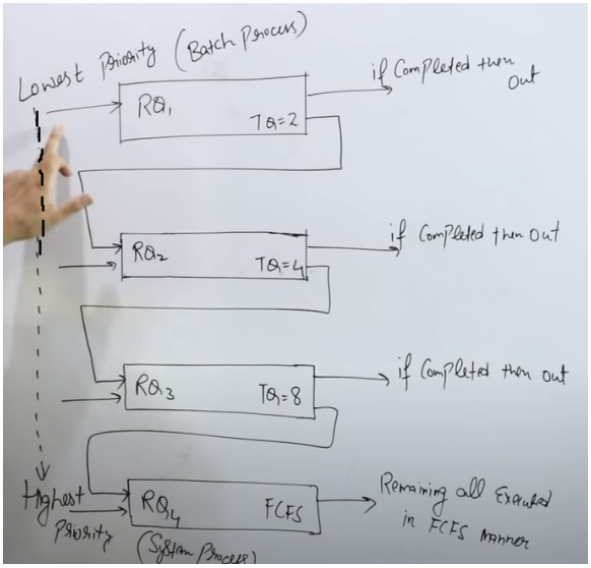
Multilevel Queue Scheduling:

3 types of ready queue depending on priority





Multilevel feedback queue:



Process Synchronization!!

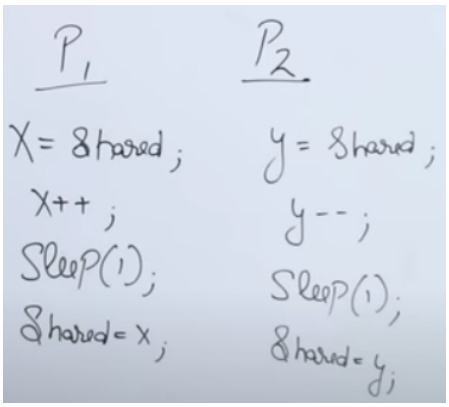
Two types of processes:

1. Independent Process
2. Cooperative process: Share common variables or code or resources or memory

For cooperative processes, we divide our code into two parts- non-critical section and critical section.
Critical section is the place with common things where we actually need process synchronization

Problems:

1. **Race Condition:** Let there be two processes running parallelly where variable shared=5.



Maan lo P1 pehle execute kiya.

1. x=shared //x=5

2. x++ //x=6
3. sleep(1) //P1 preempt

so now we go to P2

1. y=shared //y=5
2. y-- //y=4
3. sleep(1) //P2 preempt

Now one sec is over so it goes back to P1 but it starts from where left as everything is saved by process control

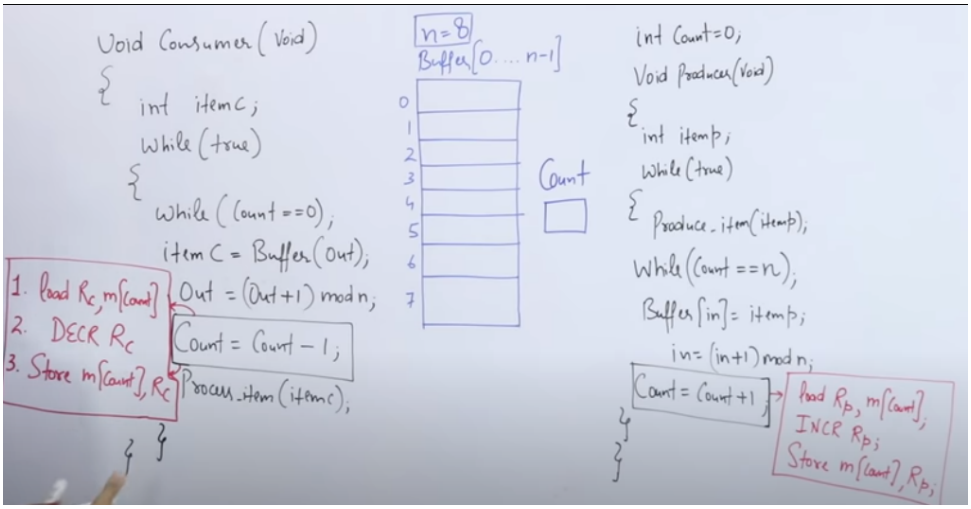
4. shared = x //shared=6

Now process terminated we go back to P2

4. shared = x //shared=4

So finally we get shared = 4 but if we see logically, in P1 we add 1 and in P2 we subtract one so shared should have remained 5. Therefore there is a problem

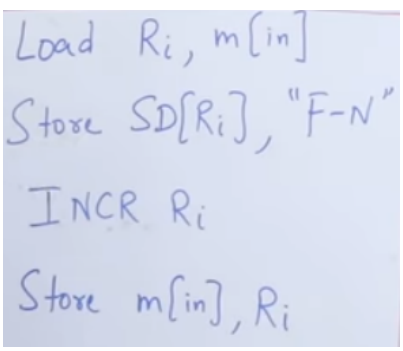
2. **Producer Consumer Problem**



The instruction count=count-1/+1 in both can be divided into three steps given in the red boxes. If for some reason after the 2nd red instruction, the process gets preempt then we get a problem because the count variable doesn't show right value

3. **Printer Spooler Problem**

So if there are many process using a printer, a spooler directory is created where whatever files are to be printed are added and one by one printer prints them. Every process before putting their file in spooler have to execute 4 lines:



If there are two processes P1, P2 and P1 gets preempted after line 3, then the value of m[in] hasn't changed and therefore P2's file will overwrite P1's.

changed and therefore P2 is not overwrites P1.

Solutions!

To achieve synchronization there are 4 conditions:

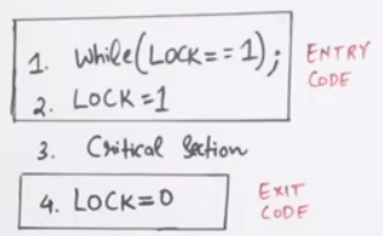
1. Mutual Exclusion (if there is already a process in critical section, no one else can enter)
2. Progress (no process should stop another when there is no process in critical section)
3. Bounded Wait (response time shouldn't be infinity for any process)
4. No assumption related to hardware of speed

1 and 2 are super important but 3 and 4 are optional but preferable

So synchronize we add an entry code and an exit code before and after the critical section code so that at a time only a particular process can go into critical section

1. **Solution using "Lock"**

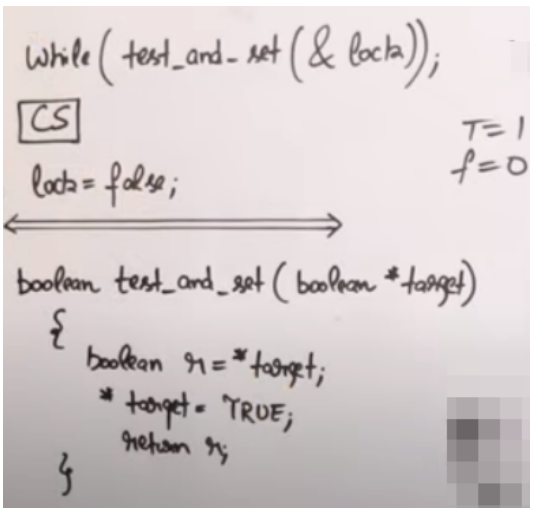
If Lock=0 , critical section is vacant
If Lock=1, critical section is full



This works but this solution doesn't follow mutual exclusion in special cases. How? Let's see
Let there be 2 processes P1,P2
Initially lock=0; P1 executes line 1 and them preempts.
P2 executes line 1,2,3 successfully as lock was still 0.
After this P1 again starts executing and executes line 2 and 3 successfully as it had already executed line 1 before the preempt.
Now P1 and P2 have both entered critical section at the same time which is against mutual exclusion.

2. **Solution using "Test and Set"**

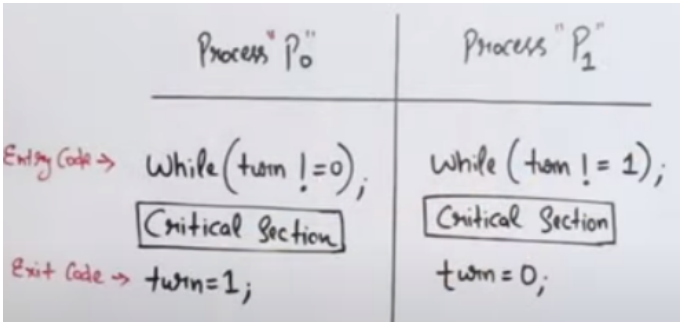
Basically in this one we combine both the lines in the lock solution so that either they both run or none of them run.



This follows mutual exclusion as well as progress

3. **Turn Variable/ Strict Alteration Method**

It is a 2 process method and works in user mode



All cool but one problem. Assume if turn =1 in the beginning and first P1 wants to go but it won't be allowed to even if critical section is empty because first P0 has to go in and out to make the value of turn =1. Therefore progress is not there.

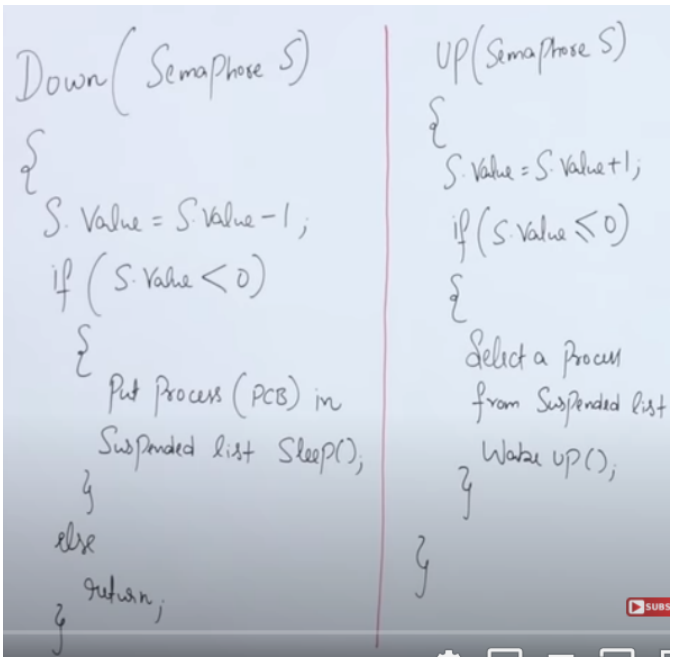
4. **Semaphore:** prevents race condition

Two types: 1. counting (can have values form -infinity to +infinity) 2. binary (has values 0 or 1)

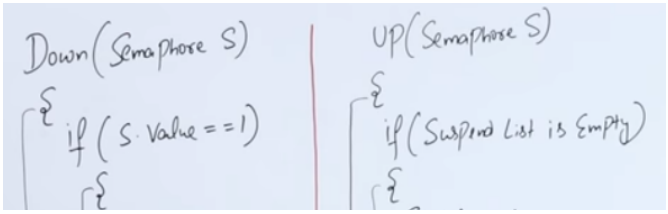
Operations in Semaphore:

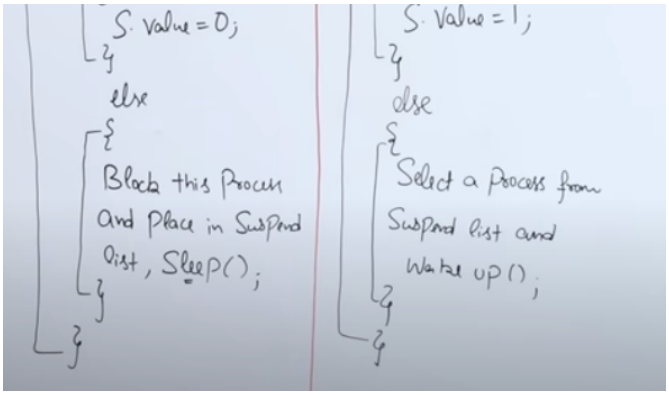
1. P(),Down,Wait // all synonyms used in entry section
2. V(),Up,Signal // all synonyms used in exit section

Counting Semaphore:



Binary Semaphore:





YouTube Video

YouTube Video

YouTube Video

Deadlock: If two processes are waiting on happening of some event which never happens, we say these process are involved in deadlock.
Necessary conditions for deadlock:

1. Mutual Exclusion
2. No preemption
3. Hold and wait
4. Circular wait

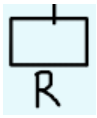
Resource Allocation Graph(RAG): Helps in figuring out if there is a deadlock or not
Two parts:

1. Vertex
    1. Process Vertex:

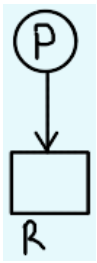
    2. Resource Vertex:
        1. Single Instance (Ex: CPU,Monitor):

        2. Multi Instance (Ex: Registers):

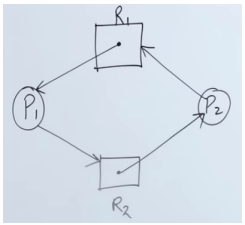
2. Edge:
    1. Assign Edge:



2. Request Edge:



Example of RAG:



Note: If single instance resource allocation graph has circular wait, then it is always deadlock

Methods to handle deadlocks:

1. Deadlock Ignorance (Ostrich method) (widely used): Just ignore it. Like when laptop hangs, we just restart it. It is done because deadlock is very rare and if we add a separate functionality to handle it, it might affect performance and speed
2. Deadlock Prevention: Before the deadlock try to remove at least one of the 4 necessary conditions of deadlock
3. Deadlock Avoidance (Banker's Algo): We check safe or unsafe for every resource we provide
4. Deadlock detection & Recovery

Banker's Algorithm: Pehle se hi bata do ki max har resource ki kitni need hogi

YouTube Video

Resources+Processes>Demand

Memory Management: We try to add as many processes as possible in RAM to increase CPU utilization

Memory Management Techniques:

1. Contiguous
    1. Fixed/Static Partition
    2. Variable/Dynamic Partition
2. Non Contiguous
    1. Paging
    2. Multilevel paging

2. Multilevel paging
3. Inverted paging
4. Segmentation
5. Segmentation paging

Contiguous:

Fixed Partitioning:

1. No. of partitions are fixed
2. Size of partition may or may not be fixed
3. Contiguous memory allocation

Advantages: Simple and easy to implement

Disadvantages:

1. Internal Fragmentation happens (Wastage of space like agar 4mb ka partition hai and your process is of 2mb, toh 2mb space waste ho raha hai
2. Limit in process size (Agar 4mb max partition ka size hai so even if 2 partitions khaali ho 4mb se zyada ka process nahi daal sakte)
3. Limitation on degree of multiprogramming (max no of process nahi daal pa rahe hai RAM ke andar)
4. External Fragmentation happens (If there is a process of 5MB and there is space in RAM but in fragments due to internal fragmentation like 1MB, 2MB,3MB. Although we have 6MB space still a 5MB process can't be put in RAM)

Dynamic Partitioning:

1. No. of partitions are not fixed
2. Size of partition may or may not be fixed
3. Contiguous memory allocation

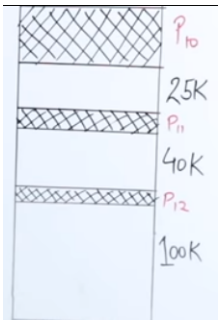
Advantages:

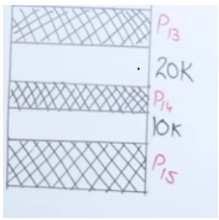
1. No internal fragmentation
2. No limitation on degree of multiprogramming
3. No limitation on process size

Disadvantages:

1. External fragmentation happens (Suppose two processes P2 and P4 of 4MB each leave RAM creating two holes on alternate position, then if we get a new process of 8MB it can't be added to RAM because it is not contiguous)
2. Allocation/Deallocation is complex (No. of process are dynamic as well as holes so its difficult to manage the partitions)

Algorithms to allocate:



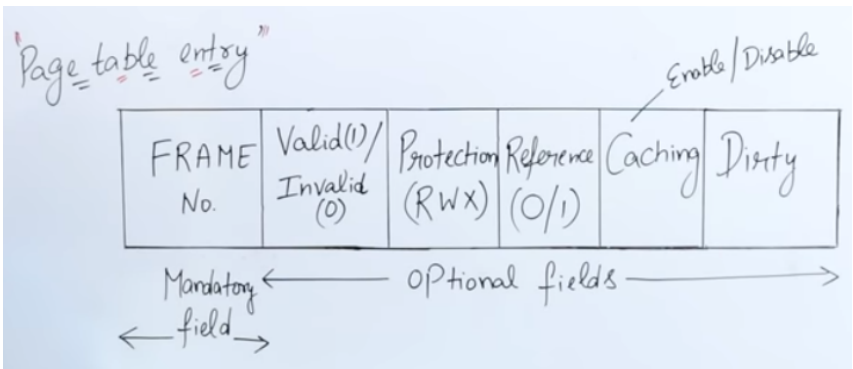


Suppose this is ram where the blank ones are holes. There are 4 algorithms to allocate:

1. First-fit: Allocate the first hole that is big enough (Very fast)
2. Next-fit: Same as first fit but start search always from last allocated hole
3. Best-fit: Allocate the smallest hole that is big enough (minimum internal fragmentation) (In 10K example)
4. Worst-fit: Allocate the largest hole (In 100K example)

Non Contiguous:
Prevents external fragmentation but the cost of division of processes is high
Paging: Processes are divided into equal partitions called pages in the secondary memory itself. The main memory(RAM) is also divide into partitions called frames.
Page_size=Frame_size
We create a page table for each page which stores which page is stored in which frame
Logical address: Page no and Page offset
Physical address: Frame no and Frame offset

YouTube Video

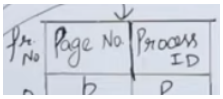


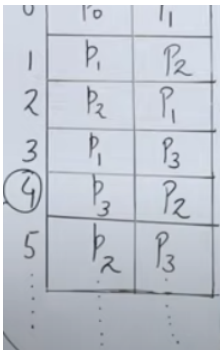
Reference: Has it been brought to main memory before or not. Method-LRU (Least Recently Used)
Dirty: Is the page modified or not

2-level Paging: If size of page table is greater than the main memory then we again divide the pages into subpages of the same page size

YouTube Video

Inverted Paging: So in normal paging each page table is also stored in main memory so it takes up a lot of space. So we create a global page table instead of single ones for all pages.
The structure of the page table is like:



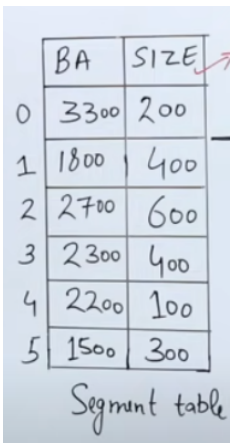


But this process takes up a lot of time in searching because pehle page no. dhundo phir process id phir dekho ki dono kahan match kar rahe hai and uska fram no. kya hai

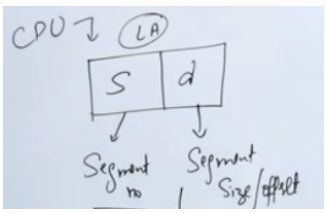
Thrashing: Thrashing occurs when a computer's virtual memory resources are overused, leading to a constant state of paging and page faults, inhibiting most application-level processing. This causes the performance of the computer to degrade or collapse.
To prevent thrashing:

1. Increase MM(main memory) size
2. Use long time scheduler to reduce degree of multiprogramming

Segmentation: In paging the process is divided into pages without looking at where it is divided which might divide like a function in between for which you might have to run two parts one after the other to make it run properly.
Instead in segmentation, we create segments of different sizes so that it creates logical segments.
We also create a segmentation table of the format:



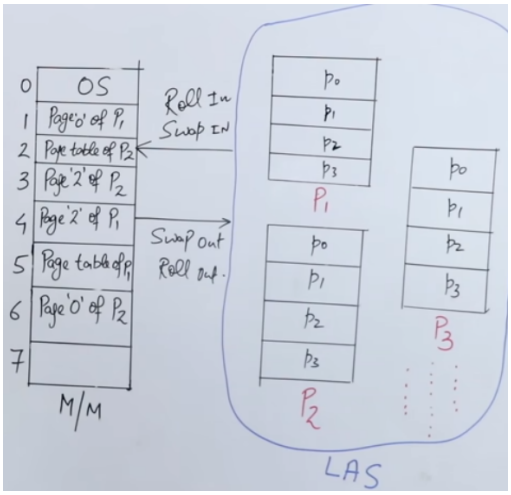
Logical Address:



Here d<=size because it tells how much part of a particular segment CPU wants to read

Overlay: Used when process size is greater than main memory. Used in embedded systems. User itself divides the processes into independent parts and adds it to main memory one by one.

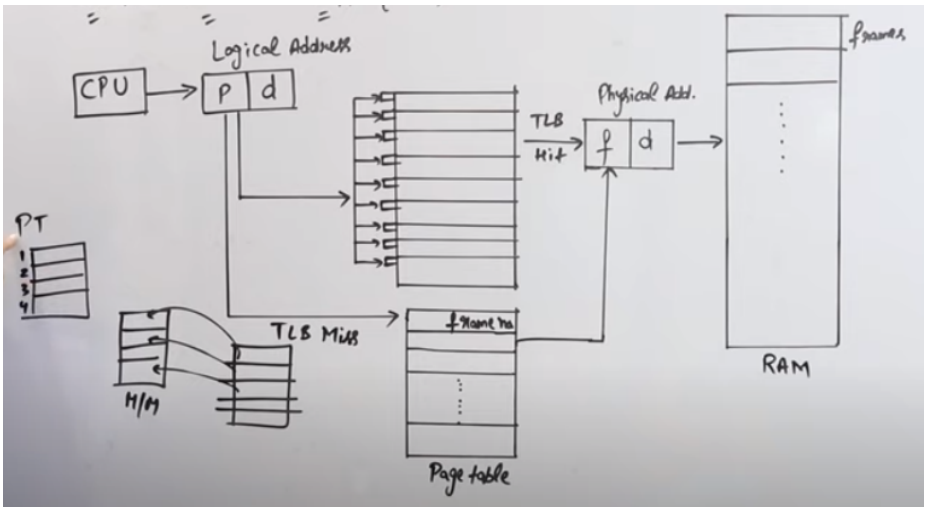
Virtual Memory: Virtual Memory is a storage mechanism which offers user an illusion of having a very big main memory. It is done by treating a part of secondary memory as the main memory. Therefore, instead of loading one long process in the
main memory, the OS loads the various parts of more than one process in the main memory.
We swap in/roll in pages whenever needed



But if by chance the page we need is not in the main memory, then there will be page fault. And OS will take a lot time to fix it because first it'll check the authentication of user then it will find the page in secondary memory, then add it to main memory, then change the value in page table, then the page fault is fixed. So it's a problem..

Effective Memory Access Time(EMAT)=p(page fault service time) +(1-p)(main memory access time)

Translation Lookaside Buffer(TLB): So basically let time taken to access the main memory be x. So in a normal process when CPU gives the logical address, we first look in the page table which is also situated in MM and then we again look in memory for the corresponding frame number. So time is x+x=2x



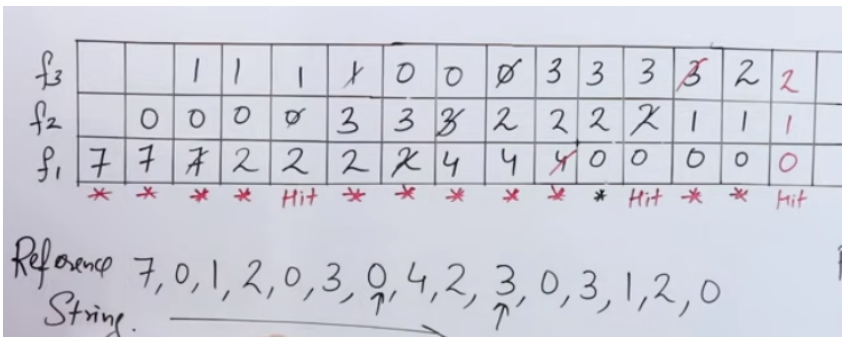
So to reduce the 2x time we create a TLB which takes less time to access. We add some frame numbers to it and if it is a hit then the time is less than x.
EMAT=(hit%)(TLB+x)+(miss%)(TLB+x+x)
Here the first part is for when it is a hit, so TLB time to access TLB and then x time to access main memory. Second part is when it is a miss so additional TLB time is wasted on top of the 2x time. TLB is useful only

if hits are more than miss
Also this is the EMAT if we consider there are no page faults.

Page Replacement: 4 methods

1. FIFO
2. Optimal Page Replacement
3. Least Recently Used
4. Most Recently Used

FIFO: First in first out
So whenever a page is not there in mm we get a page fault (marked by *) and if it is there then there is a hit. We replace in FIFO order as shown. Reference string is the sequence in which CPU is asking for pages

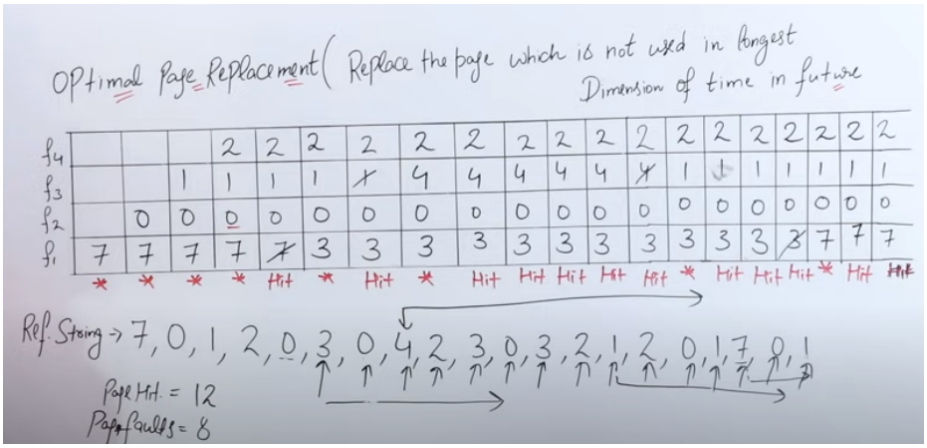


In this case page hit=3, page fault=12

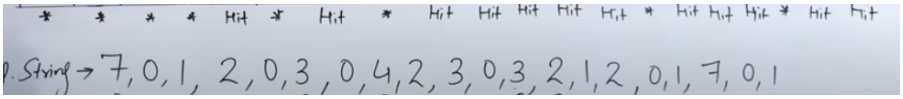
Belady's Anamoly: According to logic, if we increase no. of frames, no. of hits should increase but in fifo sometimes that does not happen.
Refer to this example

YouTube Video

Optimal Page Replacement: Replace the page which is not used in longest dimension of time in future

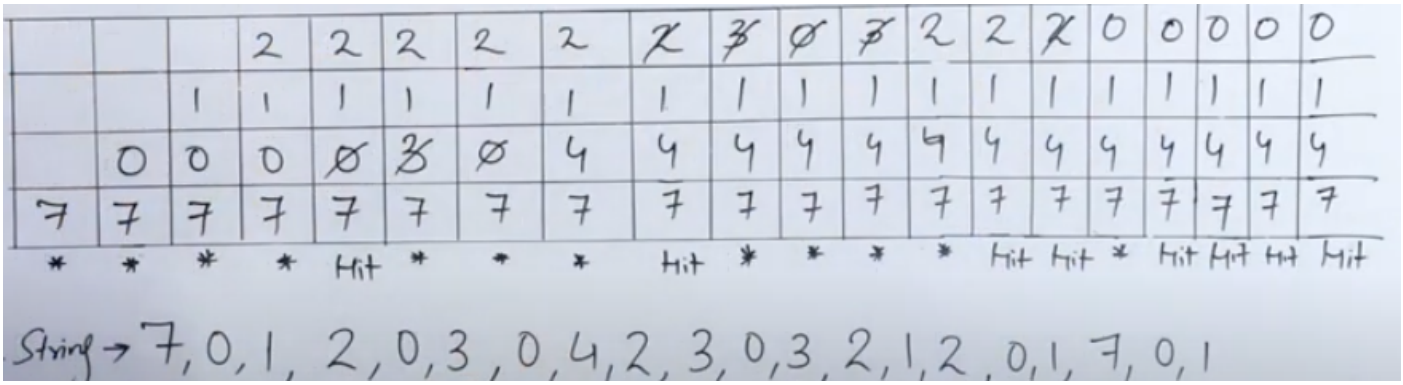


Least Recently Used Page Replacement: Replace the least recently used page in past



In this case: Page hits=12, Page faults=8

Most Recently Used Page Replacement: Replace the most recently used page in past

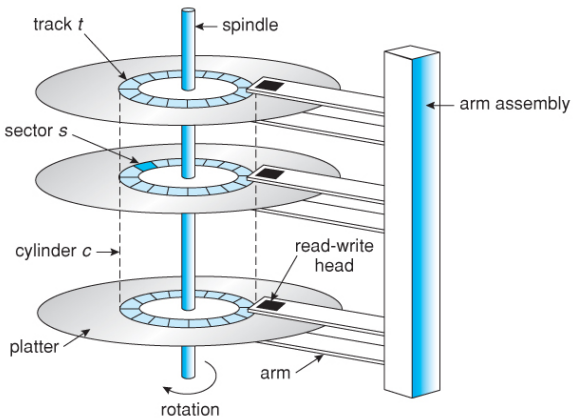


In this case: Page hits=8,Page faults=12

Disk Architecture:

Platters->Surface->Track->Sectors->Data

Disk size: P*S*T*S*D



Disk Access Time:

1. Seek Time: Time taken by R/W head to reach desired track
2. Rotation Time: Time taken for one full rotation
3. Rotational Latency: Time taken to reach desired sector (half of rotation time)
4. Transfer time: Data to be transferred/Transfer rate
5. Transfer Rate: No. of heads*capacity of one track*no. of rotations in one sec

Total disc access time= ST+RT+TT+CT+QT

CT=control time (optional)

QT=queue time (optional)

Disk Scheduling Algorithms: Methods used to minimize seek time

1. FCFS (First Come First Serve): Although it is easy to do as it simply goes in the order given but there is performance issue because it doesn't care how far the track are but it just goes in order so track movement becomes huge

YouTube Video

2. SSTF (Shortest Seek Time First): We look at the minimum distance and make the order of tracks accordingly. Performance is better, track movement is less but there can be starvation(like if there is 40,61,30,20,10, so even though 61 was 2nd but it has to wait a lot because it will be executed last). Also at each step you will have to search the track closest.
3. SCAN: We go till one end (say 199) and mark whatever requests come in between, then we change the direction and move till the last request in that direction. We take it till the end so that if by chance dynamically a request comes with a greater value then this thing will help
4. LOOK: First goes in one direction till the last request, then changes the direction and goes to the last request in that direction. SCAN mein jo extra space le rahe the for going till the end, woh bach ja raha hai
5. CSCAN (Circular SCAN): Like SCAN goes till the last value, then changes the direction but doesn't cover any track but goes till the end, after that changes direction again till the last request and marks the tracks. Starting direction is always given
6. CLOOK (Circular LOOK): Like LOOK goes till the last request, then changes the direction but doesn't cover any track but goes till the last request in that direction, after that changes direction again till the last request and marks the tracks. Starting direction is always given

---

File System in OS: It is software that manages files like how it is stored or fetched etc. File system logically divides the files in blocks and maps them to different sectors
Operations on Files: Creating, Reading, Writing, Deleting, Truncating(Delete content in file but not the file, i.e. attributes remain intact), Repositioning
File Attributes: Name, extension, identifier, location, size, modified date, created date, protection/permission, encryption, compression
File attributes are metadata of each file which is also stored is disk so that it is easy to locate and handle file

File Allocation Methods:
Main aim of this is to utilize data efficiently and make accessing faster

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

Contiguous File Allocation:
Directory contains 3 columns: file name, start and length
In the disk we go to the start position and contiguously add the file till the length.
Advantages:

1. Easy to implement
2. Excellent Read performance

Disadvantages:

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

Linked List Allocation: we divide each file in multiple blocks. The directory has 2 columns: file name and start. Each block is further divided into two parts containing a data and a pointer to the next block like in a linked list

Advantages:

1. No internal fragmentation
2. File size can increase

Disadvantages:

1. Large Seek Time
2. Random Access/Direct Access difficult
3. Overhead of pointers

Indexed Allocation: Directory contains 2 columns: File name and index block. Each file has an index block in which the position of every block is stored, like index in a starting of a book

Advantages:

1. Support direct access
2. No external fragmentation

Disadvantages:

1. Pointer overload
2. Multilevel Index (Index block might not fit in the block of hard disk so we'll have to do multilevel indexing like we did in multilevel paging)

Unix Inode structure:(not that important)

YouTube Video

Security:

1. Breach of Confidentiality
2. Breach of Integrity
3. Breach of Availability
4. Theft of Service
5. Denial of Service

Domain: Set of access rights

Access Matrix:

We can also create tables to represent the same

Security Violation Methods:

1. Masquerading: Pretending to be an authorized user to escalate privileges
2. Replay attack: As is or with message modification
3. Man-in-the-middle attack: Intruder sits in data flow, masquerading as sender to receiver and vice versa
4. Session hijacking: Intercept an already established session to bypass authentication

Security Measure Levels:

1. Physical: data centers, servers, connected terminals
2. Human: avoid phishing, social engineering
3. Operating System: Protection mechanisms, debugging
4. Network: Intercepted communication, interruptions

Threats:

1. Trojan Horse
2. Viruses
3. Spyware
4. Worm