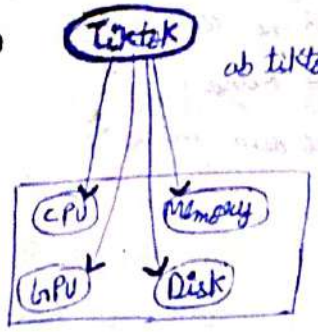
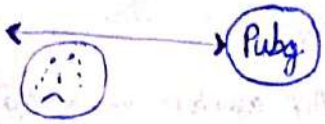
# OPERATING SYSTEMS

- Story :-

- 


ab tiktok chalane ke liye iske ye 4 chij chahiye
to ye sari chij le-lega aur hack kar lega
Kewal usi ke pass rahega full 100%.




ab pubg open karenge to nhi khulega kyuki System me na memory h na disk h na kuch h.

- ab operating System aayega aur bolega,

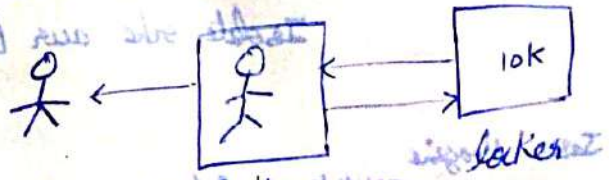
Isne 'Resource Management' kiya

TikTok → 5% CPU le
→ 10% Memory
→ 12% GPU

aur bhai pubg tu → 50% CPU le
→ 30% memory
→ 60% GPU

EK aur e.g.

Main bank se Pese le gya,




Main ne dek diya Bankar ko, Banker uncle ne locker se Pesa liya aur Phir mujhe diya

Interface

'OS is a interface'

Isi tarah

Koi user app aaya (like - tiktok) ⇒ OS ke Pass gya aur wo Storage diya ⇒ Resource CPU GPU

Tiktok → Iske developer ne code likha ki kitna Resource aur memory lage.

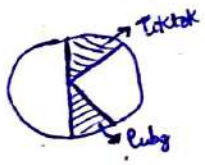
Wese hi Ruby → Iske v coder ne code likha hoga ki kitna Resource aur memory lage

To isse apps bulky hojayegi (dono App bahut bahut memory & Resource lagi)

Some code Repeat ho rahi h, Tiktok aur Ruby ke some code h wo repeat ho rahe h

Jo ki violation h DRY ka i.e. Do not repeat yourself.

Isiliye OS ka use karta h, taki Sare code OS ke Pass ho aur wo apne according dono ko resource aur memory de. (aur Memory aisa de ki dono ko lage ki wo Pure System me akela h).




★ Isolation and Protection

OS ke Madat se dono Tiktok aur Ruby Isolate rahe aur Protected rahe EK dusre se.

Jaise imagine Tiktok → Ruby me ghus ke health ko 0% karde to usse Ruby me us bande ki death ho Jayegi.

→ Isiliye OS use karta h ☺

↓

taki OS wo dono ko dur rakhe.

Notes :-

(b)

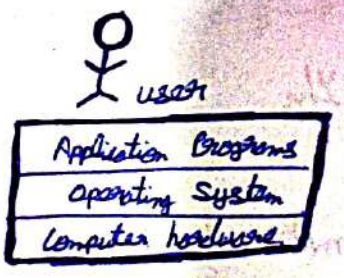
# AN INTRODUCTION TO OPERATING SYSTEMS

- Application software performs specific tasks for the user.
- System software operates and controls the computer system and provides a platform to run application software.
- (An operating system is a piece of software that manages all the resources of a computer system, both hardware and software, and provides an environment in which the user can execute his/her programs in a convenient and efficient manner by ~~hid~~ hiding underlying complexity of the hardware and acting as a resource manager.

## ★ WHY OS?

1. What if there is no OS?
   a) Bulky and complex app. (Hardware interaction code must be in app's code base)
   b) Resource exploitation by 1 App.
   c) No memory protection.
2. What is an OS made up of?
   a) Collection of System software

- An operating system function –
  - ★ Access to the computer hardware
  - ★ interface between the user and the computer hardware
  - ★ Resource management (AKa, arbitration) (memory, device, file, security, process etc)
  - ★ Hides the underlying complexity of the hardware. (AKa, abstraction)
  - ★ facilitates executions of application programs by providing isolation and protection

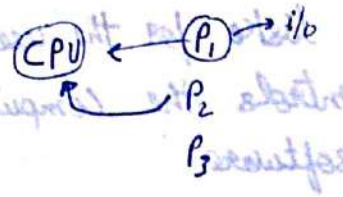



- The operating system provides the means for proper use of the resources in the operation of the computer system.

#2

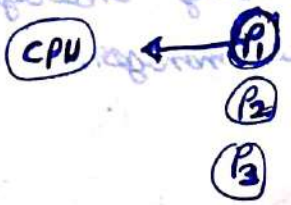
Learn all e.g. → Top → 4 are Imp

Goals of OS :- ① Maximum CPU utilization




CPU Khali na baithe agar $P_1$ Processor input/output Ke liye thora dar gya ho tab tak '$P_2$' Processor CPU Ke Kam Kare.

② Process Starvation na ho



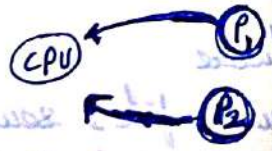

Soche $P_1$ - Processor Ka code while loop lga Ke chhor diya to wo to infinite time tak chalta hi rahega, aise to $P_2$ & $P_3$ Ko moka hi nhi milega.

③ High Priority execution

Jase $P_4$ aaya aur usko table execute hona do Kyoki ho PC-Scan Karega (Anti-Virus).

★ Types of OS :-

① Single Process OS :-




EK baar me EK hi Kam hoga

Pahle $P_1$ Ka Kam hoga Phir $P_2$ Ka.

Problem ↓ 'all 3 case'

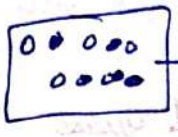
Isme $P_1$ agar I/O Kame chali gyi to Yha $P_2$ to moka nhi milega

∴ Maximum CPU utilization wala moka nhi milega.

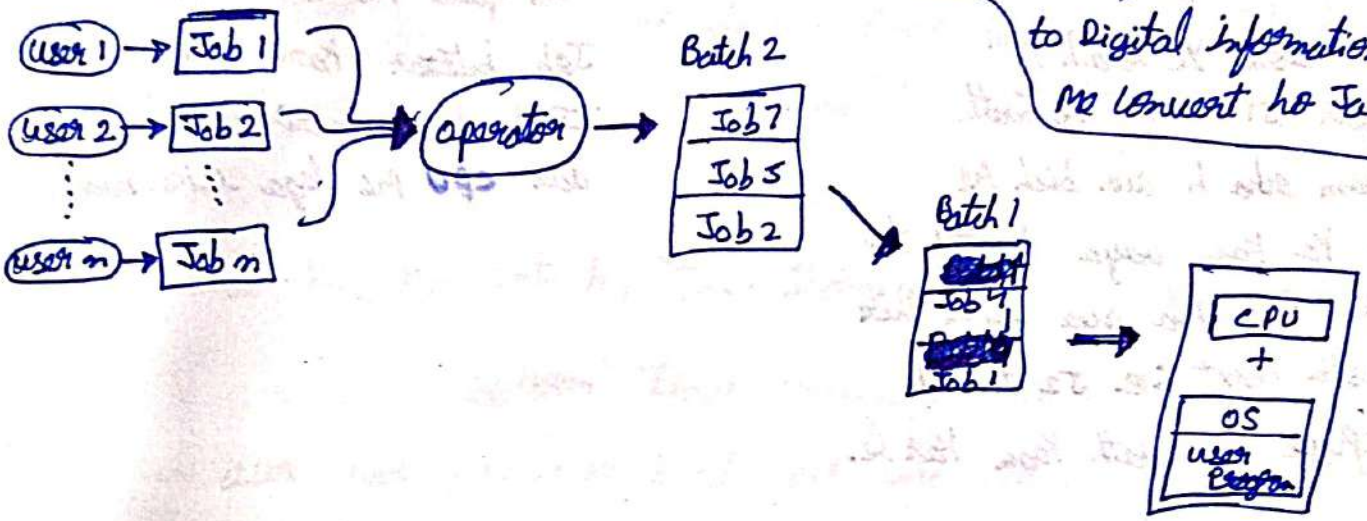
e.g. → MS DOS

## ② Batch Processing OS:-

Pahle ke time pe Punch card hoti thi



→ card → Isko scan kiye → to Digital information me convert ho jati thi.




- Isme 'n' user 'n' tarah ki Jobs lake aaye aur operator upper-upper se sort kar diya Jobs ko aur phir batch me divide kar diya like- Batch 1 me Job 1, Job 4 aur aise hi batch 2 bhi.

Note → Job 1 Pahle Jayega Phir Job 4

Ye EK-EK Batch Jayega OS me aur usko Single Process OS Jaisa karega.

Now, Isme bhi wahi 3 Problem h, To isme basically, enter huya hi nhi. batch me baat diye lekin kaam to Single Processor Jaisa hi tha.

e.g → ATLAS

## ③ Multi-Programming OS:-

→ Single CPU
→ Context Switching

(e.g → THE)

Jobs → J1, J2, J3, J4

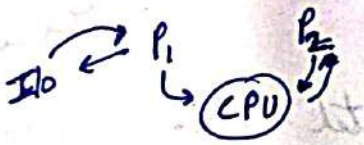
Redi queue

Isme ye hota h ki agar J1 → CPU ke saath kaam rha h aur bich me I/O ka kaam aaya to J1 to uske pass chla gya lekin phir uska next i.e. J2 tab tak CPU ke saath kaam kar le.

USB

↳ Is queue me agla wala Job intezar karta h ki Iske baad meri bari h aur CPU me laga iske baad.

- **Context Switching**




$P_1$ Job wapas ayega to wo CPU ka bacha huya kaam karega aur $P_2$ utne der me jitna kaam kiya hoga utna save karke wapas ~~aayega~~ chla jayega. Phir Job finally $P_1$ ka kaam khatam huya to $P_2$ dubara aake apna kaam continue karega!!

uska state address mode sab 'PCB' me save kar lenge.

"Process control Block"

**Related Story:-**

Study chemistry than Physics after every 1 hr.
So bookmark the page of chemistry before continue Physics

## (4) Multitasking OS :- (e.g → CTSS)

→ Single CPU
→ Context Switching
→ Time Sharing (Isme ye extra h)

Dekho last case me kya ho rha tha Jab $P_1$ → I/O ke liye Ja rhi thi tabhi '$P_2$' ko moka mil rha tha. Agar Socho $P_1$ → I/O ke liye Jaye hi nhi to aesa to dikkat hojayegi, baki ko to moka hi nhi milega.

to yha hum kya karte h, 'Time Sharing' karte h

hum 'Time Quantum' ⇒ 1000ms fix kardiye

aur has 1000ms ke baad next wale i.e. $P_2$ ko moka milega CPU ke Sath kaam karne ka. Phir 1000ms baad $P_3$ Phir .....
(cycle)

↓
It solve every Problem. ☺

## (5) Multiprocessing OS :-

→ Context Switching
→ Time Sharing
→ CPU ≥ 1

} "Ye Sabse Best h"

yha Multiple CPU h, to yha starvation ki Problem aur v kam hogi

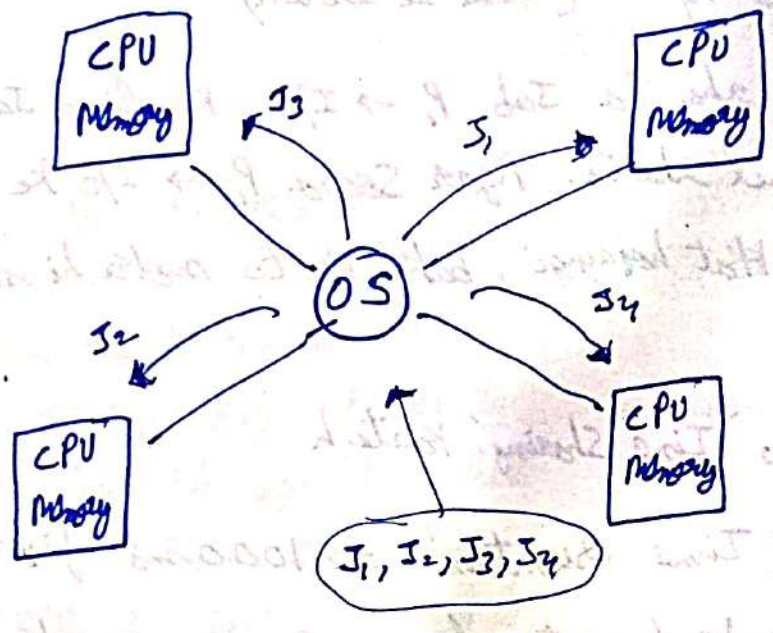
• Jese aaj ke CPU me 8 core h Matlab (8 tarah ke CPU h) → (desi bhasa me)
→ 8 logical Processor h.

Sabse best h ki Socho Pusna wale case me agar CPU li fail hogya to aye hum. yha ye best h ki 1 se Jyada CPU h, to agar EK fail huya v to dusra Sambhal lega.
→ (e.g → windows)

*ab there alag tarah ke CPU bhi hote h,,

'Loosely coupled autonomous intercomnected computers.'

## ⑥ Distributed OS:-




ab alag-alag CPU h, EK nare ghar par h, EK lone-babbar ke .. aur sare OS se connected h over the network (internet) → sab inter connected h.

ab OS apne marji se Jobs kisi v CPU ko dedega. ho sakta h koi CPU high end ho koi low end ho, to OS apne according kar lega.

eg → Leetcode pe koi Q solve kiye to wo apne linux pe kisi computer ko bolega iska Sol^n check karo aur batao iske kitne no. aaye. (to Jo computer khali hoga usko mota mil jayega)

## ⑦ RTOS (Real Time OS):-

→ Iska Jaswat wala h Jha error ka chance hi na ho.

eg → Jase flight udh rhi h to Pilot ko Pta hona chahiye sab-logo ki Jan ka sawal h.

ATC

→ No ERROR
→ Real Time (very-very fast execution)

eg → • ATC
• Industrial application

NOTES

(SUP)

★ OS goals :-

- Maximum CPU utilization
- Less Process Starvation
- Higher Priority Job execution

★ Types of operating Systems -

- Single Process operating System [MS DOS, 1981]
- Batch - Processing operating System [ATLAS, Manchester Univ. late 1950s - early 1960]
- Multiprogramming operating System [THE, Dijkstra, early 1960]
- Multitasking operating System [CTSS, MIT, early 1960s]
- Multi - Processing operating System [Windows NT]
- Distributed System [LOCUS]
- Real time OS [ATCS]

★ Single Process OS

↳ only 1 Process executes at a time from the ready queue (oldest).

★ Batch - Processing OS

→ Firstly, user prepares his Job using punch cards.
→ Then, he submits the Job to the computer operator.
→ Operator collects the Jobs from different users and sort the Jobs into batches with similar needs.
→ Then, operator submits the batches to the processor one by one.
→ All the jobs of one batch are executed together.

- Priorities cannot be set, if a Job comes with some higher Priority
- May lead to starvation (A batch may take more time to complete)
- CPU may become idle in case of I/O operations.

★ Multiprogramming :-

↳ increases CPU utilization by keeping multiple Jobs (code and data) in the memory so that the CPU always has one to execute in case some jobs gets busy with I/O.

- Single CPU
- Context Switching for Processes
- Switch happens when current process goes to wait state.
- CPU idle time reduced

★ Multitasking :-

↳ is a logical extension of multiprogramming.

- Single CPU
- Able to run more than one task simultaneously
- Context switching and time sharing used.
- Increases responsiveness.
- CPU idle time is further reduced.

★ Multi - Processing OS :-

↳ more than 1 CPU in a single computer

- Increases reliability, 1 CPU fails other can work
- Better throughput
- Lesser Process starvation (if 1 CPU is working on some Process, other can be executed on other CPU).

★ Distributed OS :-

↳ OS manages many bunches of resources, >=1 CPUs, >=1 memory >=1 GPUs etc.

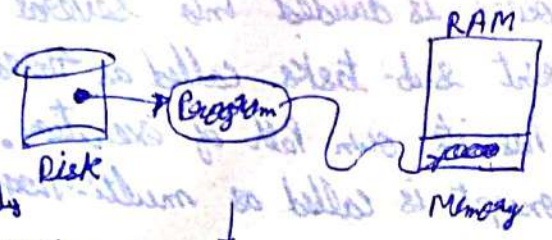
- Loosely connected autonomous, interconnected computer nodes.
- Collection of independent, networked, communicating, and Physically separate computational nodes.

★ RTOS :-

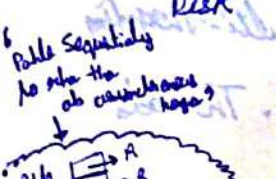
- Real Time error free, computations within tight - time boundaries.
- e.g. Air Traffic control system, ROBOTS etc.

#3 Story

ⓐ Process → Program under execution




Pehle Sequentially ho raha tha ab concurrently hoga,




Iska only kaa raha tha the current. Phir multi-threading se ↑ hum Parallely hoga.

• JPh → PNh

Job disk se RAM (RAM) me chl Jaye to Yahi Program - Process Kahelaega.

• Multi-tasking → Matlab bahut Sare Kaam ho rahe h Jese GTA v chl raha h aur Spotify chl raha etc.

• Multi-Threading → Jese 'word' khola ho, to usme Sare Kaam Parallely ho rahe h Jesa hum likh v rahe h, Saath-Saath wo spelling mistake v check Kar raha ya auto-correct v hojayega. Sare Kaam Parallely ho Jayega.

Thread

Independently cloud me v upload (Save) hote Jayega

★ Thread → Light weight Program.

Ye Chote Chote sarkaar Kaam (mujhe Kaal chahiye Kaam)

Matlab Single CPU me Multi threading nhi hoga. Jese aaj ko Time ka 8 core h (i.e. 8 CPU) to Max^m 8-Multi-threading ho sakta h.

Ye hume dikhaya sakta h kitna CPU h agar 1 core h (or 1 CPU) aur hum code 3-Threading ka likh de to wo nhi chalega

Notes

ⓑ Multi-Tasking Vs Multi-Threading

Program: A Program is an executable file which contains a certain set of instructions written to complete the specific Job or operation on your computer.

- It's a compiled code. Ready to be executed
- Stored in Disk.

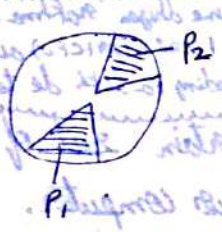
Process: Program under execution. Resides in computer's Primary memory (RAM).

Thread:

- Single sequence stream within a Process.
- An independent Path of execution in a Process.
- Light-weight Process.
- used to achieve Parallelism by dividing a Process's tasks which are independent Path of execution.
- Eg. Multiple tabs in a browser, text editor (when you are typing in an editor, spell-checking, formatting of text and saving the text are done concurrently by multiple threads.)

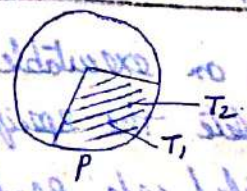
## Multi-Tasking

- The execution of more than one task simultaneously is called as multi-tasking.
- Concept of more than 1 Processes being context switched.
- Number of CPU 1.
- Isolation and memory Protection exists. OS must allocate separate memory and resources to each Program that CPU is executing.




## Multi-Threading

- A Process is divided into several different sub-tasks called as threads, which has its own Path of execution. This concept is called as multi-threading.
- Concept of more than 1 thread. Threads are context switched.
- Number of CPU >=1 (Better to have more than 1).
- No isolation and memory Protection resources are shared among threads of that Process.
  OS allocates memory to a Process; multiple threads of that Process share the same memory and resources allocated to the Process.




**Thread Scheduling:**

Threads are scheduled for execution based on their priority. Even though threads are executing within the runtime, all threads are assigned Processor time slices by the operating system.

## Difference between Thread context Switching and Process context Switching

| Thread context Switching | Process context Switching |
|---|---|
| • OS saves current state of thread and switches to another thread of same Process. | • OS saves current state of Process and switches to another Process by restoring its state. |
| • Doesn't includes switching of memory address space. (But Program counter, register and stack are included) | • Include switching of memory address space. |
| • Fast switching | • Slow switching |
| • CPU's cache state is Preserved | • CPU's cache state is flushed. |

Cache Space Jisa hota h

ab CPU baar-baar Memory (RAM) se space layega to time lagega.

To wo Pehle hi cache me 1 Space Path leta h ab Threading ke case me 1 cache me hi sabhi thread kaam karenge to wo Pehle se hi reserved kar leta h.

(Isliye Fast Switching hoga Baki threads ke beech me)

aur Process me sab alag-alag space lenge to Pehle, Pehle wala Process cache use karega Phir wo flushed hogya Phir dusra Process aaya aur naya cache use kiya Isse Switching karne me time lagta h.

# #4 Components of Operating System

ⓐ

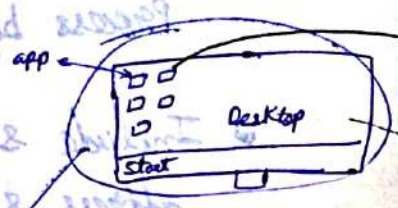
① user space ← Yaha pe app run hongi conviently

② Kernal ↔ Isko pass access hota h underlying hardware ka

• user space kuch bola to kernal jaega hardware pe aur wo kam isko diga. That's why kernal is Heart of OS.

• Isko pass hardware access nhi hota

• Ye convient environment provide karta h user app ko.




→ Ye jo icon dikh rha h usko user space

→ aur Isko EK aur chij bolte h → GUI

To humare Windows/Mac me jo user space hota h usko → GUI bol sakte h
↳ Ya CLI bol sakte h

• GUI → hum normally mouse se click karte h app ko use karte h (Ya user directly GUI se interact karta h)

• CLI (Command line interface) → Isse 'terminal' (Linux me hota) / Powershell (Windows me hota) → hum Yaha pe kuch command likhte h aur system leta h aur usko execute karta h (like open app)

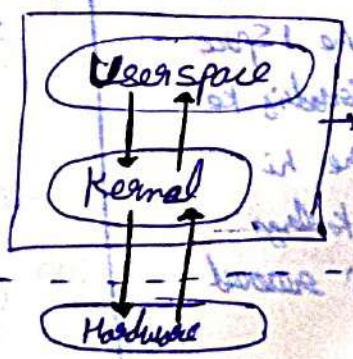
ⓐ Create New folder :-

(e.g.) • GUI se → mouse se new folder bna lenge.

• CLI se → kuch command likhenge aur phir folder bna. → (e.g.) Linux me → (mkdir) (isse new folder bn jaega)

• user space hi 'interacts with kernal'.

#2 Kernal → • Heart of OS
• interact with hardware




• Function of Kernal :-

① Process management Yahi karwata h Isse :-
ⓐ Process creation, Termination
ⓑ Process & Thread schedule
ⓒ Process Synch
ⓓ Process Communication → Ho P1, P2 → Isko sahi h / I know but wahi aapas me baat karne de

② Memory management
ⓐ allocate/deallocate
ⓑ free space mng.

③ file management → create/delete
→ directory management → tree ke form me rakhta h

Wo kam v Kernal karwata h.

④ I/O management :-

• management & controlling of all I/O devices

Imp. Kam :-
a) Spooling
b) Buffering
c) Caching

• Types of Kernel :-

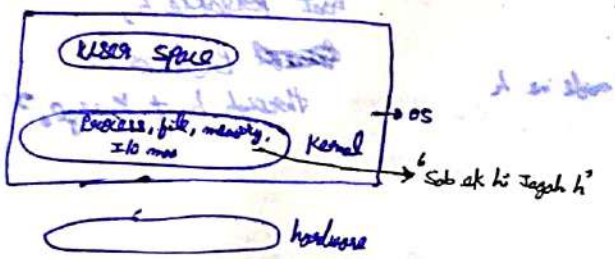
① Monolithic (4) → Isse sare Kaam ek [redacted] Yhi Kernel hi karega.

(e.g.) Linux, Unix, MS-DOS

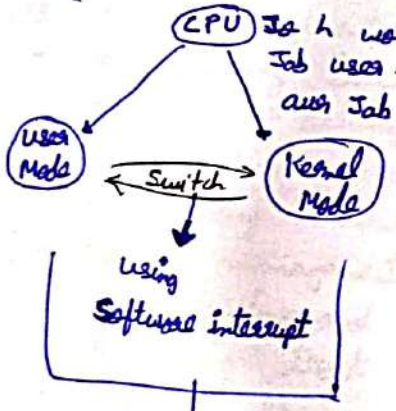
Advantage :
① Fast communication b/w diff component of kernel

Disadvantage :
① Bulky - kernel | ② Less Reliability. → agar ek gya to sab gya. (fat gya)

Sabse Purana ye.

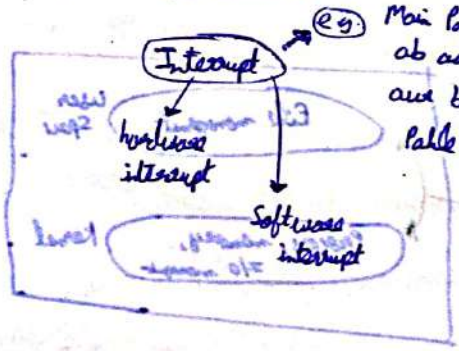



(Here module kernel me present h)
• Sab ek hi Jagah h, to baate/communication hogi wo efficient aur fast hogi.




Jo h wo
Jab user mode me hoga to wo wahi ka Kaam karega
aur Jab Kernel mode me hoga to wha ka karega

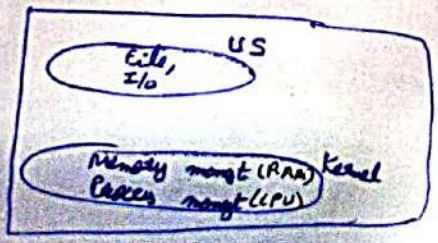
(e.g.) Jese Main ne CLI me ek code likhi (mkdir) to main to user mode me tha, ab code process hoke file management me gyi (Kernel me) to basically we are in Kernel Mode. Phir Job ho gya to finally we are in user mode.




(e.g.) Main padh rha hu, ab achank se koi aaya aur bola ya sab chhor pakke gane khelo.

② Micro Kernel

→ Major component/function jo h usi ko kernel me rakhte h baki ko user space me sambhal lete h.




Advantage :-
① Less bulky
② More reliable (Chije fatne ki chances thoda kam hogyi)
↓
Stable ✓

(e.g.) → • L4 linux
• Symbian OS

Disadvantage :-
① Performance low → (U.S. se kernel switch hone me time lagega)
← (me cost Aate h) ← (⇄) (user switch)

user-user switch
↑
Overhead aaega

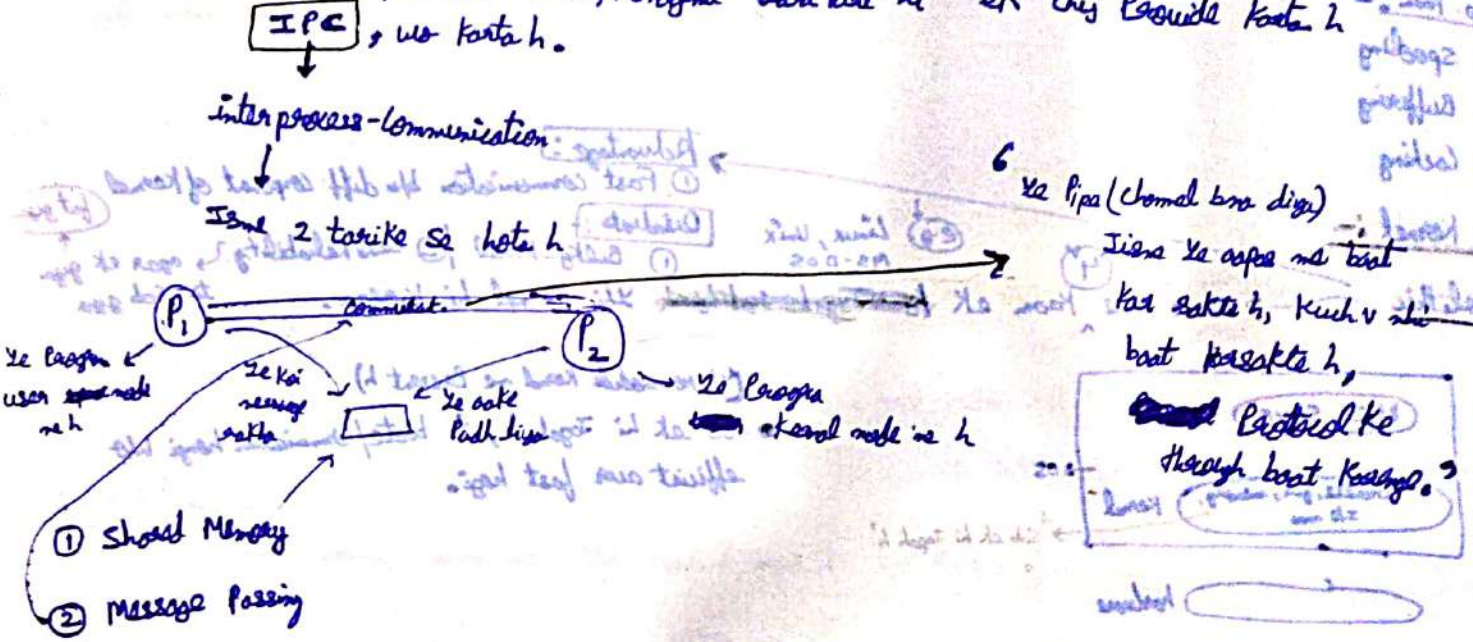
INTERVIEW Q → Micro-Kernel me user mode aur Kernel mode me Communication Kaise hota h?

→ OS ka process method/management wala part hi ek chij Provide karta h IPC, wo karta h.

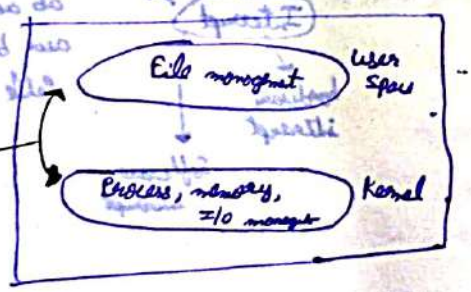
↓

inter process-communication

Isme 2 tarike se hota h




③ Hybrid Kernel → Combined approach of monolithic & micro-Kernel.




- Switching me thoda kam hoga
- speed increase hoga.

eg → • Mac OS
• Windows NT (7+)

2 Micro Kernel:
- Nano-Kernel → HW
- Exo-Kernel

Not much imp for interview.

(b)

## Components of OS

1. Kernel: A Kernel is that Part of the operating system which interacts directly with the hardware and Performs the most crucial tasks.
   a) Heart of OS / core component
   b) Verify first Part of OS to load on start-up.
2. User space: Where applications software runs, apps don't have Privileged access to the underlying hardware. It interacts with kernel.
   a) GUI
   b) CLI

- A Shell, also known as a command interpreter, is that Part of the operating system that receives commands from the users and gets them executed.

★ Function of Kernel :-

(i) Process Management :
- Scheduling Process and threads on the CPUs.
- Creating & deleting both user and system process.
- Suspending and resuming processes.
- Providing mechanisms for Process synchronization or Process communication.

(ii) Memory management :-
- Allocating and deallocating memory space as Per need.
- Keeping track of which Part of memory are currently being used and by which Process.

(iii) File management :-
- Creating and deleting files
- Creating and deleting diretories to organize files.
- Mapping files into secondary storage.
- Backup support onto a stable storage media.

(iv) I/O management :- To manage and control I/O operations and I/O devices
   a) Buffering (data copy between two devices), Catching and spooling.
      (i). Spooling
         1. Within different speed two Jobs
         2. eg → Print spooling and mail spooling
      (ii) Buffering
         1. Within one Job
         2. eg → Youtube video buffering
      (iii) Catching
         1. Memory catching, Web catching etc.

- Types of Kernels :-

1. Monolithic Kernel

a) All functions are in Kernel itself

b) Bulky in size

c) Memory required to run is high

d) Less reliable, one module crashes -> whole Kernel is down.

e) High Performance as communication is fast (Less user mode, Kernel mode overheads).

f) eg. → LINUX, Unix, MS-DOS.

2. Micro Kernel

a) only major functions are in Kernel

(i) Memory management

(ii) Process management

b) File management and IO management are in User-Space.

c) Smaller in size

d) More reliable

e) More stable

f) Performance is slow

g) overhead switching b/w user mode and Kernel mode.

h) eg → L4 Linux, Symbian OS, MINIX etc.

3. Hybrid Kernel :

a) Advantages of both worlds (File management in User space and rest in Kernel Space).

b) Combined approach

c) Speed and design of mono.

d) Modularity and stability of micro

e) eg → MacOS, Windows NT/7/10

f) IPC also happens but lesser overheads.

4. Nano/Exo Kernel

Q How will communication happen between user mode and kernel mode?

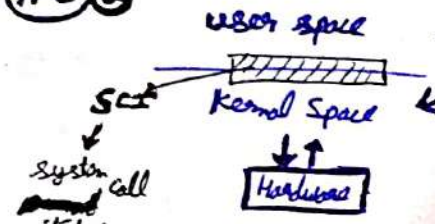
A → Inter Process Communication (IPC) ←

1. Two processes executing independently, having independent memory space (memory protection), but some may need to communicate to work.

2. Done by shared memory and message passing.

---

#5 @




Yha se wha hum → System call ki help se Jaate h.

GUI → New folder (internally yahi kam ho rha h)

CLI → and ≡ mkdir

↓

user-space mkdir lekr gya SCI ke pass, aur phir wo kernel se bolega isse related to kam h wo karke do,

Kernel me kuch 'C'-code ( ) likha hoga

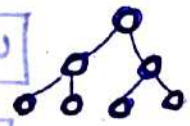
basically sari implementation 'C' me likha hota h.

Phir wo bna dega New folder

e.g. → (tree jesa) root node

New folder

* fork → hmesha child process bnata h



e.g., Main ne terminal khola to exe. run kari,

ab terminal → root node
exe run → child node

* chmod

↓

ye mode batata h file ka, ki readable h ya writable h

* chown → change ownership

e.g. ye kaun particular inodes access kar sakta h.

Using IPC

message pipe → pipe()

Shared mem. → shmget

(b)

# System Calls

Q) How do apps interact with kernel? → using System calls.

e.g → mkdir lab's

- Mkdir indirectly calls kernel and asks the file management module to create a new directory.
- Mkdir is just a wrapper of actual system calls.
- Mkdir interacts with kernel using system calls.

e.g → Creating a process

- user execute a process (user space)
- gets system call (US)
- execute system call to create a process (KS)
- Return to US.

★ Transitions from US to KS done by software interrupts.

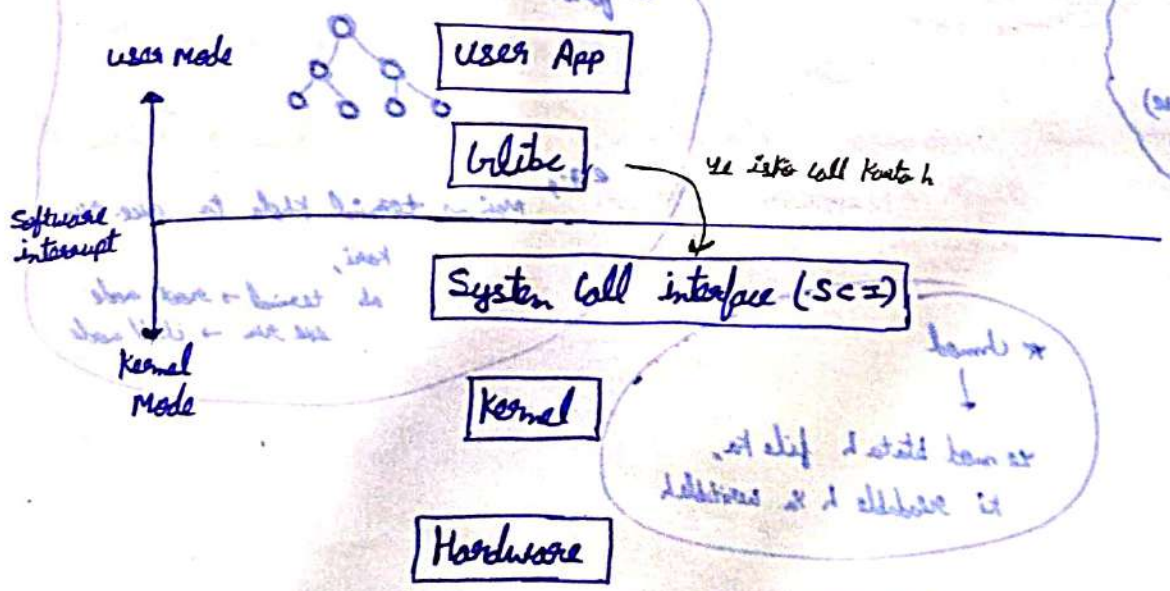
★ System calls are implemented in C

- A system call is a mechanism using which a user program can request a service from the kernel for which it does not have the permission to perform.

User programs typically do not have ~~[illegible]~~ permission to perform operations like accessing I/O devices and communicating other programs.

- System calls are the only way through which a process can go into kernel mode from user mode.

★



★ Types of System calls :- (PiDFC)

1) 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

2) File management
   - ⓐ create file, delete file
   - ⓑ open, close
   - ⓒ read, write, reposition
   - ⓓ get file attributes, set file attributes

3) Device management
   - ⓐ request device, release device
   - ⓑ read, write, reposition
   - ⓒ get device attributes, set device attributes
   - ⓓ logically attach or detach devices

4) Information maintenance
   - ⓐ get time or date, set time or date
   - ⓑ get system data, set system data
   - ⓒ get Process, file, or device attributes
   - ⓓ set Process, file, or device attributes

5) Communication Management
   - ⓐ create, delete communication connection
   - ⓑ send, recieve message
   - ⓒ transfer status information
   - ⓓ attach or detach remote devices.

## Examples of Windows & Unix system calls:-

| Category | Windows | Unix |
|---|---|---|
| • Process control | CreateProcess()<br>Exit Process()<br>Wait For Single Object() | fork()<br>exit()<br>wait() |
| • File Management | CreateFile()<br>Read File()<br>Write File()<br>CloseHandle()<br>SetFileSecurity()<br>InitializeSecurityDescriptor()<br>SetSecurityDescriptorGroup() | open()<br>read()<br>write()<br>close()<br>chmod()<br>umask()<br>chown() |
| • Device Management | SetConsoleMode()<br>ReadConsole()<br>WriteConsole() | ioctl()<br>read()<br>write() |
| • Information Management | GetCurrentProcessID()<br>Set Timer()<br>Sleep() | getpid()<br>alarm()<br>sleep() |
| • Communication | CreatePipe()<br>CreateFileMapping()<br>MapViewOfFile() | pipe()<br>shmget()<br>mmap() |

#6 ② → Interesting Topic :)

What happs when u turn on your computer?

5 Steps

① Power ON

ON → Power Supply

electricity ka aai → → Mother board
→ Harddisk/ storage

② CPU loads BIOS or UEFI (upgraded version) → Unified extensible firmware interface

BIOS ↓ basic i/o system

chip

isme sare program likhe h ki kese sabko initialize karna h

① CPU initialize (sabse pahle yahi hoga)
② goes to a chip BIOS

③ BIOS or UEFI run test & init hardware

ⓐ Loads some settings from a memory area

(I) backed by CMOS Battery → (ye bahut imp h) → (ha Electricity de sake ha phir v ye imp h battery)

ⓑ Program loads with settings

POST → Power on selftest

→ Yha le dekhta h sab essential hardware h ya nhi e.g. RAM/ROM. (nhi hoga to error dega → computer ON nhi hoga)

④ BIOS or UEFI, hands off to Boot device

→ disc (HDD/SSD)
→ CD
→ USB device

Boot loader

↓ Program execute → ON (actual OS)

ab mujhe actual program chalaye jo 'OS' ko on karega

Boot loader kya kya karta h?

① MBR → Master Boot Record

→ ye basically disk ki 0th index le kata h

To ab simple anish, BIOS → old system me hota tha kyuki wo MBR ka use karta tha

aur UEFI → EFI ka use karta h :)

② EFI → Isme hum Jese disc ki → C-drive, D-drive

partition ke andar boot loader h,

Partition hi bna di pahle Jsa 0th index le ski.

⑤ ab Ha le BIOS bolega nera Kam Khatam, ab Boot loader teri Jimadari,

Sab dog alag Kaage System aurdin

Windows → bootmgr.exe
Mac → boot.efi
Linux → Grub

actual OS ab on hoga

⑥ What happens when you turn on your computer?

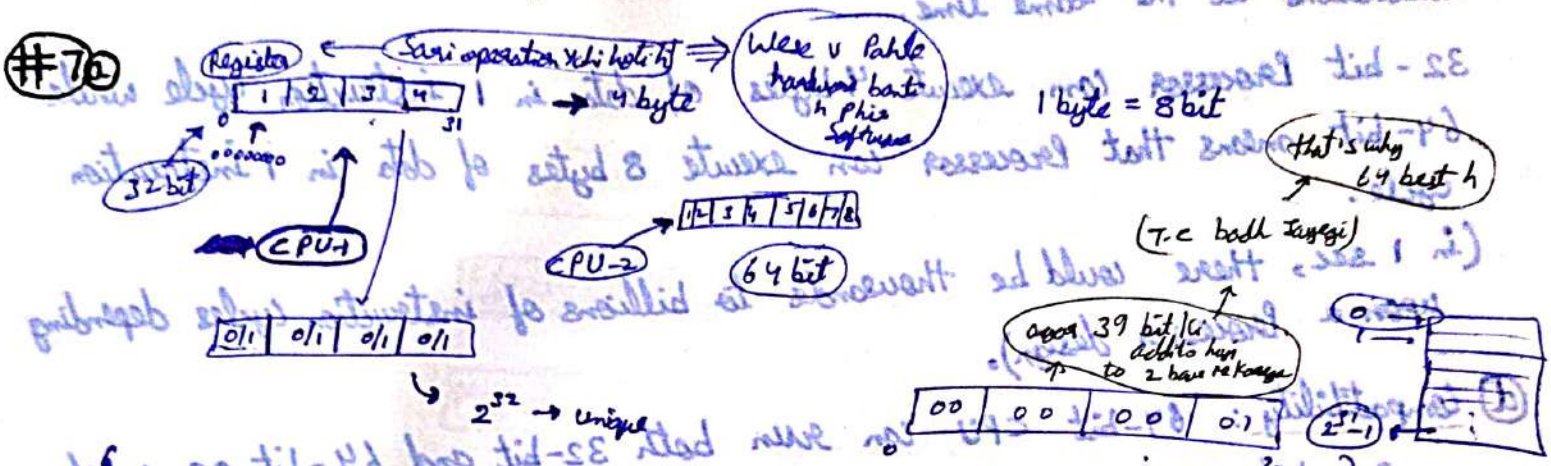
1. PC on
2. CPU initializes itself and looks for a firmware Program (BIOS) stored in BIOS chip (basic input-output system chip is a ROM chip found on mother board that allows to access & setup computer system at most basic level)
   ⓐ In modern PCs, CPU looks UEFI (Unified extensible firmware interface)
3. Bios loads configuration settings
4. CPU runs the BIOS which tests and initializes system hardware. If something is not appropriate (like missing RAM) error is thrown and boot Process is stopped.

This is called POST (Power on self-test) Process.
(UEFI can do a lot more than Just initialize hardware; it's really a tiny operating system. For example, Intel CPUs have the <u>Intel Managment Engine</u>. This provides a variety of features, including Supporting Intel's active Management Technology, which allows for remote management of business PCs.)

5. BIOS will handoff responsibility for booting your PC to your OS's bootloader.
   ⓐ BIOS looked at the MBR (master boot record), a special boot sector at the beginning of a disk. The MBR contains code that loads the rest of the operating system, known as a "bootloader". The BIOS executes the bootloader, which takes it from there and begins booting the actual operating system - Windows or Linux, for example,
   In other words, the BIOS or UEFI examines a ~~storing~~ storage device on your system to look for a small Program, either in the MBR or on an EFI system Partition, and runs it.

6. The bootloader is a small program that has the large task of booting the rest of the operating system. Windows uses a bootloader named Windows Boot Manager (Bootmgr.exe), most Linux systems use GRUB, and Macs use something called boot.efi.

#7(a)




① addressable space
aur
② Performance kese
64 bit OS ki better h
32 bit OS se
} → all about this

#7(b) 32-Bit vs 64-Bit OS

1) A 32 bit OS has 32-bit registers, and it can access 2^32 unique memory addresses. i.e, 4GB of Physical memory.
2) A 64-bit OS has 64-bit registers, and it can access 2^64 unique memory addresses. i.e, 17,179,869,184 GB of Physical memory.
3) 32-bit CPU architecture can Process 32 bits of data and information.
4) 64-bit CPU architecture can Process 64 bits of data and information.
5) Advantages of 64-bit over the 32-bit operating system:
   ⓐ Addressable Memory: 32-bit CPU → $2^{32}$ memory addresses, 64-bit CPU → $2^{64}$ memory addresses.
   ⓑ Resource usage: Installing more RAM on a system with a 32-bit OS doesn't impact Performance. However, upgrade that system with excess RAM to the 64-bit version of windows, and you will notice a difference.

(c) **Performance**: All calculations take place in the registers. When you are performing math in your code, operands are loaded from memory into registers. So, having larger registers allow you to perform larger calculations at the same time.

32-bit processor can execute 4 bytes of data in 1 instruction cycle while 64-bit means that processor can execute 8 bytes of data in 1 instruction cycle.

(in 1 sec, there could be thousands to billions of instruction cycles depending upon a processor design).

(d) **Compatibility**: 64-bit CPU can run both 32-bit and 64-bit OS. While 32-bit CPU can only run 32-bit OS.

(e) **Better Graphics Performance**: 8-bytes graphics calculations make graphics-intensive apps run faster.

#8 (2)

- (CPU se sabse pass [Register] hota h) → fastest → ab ye sabse costlier hota h (kyuki ye transistor se bana hua hota h) → smallest size

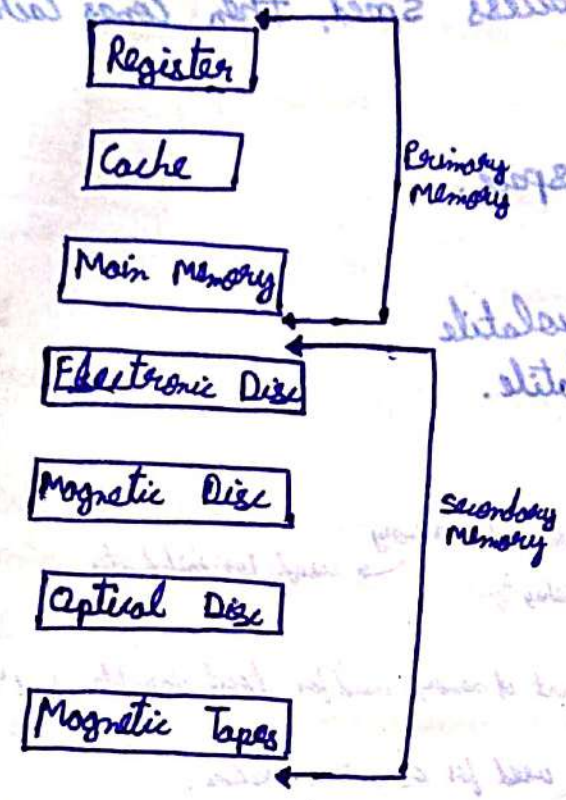
(OS → socho your ye baar use ho rahi h to isko uthake cache me rakh dete hai)

- (Secondary memory me program/data hoti h ye binary memory me hoti h process hoti h)
  - ↓ ye sab to low cost me mil jayegi
  - large size → disc → 1TB wagaira

- Volatility → computer off karne ke baad process wali jati h (Vanish off)
  - Binary me → flush/Vanish ho jati h
  - secondary me wali hoti h (ye rahta h) → saved right data.

#8(b) Storage Devices Basics

- What are the different memory present in the computer system?




1) Register: Smallest unit of storage. It is a part of CPU itself. A register may hold an instruction, a storage address, or any data (such as bit sequence or individual characters).
Registers are a type of computer memory used to quickly accept, store, and transfer data and instructions that are being used immediately by the CPU.

2) Cache: Additional memory system that temporarily stores frequently used instructions and data for quicker processing by the CPU.

3) Main memory: RAM

4) Secondary Memory: storage media, on which computer can store data & programs.

COMPARISON

1) Cost
a) Primary storages are costly
b) Registers are most expensive due to expensive semiconductors & labour.
c) Secondary storages are cheaper than primary.

2) Access Speed:

a) Primary has higher access speed than secondary memory.

b) Registers has higher access speed, then comes cache, the main memory.

3) Storage Size:

a) Secondary has more space.

4) Volatility:

a) Primary memory is volatile

b) secondary is non-volatile.

#9 (a)

Steps

① Load the Program & static data into memory → used for initialization

char * name = "Lakshay";

② Allocate Runtime stack :→ Part of memory used for local variable, funct^n argument & return value

③ Allocate heap :→ Part of memory used for dynamic allocation.

④ I/P, O/P, error

• stack overflow

• out of memory

• attribute of Process

Process Table

| | |
|---|---|
| 2 | $P_2$ |
| 3 | $P_3$ |

PS

#9 (b) Introduction to Process

1) What is a Program? Compiled code, that is ready to execute.

2) What is a Process? Program under execution

3) How OS creates a Process? Converting Program into a Process.

STEPS:

a) Load the Programs and static data into memory. → Initialization data

b) Allocate runtime stack → Pable wala data (OS Gaddha h Pable SO) → int num ← integer use kro

c) Heap memory allocation → funct^n call Ke time change

d) I/O tasks

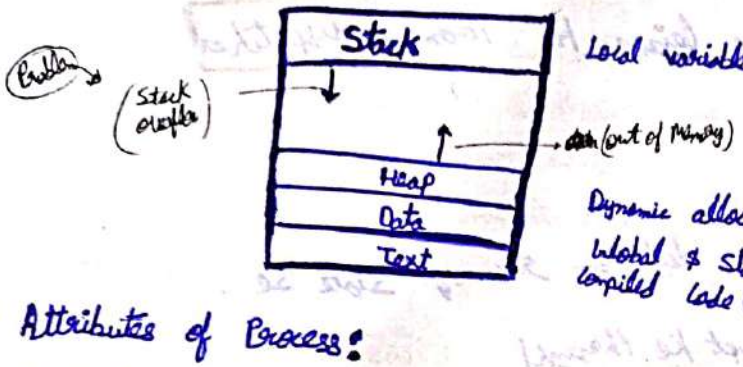
e) OS handoffs control to main().

maybe aur data change hoga (dynamic memory)

**How OS create a Process**

Program se Process tk ka Safar?

4) Architecture of Process:




5) Attributes of Process:

a) Feature that allows identifying a Process uniquely

b) Process table

(i) All Processes are being tracked by OS using a table like data structure

(ii) Each entry in that table is Process control block (PCB).

c) PCB: Stores info/ attributes of a Process.

(i) Data structure used for each Process, that stores information of a Process such as Process id, Program counter, Process state, Priority etc.

6) PCB structure:

| | |
|---|---|
| Process ID | unique identifier |
| Program counter (PC) | Next instruction address of the Program |
| Process State | stores Process state |
| Priority | Based on Priority a Process gets CPU time |
| Registers | |
| List of open files | |
| List of open devices | |

- Registers in the PCB, it is a data structure. When a Processes is running and it's time slice expires, the current value of Process specific registers would be stored in the PCB and the Process would be swapped out. When the Process is schedule to be run, the register values is read from the PCB and written to the CPU registers. This is the main Purpose of the registers in the PCB.

#10 ⓐ

- Jitni Kaam/Job wait Kar rahi h run hone ke liye wo Sabka Matlab

Ready State → Running State me lejane Ka Kaam dispatcher Karta h

- agar Time Quantum Khatam hogyi to wo dubara running State se ready state me aajayegi (Interrupt ke through)

Degree of multiprogramming Yahi manage Karta h ⇒ Yahi decide Karta h memory se ready queue me kitne process dalne h

- Job Scheduler (Long Term Scheduler) → based on priority
- CPU Scheduler (Short Term Scheduler) → CPU Jaldi Kam Karta (agar koi P1 I/O (Ka run) to main Jaldi se P2 ke Kam Karunga)

(ye aalsi h Lye 5 Job daba Phir those (ha bood ke 5 Job aur daba ready queue)

## #10 ⓑ Process States / Process Queues

1) Process States: As Process executes it changes state. Each Process may be in one of the following states.

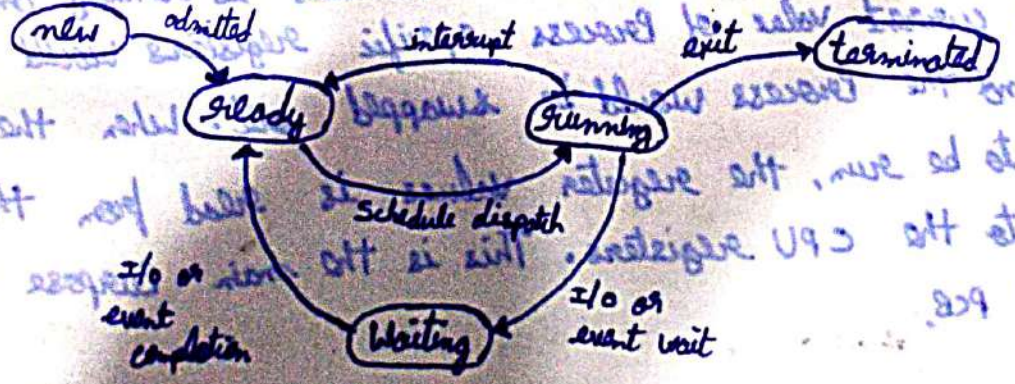
ⓐ New: OS is about to pick the Program and convert it into Process. OR the Process is being created.

ⓑ Run: Instructions are being executed; CPU is allocated.

ⓒ Waiting: Waiting for IO (Khatam hojaye I/O tab dubara Run state me aau main)

ⓓ Ready: The Process is in memory, waiting to be assigned to a Process

ⓔ Terminated: The Process has finished execution. PCB entry removed from Process table.



2) Process Queues:

ⓐ Job Queue:

(i) Processes in new state

(ii) Present in secondary memory

(iii) Job Scheduler (Long term Scheduler (LTS)) picks process from the pool and loads them into memory for execution.

ⓑ Ready Queue:

(i) Processes in Ready state

(ii) Present in main memory

(iii) CPU scheduler (Short-term scheduler) picks process from ready queue and dispatch it to CPU.

ⓒ Waiting Queue:

(i) Processes in wait state.

3) Degree of multi-programming: The number of processes in the memory.

ⓐ LTS controls degree of multi-programming.

4) Dispatcher: The module of OS that gives control of CPU to a process selected by STS.

#11 ⓐ

- "Ready Queue me kafi processes aagyi ab kuch ko CPU ke help se bahar phek dunga (Secondary storage me) Phir jab jagah khali hoga tab wapas Ready Queue me dal dunga."

↓

Ye MTS karta h

↓

STS "ye bich me rakhta h" LTS

PCB →

| PID |
|---|
| PC |
| register |
| state |
| FDs |

"Context Switching → Kernel karta h."

Init → 1st process of LINUX (sabse pehle yahi h) → (grand-grand father)

- $P_1$ running —fork()→ $P_2$ child running

$P_1$ → execute → terminate

ab OS $P_2$ ko init ko ~~sop~~ sop dega.

- Parent jab child bnati h to tab tak wait karti jab tak child apni kaam khatam na karle.

# #11 (b) Swapping / Context - Switching / Orphan Process / Zombie Process

## 1) Swapping

(a) Time-sharing system may have medium term scheduler (MTS).

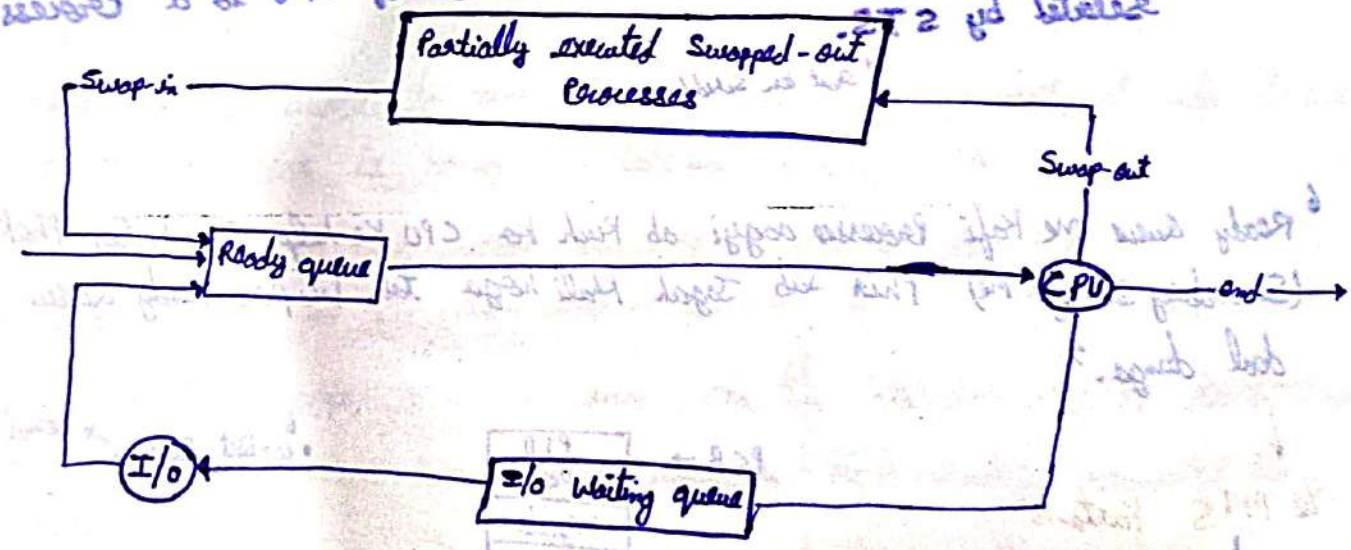
(b) Remove processes from memory to reduce degree of multi-programming

(c) These removed processes can be reintroduced into memory, and its execution can be continued where it left off. This is called Swapping.

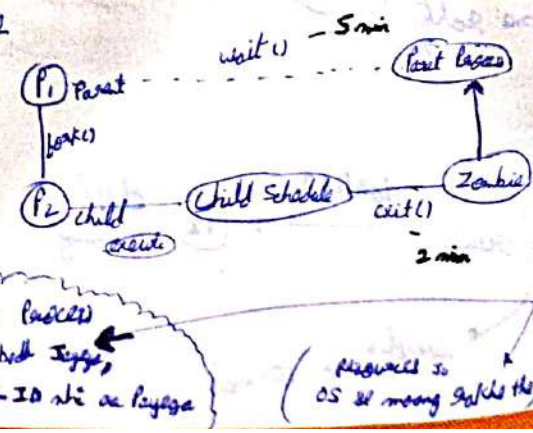
(d) Swap-out and Swap-in is done by MTS

(e) Swapping is necessary to improve process mix or because a change in memory requirements has overcommitted available memory, requiring memory to be freed up.

(f) Swapping is a mechanism in which a process can be swapped temporarily out of main memory (or more) to secondary storage (disk) and make that memory available to the other processes. At some later time, the system swaps back the process from the secondary storage to main memory.




(4a) Zombie




• (code me $ lagao so we background me chalti h)

2) Context - Switching

ⓐ Switching the CPU to another Process requires performing a state save of the current Process and a state restore of a different Process.

ⓑ When this occurs, the Kernel saves the context of the old Process in its PCB and loads the saved context of the new Process scheduled to run.

ⓒ It is Pure overhead, because the system does no useful work while switching.

ⓓ Speed varies from machine to machine, depending on the memory speed, the number of registers that must be copied etc.

3) Orphan Process

ⓐ The Process whose Parent Process has been terminated and it is still running.

ⓑ Orphan Processes are adopted by init Process.

ⓒ Init is the first Process of OS.

4) Zombie Process / Default Process

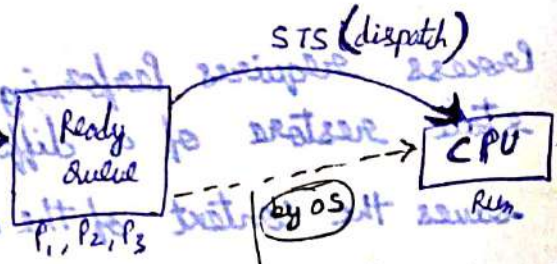
ⓐ A Zombie Process is a Process whose execution is completed but it still has an entry in the Process table.

ⓑ Zombie Processes usually occur for child Processes, as the Parent Process still needs to read its child's exit status. Once this is done using the wait system call, the zombie Process is eliminated from the Process table. This is known as reaping the zombie Process.

ⓒ It is because Parent Process may call wait() on child Process for a longer time duration and child Process get terminated much earlier.

ⓓ As entry in the Process table can only be removed, after the Parent Process reads the exit status of child Process. Hence, the child Process remains a zombie till it is removed from the Process table.

#12 (a)




Selection of Process from ready Queue to CPU done by CPU Scheduler (OS)

- Non-Preemptive → In one flow → Yha pe 'Starvation' ki Problem Jyada hai
- Preemptive → Time Quantum → Yha pe Jyada CPU utilization
  → Jyada overhead hoga.

- Goal of CPU Scheduling :-
  1. Max. CPU utilization
  2. Min TAT
  3. Min WT
  4. Min Response time
  5. Max. Throughput

#12 (b) <u>Intro to Process Scheduling | FCFS | Convoy Effect</u>

1. Process Scheduling
   - (a) Basis of Multi-Programming OS
   - (b) By switching the CPU among Processes, the OS can make the Computer more Productive
   - (c) Many Processes are kept in memory at a time, when a Process must wait or time quantum expires, the OS taken the CPU away from that Process, and gives the CPU to another Process and this Pattern continues

2. CPU Scheduler

ⓐ Whenever the CPU become ideal, OS must select one ~~process~~ Process from the ready queue to be executed

ⓑ Done by STS.

3. Non - Preemptive Scheduling

ⓐ Once CPU has been allocated to a Process, the Process keeps the CPU until it releases CPU either by terminating or by switching to wait state.

ⓑ Starvation, as a Process with long burst time may starve less burst time Process.

ⓒ Low CPU utilization.

4. Preemptive Scheduling

ⓐ CPU is taken away from a Process after time quantum expires along with terminating or switching to wait - state.

ⓑ Less starvation

ⓒ High CPU utilization

5. Goals of CPU scheduling

ⓐ Maximum CPU utilization

ⓑ Minimum Turnaround Time (TAT)

ⓒ Min - Wait - Time

ⓓ Min - response Time

ⓔ Max. throughput of system

6. Throughput: No. of Processes completed per unit time.

7. Arrival time (AT): Time when Process is arrived at the ready queue.

8. Burst time (BT): The time required by ~~the~~ Process for its execution.

9. Turnaround time (TAT): Time taken from first time Process enters ready state till the terminates (CT - AT).

10. Wait time (WT): Time Process spends waiting for CPU. (WT = TAT - BT)

11. Response time: Time duration between Process getting into ready queue and Process getting CPU first time.

12. Completion Time (CT): Time taken till process gets terminated.

13. FCFS (First come - first serve)

ⓐ whichever process comes first in the ready queue will be given CPU first.

ⓑ In this, if one Process has longer BT. It will have major effect on average WT of diff Processes, called <u>convey effect.</u>

ⓒ Convey effect is a situation where many Processes, who need to use a resource for a short time, are blocked by one Process holding that resources for a long time.

i.e → This cause Poor resource managment.

#13 ⓐ

Solve all scheduling → Q

- Preemptive → SJF me convey effect aayega hi nhi
- Biggest Drawback of Priority Scheduling → <u>Indefinite Waiting</u> or <u>extreme Starvation</u> (of low Priority)

Jiski Jyada burst time h, uski 1st Priority ho Jaye tab ☹

aur hoa koi aate Jate Jiski high Priority ho

Solution?

↓

$\boxed{\text{ageing}}$ → gradually increasing Priority of lowest Job

↳ like → hoa 15 second me lowest Priority Job ko (+1) Karta Jaunga.

#13 (b)

# CPU scheduling | SJF | Priority | RR

1. Shortest Job First (SJF) [Non-Preemptive]:
   - (a) Process with least BT will be dispatched to CPU first.
   - (b) Must do estimation for BT for each Process in ready queue beforehand, Correct estimation of BT is an impossible task (ideally)
   - (c) Run lowest time Process for all time then, choose Job having lowest BT at that instance.
   - (d) This will suffer from convoy effect as if the very first Process which came is Ready state is having a large BT.
   - (e) Process starvation might happen.
   - (f) Criteria for SJF algos, AT+BT

2. SJF (Preemptive):
   - (a) Less starvation
   - (b) No convoy effect
   - (c) Gives average WT less for a given set of Processes as scheduling short Job before a long one decreases the WT of short Job more than it increases the WT of the long Process.

3. Priority Scheduling [Non-Preemptive]:
   - (a) Priority is assigned to a Process when it is created.
   - (b) SJF is a special case of general Priority scheduling with Priority inversely Proportional to BT.

4. Priority Scheduling [Preemptive]:
   - (a) Current RUN state Job will be ~~preemptive~~ preempted if next Job has higher Priority.
   - (b) May cause indefinite Waiting (Starvation) for lower Priority Jobs. (Possibility is they won't get executed ever). (True for both preemptive and non-Preemptive version)
     - (i) Solution: Ageing is the solution
     - (ii) Gradually increase Priority of Process that wait so long. Eg. increase Priority by 1 every 15 minutes.

5. Round Robin Scheduling (RR):

ⓐ Most Popular

ⓑ Like FCFS but preemptive

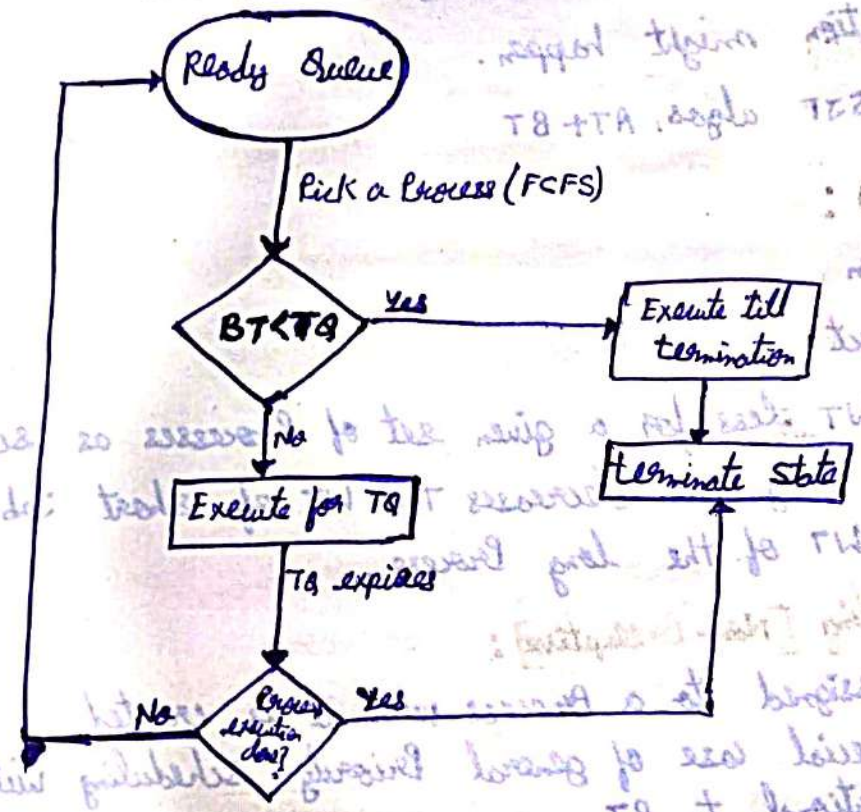
ⓒ Designed for time sharing systems.

ⓓ Criteria : AT + time ~~quad~~ quantum (TQ), Doesn't depend on BT.

ⓔ No Process is going to wait forever, hence very low starvation [No convoy effect]

ⓕ Easy to implement

ⓖ If TQ is small, more will be the context switch (more overhead)




#14 ⓐ

- hum generally use Karte h → (MLFQ)
- Multi-level Queue Scheduling Karte h :-
  1. System Process :→ Created by OS (Jese 'init' - OS khud bnata h)
  2. Interactive Process → user input required (fore ground)
  3. ~~Bat~~ Batch Process → no i/p (back-ground) (no sunoga)

- Design of MLFQ :-
  - ⓐ No. of Queue
  - ⓑ Scheduling algo in each queue
  - ⓒ Method to upgrade a Process to higher Queue
  - ⓓ demote a Process to ~~any~~ Lower Priority
  - ⓔ Process $P_1$ → which queue will ~~be~~ it be Pushed

#14 (b)

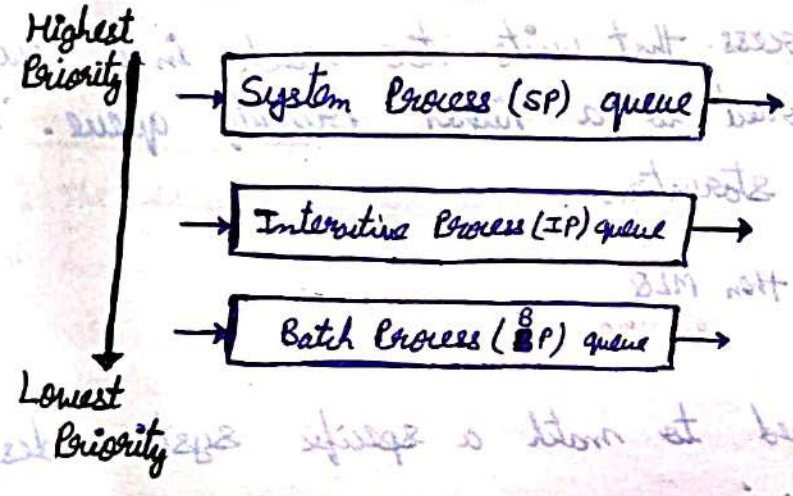
# MLQ | MLFQ

1. Multi-level queue scheduling (MLQ):

(a) Ready queue is divided into multiple queue depending upon priority.

(b) A process is permanently assigned to one of the queue (inflexible) based on some property of process, memory, size, process priority or process type.

(c) Each queue has its own scheduling algorithm. eg → SP → RR, IP → RR and BP → FCFS.




(d) System Process: created by OS (Highest Priority)
Interactive Process (Foreground Process): Needs user input (I/O).
Batch Process (Background Process): Runs silently, no user input required.

(e) Scheduling among different sub-queues is implemented as fixed priority preemptive scheduling. Eg → foreground queue has absolute priority over background queue.

(f) If an interactive process comes and batch process is currently executing. Then, batch process will be preemptive.

(g) Problem: only after completion of all the processes from the top-level ready queue, the further level ready queues will be scheduled.
This come starvation for lower priority process.

(h) Convoy effect is present.

## 2. Multi-level feedback queue scheduling (MLFQ):

ⓐ Multiple sub-queues are ~~different~~ present.

ⓑ Allows the process to move between queues. The idea is to separate processes according to the characteristics of their BT.

If a process uses too much CPU time, it will be moved to lower priority queue. This scheme leaves I/O bound and interactive processes in the higher-priority queue.

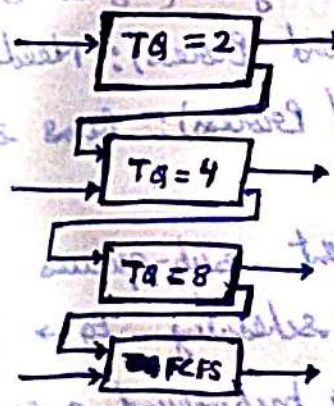
In addition, a process that waits too much in a lower-priority queue may be moved to a higher priority queue. This form of <u>ageing</u> prevents starvation.

ⓒ Less starvation than MLQ

ⓓ It is flexible

ⓔ Can be configured to match a specific system design requirement

Sample MLFQ design.

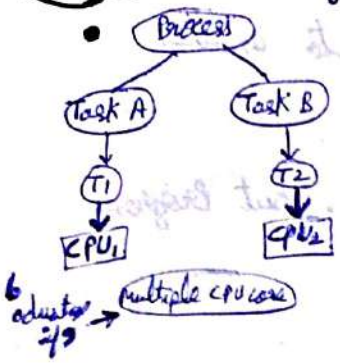


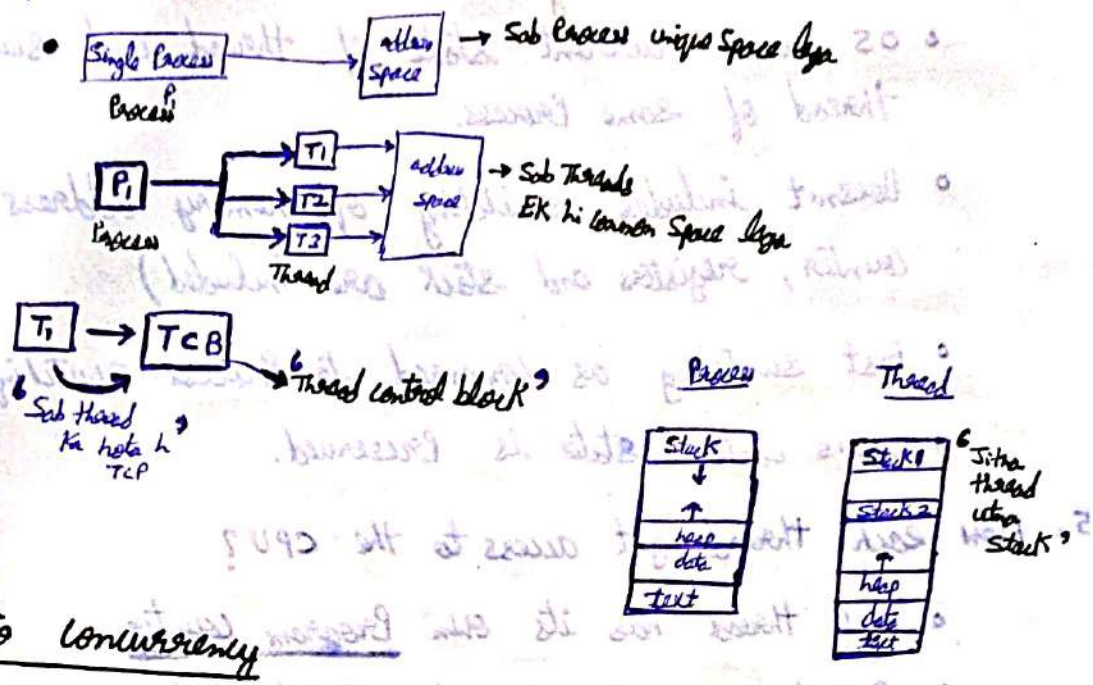

## 3. Comparison:

| | FCFS | SJF | PSJF | Priority | P-Priority | RR | MLQ | MLFQ |
|---|---|---|---|---|---|---|---|---|
| Design | Simple | Complex | Complex | Complex | Complex | Simple | Complex | Complex |
| Preemptive | No | No | Yes | No | Yes | Yes | Yes | Yes |
| Convoy effect | Yes | Yes | No | Yes | Yes | No | Yes | Yes |
| Overhead | No | No | Yes | No | Yes | Yes | Yes | Yes |

starvation

#15 (a) • Concurrency → multiple instruction at the same time




## #15 (b) Introduction to concurrency

1. Concurrency is the execution of the multiple instruction sequences at the same time. It happens in the operating system when there are several process threads running in parallel.
2. Thread:
   - Single sequence stream within a process
   - An independent path of execution in a process
   - Light weight process.
   - used to achieve parallelism by dividing a process's tasks which are independent path of execution.
   - Eg Multiple tabs in browser, text editor (when you are typing in an editor, spell checking, formatting of text and saving the text are done concurrently by multiple threads.)
3. Thread Scheduling: Threads are scheduled for execution based on their priority. Even though threads are executing within the runtime, all threads are assigned processor time slices by the operating system.

4. Threads context Switching

- OS saves current state of thread and switches to another thread of same Process.
- Doesn't includes switching of memory address space. (But Program counter, registers and stack are included)
- Fast switching as compared to Process switching.
- CPU's cache state is Preserved.

5. How each thread get access to the CPU?

- Each thread has its own <u>Program counter</u>
- Depending upon the thread scheduling algorithm, OS schedule these threads.
- OS will fetch instructions corresponding to PC of that thread and execute instruction.

6. I/O or TQ, based context switching is done here as well.

- We have TCB (Thread Control block) like PCB for state storage management while Performing context Switching.

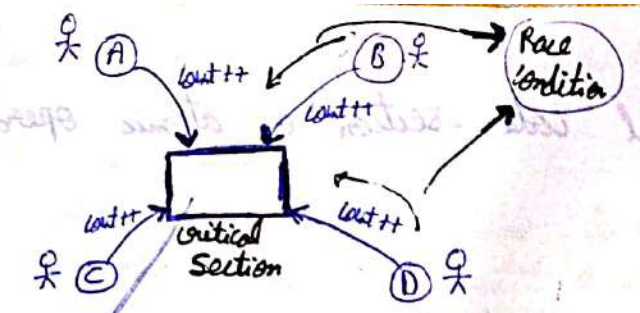
7. Will Single CPU system would gain by multi-threading technique?

- Never
- As two threads have to context switch for that single CPU.
- This won't give any gain.

8. Benefits of Multi-threading

- Responsiveness ↑
- Resource sharing : Efficient resource sharing
- Economy : It is more economical to create and context switch threads.
  1. Also, allocating memory and resources for Process creation is costly, so better to divide tasks into threads of same Process.
- Threads allow utilization of multiprocessor architectures to a great scale and efficiency.

#16(a)




"Same data-base ko alter kar sakta h kafi thread"

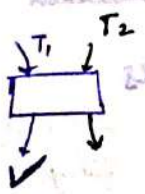
"Jab thread-switching ho sakta to temp = 10 that (+1) huya hi ki dusra ko rota mil gya ab To (14) aaya tha wo (11) hi aaya"

Solution

① Atomic → ek hi CPU me hoyega

② Mutual exclusive

LOCKS → Kisi v language me hota h ye → "Ye lock karega Pehla kaam karke Jayega tabhi dusra aayega"




EK aaye kaam karke Jaye Phir dusra aaye

Entry Section
Critical Sect"
Exit Section

③ Semaphores → Multiple Instances of Resources → to ab main 2-3 thread ko ek saath bhej sakta h resource ke liye

• Sol^n of C.S should have 3 conditions?

① Mutual exclusive

② Progress → Koi fix na ho kisi v thread ko rota mile

③ Bounded Waiting → No Infinite waiting (agla wala thread aa hi jayega thoda dera)

## #16(b) Critical Section Problem and How to address it

1. Process synchronization techniques play a key role in maintaining the consistency of shared data.

2. Critical Section (CS)

   ⓐ The critical section refers to the segment of code where processes/threads access shared resources, such as common variables and files, and perform write operations on them. Since processes/threads execute concurrently, any process can be interrupted mid-execution.

3. Major Thread scheduling issue

   ⓐ Race condition

   ① A race condition occurs when two or more threads can access shared data and they try to change it at the same time. Because the thread scheduling algorithm can swap between threads at any time, you don't know the order in which the threads will attempt to access the shared data. Therefore, the result of the change in data is dependent on the thread scheduling algorithm, i.e. both threads are "racing" to access/change the data.

4. Solution to Race condition

ⓐ Atomic operations: Make critical code section on atomic operation i.e. Executed in one CPU cycle.

ⓑ Mutual Exclusion using locks

ⓒ Semaphores

5. Can we use a simple flag variable to solve the Problem of race condition?

→ Mutual exclusion ✓ Progress ✗

• No [because can't satisfy Progress]

→ turn = 1 → T2 ko Mauka milega

× (Kai v Jaye) → Condition fulfil nahi hoti

```
(T1)                        (T2)
while (1) {                 while (1) {
    while (turn == 0);          while (turn == 1);
    C.S                         C.S
    turn = 1                    turn = 0
    R.S                         R.S
```

→ It satisfies 1st condition i.e. Mutual exclusion but not second → i.e. Progress

modify

6. Peterson's solution can be used to avoid race condition but holds good for only 2 process/threads.

Mutual Excl ✓ Progress ✓

→ Code on video → It Solves both conditions

7. Mutex/Locks

ⓐ Locks can be used to implement mutual exclusion and avoid race condition by allowing only one thread/Process to access critical section

ⓑ Disadvantages:

(i) Contention: One thread has acquired the lock, other threads will be busy waiting, what if thread that had acquired the lock dies, then all other threads will be in infinite waiting.

(ii) Deadlocks

(iii) Debugging

(iv) Starvation of high Priority threads (ab sabko low Priority wale ne Jagah deko bathna ha ☹)

#17 ⓐ

• Deadlock → $P_1$ ← $R_1$, $P_2$ → $R_2$

$P_1$ → mujhe $R_2$ ki chahiye tabhi chalegi

$R_1$ → mujhe $R_1$ ki chahiye tabhi chalegi

• lock acha tarika h, lekin '$P_2$' ko wait (Pta nhi kab)

Kaise pata sha h, isse acha method h → Conditional variable → Wait, Signal

```
T1 → condit.wait
T2 → C.S → Signal()
Notify
```

$T_2 - T_1$ (acha chalega)

# #11 (b) Conditional Variable and semaphores for Threads Synchronization

## 1. Conditional Variable

(a) The conditional variable is a synchronization primitive that lets the thread wait until a certain condition ~~occurs~~ occurs.

(b) Works with a lock

(c) Thread can enter a wait state only when it has acquired a lock. When a thread enters the wait state, it will release the lock and wait until another thread notifies that the event has occurred. Once the waiting thread enters the running state, it again acquires the lock immediately and starts executing.

(d) Why to use conditional variables?

(i) To avoid busy waiting.

(e) Contention is not here

## 2. Semaphores

(a) Synchronization method

(b) An integer that is equal to number of resources

(c) Multiple threads can go and execute CS concurrently.

(d) Allows multiple program threads to access the finite instance of resources where mutex allows multiple threads to access a single shared resource one at a time.

(e) Binary semaphore: value can be 0 or 1.

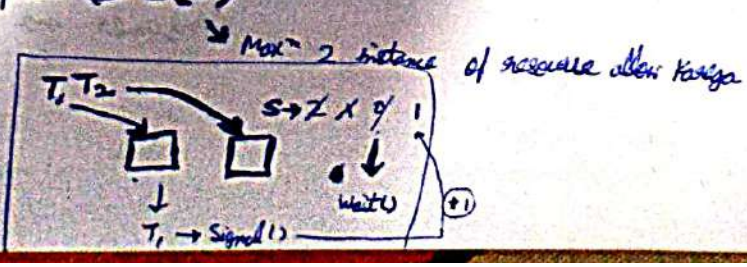
(i) Aka, mutex locks

(f) Counting semaphore

(i) Can range over an unrestricted domain

(ii) Can be used to control access to a given resource consisting of a finite number of instances.

(eg) Semaphore S(2)



(a) To overcome the need for busy waiting, we can modify the definition of the wait() and signal() semaphore operations. When a process ~~execution~~ executes the wait() operation and finds that the semaphore value is not positive, it must wait. However, rather than engaging in busy waiting, the process can block itself. The block-operation places a process ~~[illegible]~~ into a waiting queue associated with the semaphore, and the state of the process is switched to the waiting state. Then control is transferred to the CPU scheduler, which selects another process to executes.

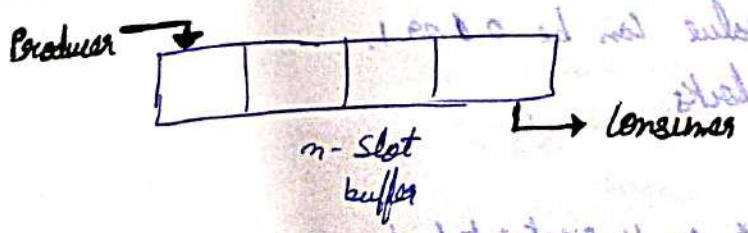
(b) A process that is blocked, waiting on a semaphore S, should be restarted when some other process executes a signal() operation. The process is restarted by a wakeup() operation, which changes the process from the waiting state to the ready state. The process is then placed in the ready queue.

#18

## Producer-Consumer Problem (Bonded Buffer Problem)

(1) Producer thread

(2) Consumer thread




- Jab Producer data tb consumer uthane na aaJaye Jab (OR) consumer uthane aaye tb us waqt Producer data na aajaye.
- Jab data full ho tab Producer aake usna na data Kuch
- Jab Khali ho buffer tb consumer na aaye uthane.

Ans → We use Semaphores

① m, mutex → Binary semaphore used to acquire lock on buffer

② empty → a counting semaphore, initial value is n
↳ tracks empty slot

③ full → tracks filled slot, initial value is 0

**Producer**

```
do {
  wait(empty);   // wait until, empty > 0, then empty → value--
  wait(mutex);   // C.S, add data to buffer   (-1)
  // C.S
  signal(mutex);   (+1)
  signal(full);   // increment full → value)   (+1)
} while(1)
```

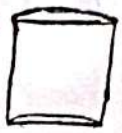
**Consumer**

```
do {
  wait(full);   // wait until full > 0, then full--;
  wait(mutex);   // remove data from buffer   (-1)
  signal(mutex);
  signal(empty);   // increment empty;   (+1)
} while(1)
```

# #19 Read - Write Problem

↳ • Ek baar me ek hi writer, write kar sake aur multiple log ek baar me read kar sake.

① Reader thread
② Writer thread



- if >1 reader are reading
  → No issue
- if >1 writer or 1 writer $ some other thread (R/W) (read)
  ↳ Race condition occur
  $
  data inconsistent

↓

## Semaphores Solution to R-W Problem

① Mutex → Binary semaphore
  ↳ to ensure mutual exclusion, when read count is updated
  ↳ No two threads modify R.C at same time

② wrt → Binary semaphore
  ↳ common for both reader, writer

③ read count (rc) → integer {0},
  ↳ tracks how many readers are reading in CS

**Writer Sol^n :-**

```
do {
  wait (wrt);
// do write operation
  signal (wrt);
} while (true);
```

**Reader Sol^n :-**

```
do {
  wait (mutex);   // to mutex readcount variable
  rc++;
  if (rc==1)
    wait (wrt);   // ensures no writer can enter if there is even one reader
  signal (mutex);
// C.S : Reader is reading
  wait (mutex)
  rcc--;  // a reader leaves
  if (rcc==0) // no reader is left ins
  signal (wrt); // writer can enter
  signal (mutex); // reader leaves
} while(1)
```

③ "↳ Is Sab code Implement Karlenge, Confidence aayega aur Interview me Pucheinge"

#20 ⓐ

5 - Philosopher → either think or eat
5 - fork → at eating each Philosopher req. 2 fork
1 - noodle

• Ise bas hum try karte h deadlock se bach Jaye → Ek deadlock dekhat h hungry sochne ki - deadlock kyo?

→ Problem ab Sab khana both give to 10 fork required hogi
↓
(Synchron. it & deadlock free)

Sol$^n$ →
- each fork → Semaphore bna do
- wait () → fork(i)
- Release () → fork[i]

Sol →
```
do {
  wait (fork[i]);
  wait (fork[(i+1)% 5]);
  // eat
  signal (fork[i]);
  signal [fork[(i+1)% 5]];
  // think
} while(1)
```

#20 ⓑ

## The Dining Philosophers Problem

1. We have <u>5 Philosophers</u>
2. They spend their life Just being in two states:
   ⓐ Thinking
   ⓑ Eating
3. They sit on a circular table surrounded by 5 chairs (1 each), in the center of table is a bowl of noodles, and the table is laid with 5 single forks.
4. <u>Thinking state</u>: when a Ph. Thinks, he doesn't interact with others.
5. <u>Eating state</u>: when a Ph. gets hungry, he tries to pick up the 2 forks adjacent to him (left and Right). He can pick one fork at a time.
6. One can't pick up a fork if it is already taken.
7. When Ph. Has both forks at the same time, he eats without releasing forks.
8. Solution can be given <u>using semaphores</u>.
   ⓐ Each fork is binary semaphores
   ⓑ A Ph. calls wait() operation to acquire a fork.
   ⓒ Release fork by calling signal().
   ⓓ Semaphore fork [5] {1};

9. Although the semaphore solution makes sure that no two neighbours are eating simultaneously but it could still create <u>Deadlock</u>.

10. Suppose that all 5 Ph. Become hungry at the same time and each picks up their left fork, then all fork semaphores would be 0.

11. When each Ph. tries to grab his right fork, he will be waiting for (Deadlock).

12. We must use <u>some methods to avoid Deadlock and make the solution work</u>

ⓐ Allow at most 4 Ph. to be sitting simultaneously

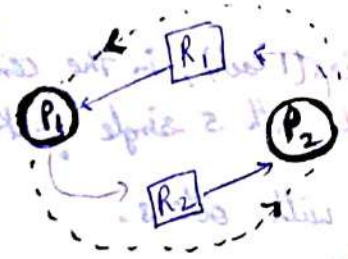
ⓑ Allow a Ph. to pick up his fork only if both forks are available and to do this, he must pick them up in a critical section (automically)

ⓒ <u>odd-even rule</u>

an odd Ph. picks up first his left fork and then his right fork, whereas an even Ph. picks up his right fork then his left fork.

13. Hence, only semaphores are not enough to solve this problem. We must add some enhancement rules to make deadlock free solution.

#20 ⓐ •



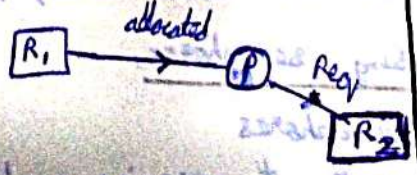

"
• agar cycle ho to deadlock
nhi hoga,
agar cycle ho to maybe
deadlock ho
"

→ e.g on video

• <u>Resource allocation graph</u>

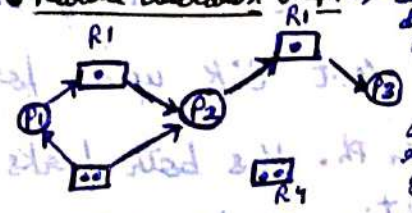
① Vertex → ① Process vertex (P)
② Resource vertex [R]

② Edges → ① Assign
② Request




• multiple Instance → [R ••••]

• <u>Resource allocation graph</u> → Deadlocks can be described more precisely in terms of a directed graph called system resource allocation graph.




○ – Process
▭ – Resource
• – Not Sharable (only 1 Process can be assigned)
•• – instance (Sharable, can be allocated 2 Processes at a time)
↘ – allocation edge (directed Resource to Process)
↖ – Request edge (directed Process to Resource)

#21 (b)

# Deadlock Part-1

1. In multi-Programming environment, we have several Processes competing for finite number of resources.

2. Process requests a resource (R), if R is not available (taken by other Process), Process enters in a waiting state. Sometimes that waiting Process is never able to change its state because the resource, it has required is busy (forever), called Deadlock (DL).

3. Two or more Process are waiting on some resource's availability, which will never be available as it is also busy with some other Process. The Processes are said to be in Deadlock

4. DL is a bug Present in the Process/thread synchronization method.

5. In DL, processes never finish executing, and the system resources are tied up, Preventing other Jobs from starting.

6. Example of resources: Memory space, CPU cycles, files, locks, sockets, IO devices etc.

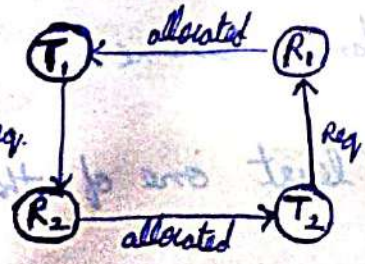
7. Single resource can have multiple instance of that Eg → CPU is a resource, and a system can have 2 CPUs.

8. How a Process/thread utilize a resource?

   (a) Request: Request the R, if R is free Lock it, else wait till it is available.

   (b) Use

   (c) Release: Release resources instance and make it available for other Processes



9. <u>Deadlock Necessary Condition</u> : 4 condition should hold simultaneously

(MHNC)

ⓐ Mutual Exclusion

i) Only 1 Process at a time can use the resource, if another Process requests that resource, the requesting Process must wait until the resource has been released.

ⓑ Hold and Wait

i) A Process must be holding at least one resource and waiting to acquire additional resource that are currently ~~beg~~ being held by other Processes.

ⓒ No-Preemption

i) Resource must be voluntarily released by the Process after completion of execution. (No resource Preemption)

ⓓ Circular wait

i) A set $\{P_0, P_1, \ldots P_n\}$ of waiting Processes must exist such that $P_0$ is waiting for a resource held by $P_1$, $P_1$ is waiting for a resource held by $P_2$, and so on.

10. <u>Methods for handling Deadlocks</u>:

ⓐ Use a Protocol to Prevent or avoid deadlocks, ensuring that the system will never enter a deadlocked state.

ⓑ Allow the system to enter a deadlocked state, detect it, and recover.

(Most used)

ⓒ Ignore the Problem [redacted] altogether and Pretend that deadlocks never occur in system. (<u>ostrich algorithm</u>) aka, Deadlock ignorance

11. To ensure that deadlocks never occur, the system can use either a <u>deadlock Prevention</u> or <u>deadlock avoidance</u> scheme.

12. Deadlock Prevention : by ensuring at least one of the necessary conditions cannot hold.

ⓐ Mutual exclusion

i) use locks (mutual exclusion) only for non-sharable resource.

ii) sharable resources like Read-only files can be accessed by multiple Process/threads.

iii) However, we can't Prevent DLs by denying the mutual-exclusion condition, because some resources are intrinsically non-sharable.

ⓑ Hold and Wait

i) To ensure H & W condition never occurs in the system, we must gurontee that, whenever a Process requests a resource, it doesn't hold any other resource.

ii) Protocol (A) can be, each Process has to request and be allocated all its resources before its execution.

iii) Protocol (B) can be, allow a Process to request resources only when it has none. It can request any additional resources after it must have released all the resources that it is currently allocated.

ⓒ No Preemption

i) If a Process is holding some resources and request another resource that cannot be immediately allocated to it, then all the resources the Process is currently holding are ~~Pre~~ Preempted. The Process will restart only when it can regain its old resources, as well as the new one that it is requesting (Live Lock may occur).

ii) If a Process requests some resources, we first check whether they are available. If Yes, we allocate them. If not, we check whether they are allocated to some other Process that is waiting for additional resources. If so, Preempt the desired resource from waiting Process and allocate them to the requesting Process.

ⓓ Circular Wait

i) To ensure that this condition never holds is to impose a proper ordering of resource allocation

ii) $P_1$ and $P_2$ both require $R_1$ and $R_1$, locking on these resources should be like, both try to lock $R_1$ then $R_2$. By this way which <u>even process first locks $R_1$ will get $R_2$.</u>

#22 ⓐ "Safe Sequence nikalo → use safe state logic"

$P_1$ $P_3$ ki $P_2$ ko do Mota

• <u>Bankers algo. to find safe state</u> :-

| P | allocated A | B | C | Max. need A | B | C | available A | B | C | Remaining need A | B | C |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $P_1$ | 0 | 1 | 0 | 7 | 5 | 3 | 3 | 3 | 2 | 7 | 4 | 3 |
| | | | | | | | +2 | 0 | 0 | | | |
| $P_2$ | 2 | 0 | 0 | 3 | 2 | 2 | 5 | 3 | 2 | 1 | 2 | 2 |
| $P_3$ | 3 | 0 | 2 | 9 | 0 | 2 | 7 | 4 | 3 | 6 | 0 | 0 |
| $P_4$ | 2 | 1 | 1 | 4 | 2 | 2 | 7 | 4 | 5 | 2 | 1 | 1 |
| $P_5$ | 0 | 0 | 2 | 5 | 3 | 3 | 7 | 5 | 5 | 5 | 3 | 1 |
| Total already : | 7 | 2 | 5 | | | | 10 | 5 | 7 | | | |

Max − allocated

Total, A → 10
B → 5
C → 7

$P_2 \rightarrow P_4 \rightarrow P_5 \rightarrow P_1 \rightarrow P_3$

↓

<u>Safe state</u>

(write each steps & show every addition & all things)

#22(b)

# Deadlock Part-2

1. **Deadlock Avoidance**: Idea is, the kernel be given in advance info concerning which resources will use in its lifetime.

By this, System can decide for each request whether the process should wait. To decide whether the current requests ~~and~~ can be satisfied or delayed, the system must consider and the future requests and releases of each process.

ⓐ Schedule process and its resources allocation in such a way that the DL never occur.

ⓑ Safe state: A state is safe if the system can allocate resources to each process (up to its max) in some order and still avoid DL.

A System is in safe state only if there exists a safe sequence.

ⓒ In an unsafe state, the operating system cannot prevent processes from requesting resources in such a way that any deadlock occurs. It is not necessary that all unsafe states are deadlocks; an unsafe state may lead to a deadlock.

ⓓ The main key of the deadlock avoidance method is whenever the requests is made for resources then the request must only be approved only in the case if the resulting state is a safe state.

ⓔ In a case, if the system is unable to fulfill the request of all processes, then the state of the system is called unsafe.

ⓕ Scheduling algorithm using which DL can be avoided by finding safe state (Banker Algorithm)

2. Banker Algorithm

ⓐ When a Process requests a set of resources, the system must determine whether allocating these resources will leave the system in a safe state. If yes, then the resources may be allocated to the Process. If Not, then the Process must wait till other Processes release enough resources.

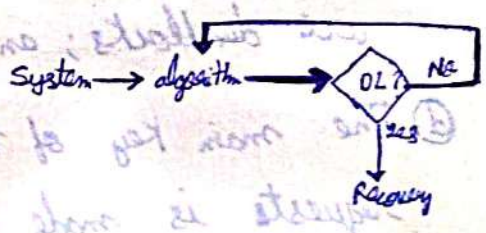
3. Deadlock Detection: Systems haven't implemented deadlock - Prevention or a deadlock avoidance technique, then they may employ DL detection then, recovery technique.

ⓐ Single instance of each resource type (Wait - for graph method)

i) A deadlock exists in the system if and only if there is a cycle in the Wait - for graph. In order to detect the deadlock, the system needs to maintain the Wait - for graph and Periodically system invokes an algorithm that searches for the cycle in the Wait - for graph

ⓑ Multiple instances for each resources type.

i) Banker algorithm




4. Recovery from Deadlock

ⓐ Process termination

i) Abort all DL Processes

ii) Abort one Process at a time until DL cycle is eliminated.

ⓑ Resource Preemption

i) To eliminate DL, we successively Preempt some resources from Processes and give these resources to other Processes until DL cycle is broken.

#23

Q→ Print order

```
class Foo {
    std: mutex m;
    std:: condition_variable cv;
    int turn;
Public:
    Foo() {
        turn = 0;
    }
    void first (function<void()> PrintFirst) {
        Print First();
        turn = 1;
        cv.notify_all();
    }
    void second(function<void()> Print Second) {
        std:: unique_lock<std::mutex> lock(m);
        while (turn != 1) {
            cv.wait(lock);
        }
        Print second();
        turn = 2;
        cv.notify_all();
    }
    void third (function<void()> PrintThird) {
        std:: unique_lock<std::mutex> lock(m);
        while (turn != 2) {
            cv.wait(lock);
        }
        Print Third();
    }
};
```

Q→ Fizz buzz Problem

```
class Fizzbuzz {
  Private:
    int n;
    mutex m;
  condition_variable c;
  int i;
Public:
  Fizzbuzz(int n) {
    this->n=n;
    this->i = 1;
  }
void fizz(function<void()> PrintFizz) {
  while(i<=n) {
    unique_lock<mutex> lock(m);
    while (i<=n && (((i%3)==0 && ((i%5)!=0))==0) {
      c.wait(lock);
    }
    if(i<=n) {
      PrintFizz();
      ++i;
    }
    c.notify_all();
  }
}
void buzz(function<void()> PrintBuzz) {
  while (i<=n) {
    unique_lock<mutex> lock(m);
    while (i<=n && ((i%5) ==0) && ((i%3)!=0))==0) {
      c.wait(lock);
    }
    if(i<=n) {
      PrintBuzz();
      ++i;
    }
    c.notify_all();
```

```
void fizzbuzz (function<void()> PrintFizzBuzz) {
    while (i<=n) {
      {
      unique_lock<mutex> lock(m);
      while (i<=n && ((i%5)==0) && ((i%3)==0)) == 0) {
        c.wait (lock);
      }
      if (i<=n) {
          PrintFizzBuzz();
          ++i;
      }
      c.notify_all();
    }
  }
}
void number(function<void(int)> PrintNumber) {
  while(i<=n) {
    {
    unique_lock<mutex> lock(m);
    while (i<=n && ((i%5) != 0) && ((i%3) != 0)) == 0) {
        c.wait(lock);
    }
    if (i<=n) {
        PrintNumber(i++);
    }
    c.notify_all();
    }
  }
};
```

## ⑨ → Dining Philosophers Problem

```
class Semaphore
{
Public:
Semaphore () {}
Semaphore(int c): count(c) {}
void SetCount (int a) {
    count = a;
}
inline void Signal() {
  std::unique_lock<std::mutex> lock(mtx);
  count ++;
  if (count<=0)
  cv.notify_one();
}
inline void wait() {
  std::unique_lock<std::mutex> lock(mtx);
  count --;
  while (count < 0) {
      cv.wait(lock);
  }
Private:
    std::mutex mtx;
    std::condition_variable cv;
    int count;
};

class DiningPhilosophers {
    Semaphore fork[5];
    std::mutex m, l;
Public:
    DiningPhilosophers() {
      for (int i=0; i<5; ++i) {
        fork[i].SetCount(1);
      }
    }
void WantsToEat (int Philosopher,
function<void()> PickLeftFork,
function<void()> PickRightFork;
function<void()> eat,
function<void()> PutLeftFork(),
function<void()> Put RightFork() {
std::lock_guard<std::mutex> lock
fork[(Philosopher+1)%5].wait();
fork[Philosopher].wait();
}
PickLeftFork();
PickRightFork();
eat();
PutLeftFork();
fork[(Philosopher+1)%5].signal();
Put_RightFork();
fork[Philosopher].signal();
}
};
```

#24

# Memory Management

1. In Multi-Programming environment, we have multiple processes in the main memory (Ready Queue) to keep the CPU utilization high and to make computer responsive to users.

2. To realize this increase in performance, however, we must keep several processes in the memory; that is we must share the main memory. As a result, we must manage main memory for all the different processes.

3. Logical versus Physical address Space

   ⓐ Logical Address

   i) An address generated by the CPU.

   ii) The logical address is basically the address of an instruction or data used by a process.

   iii) User can access logical address of the process.

   iv) User has indirect access to the Physical address through logical address.

   v) Logical address does not exist Physically. Hence, aka, Virtual address.

   vi) The set of all logical addresses that are generated by any program is referred to as Logical address space.

   vii) Range: 0 to max

   ⓑ Physical address

   i) An address loaded into the memory-address register of the Physical memory.

   ii) User can never access the Physical address of the program.

   iii) The Physical address is in the memory unit. It is a location in the main memory Physically.

   iv) A Physical address can be accessed by a user indirectly but not directly.

v) The set of all Physical addresses corresponding to the Logical addresses is commonly known as Physical address Space.

vi) It is computed by the Memory Management Unit (MMU)

vii) Range : (R+0) to (R+max), for a base value R

ⓒ The runtime mapping from virtual to Physical address is done by a hardware device called the memory-management unit (MMU).

ⓓ The user's program mainly generates the logical address, and the user thinks that the Program is running in this logical address, but the Program mainly needs Physical memory in order to complete its execution.

4. How OS manages the isolation and protect? (Memory Mapping and Protection)

ⓐ OS provides this virtual address space (VAS) concept.

ⓑ To separate memory space, we need the ability to determine the range of legal addresses that the Process may access and to ensure that the Process can access only these legal addresses.

ⓒ The relocation register contains value of smallest Physical address (Base address [R]); the limit register contains the range of logical addresses (e.g. relocation = 100040 & limit = 74600).

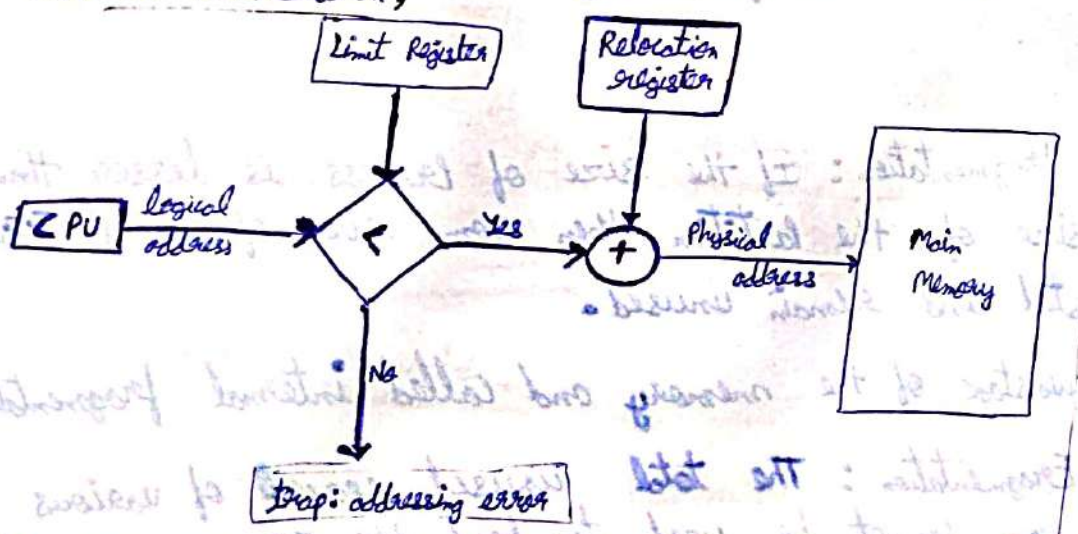
ⓓ Each logical address must be less than the limit register.

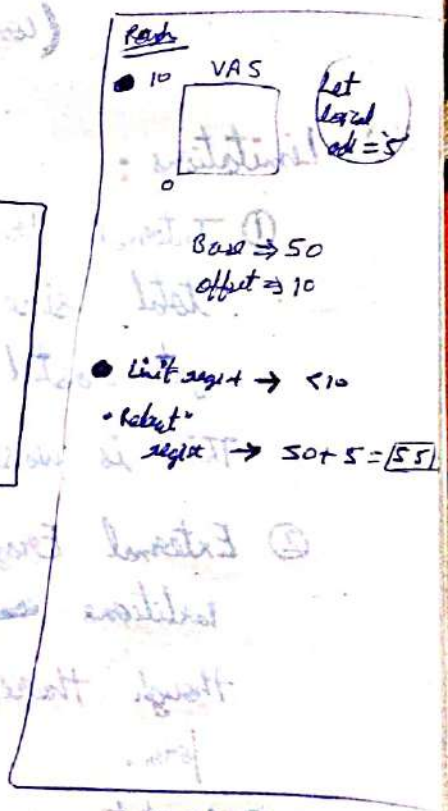
ⓔ MMU maps the logical address dynamically by adding the value in the relocation register.

ⓕ When CPU scheduler selects a process for execution, the dispatcher loads the relocation and limit registers with the correct values as part of the context ~~switching~~ switch. Since every address generated by the CPU (Logical address) is checked against these registers, we can protect both OS and other user's programs and data from being modified by running process.

ⓖ Any attempt by a program executing in user mode to ~~access~~ access the OS memory or other ~~users~~ uses memory results in a trap in the OS, which treat the attempt as a fatal error.

ⓗ Address Translation,

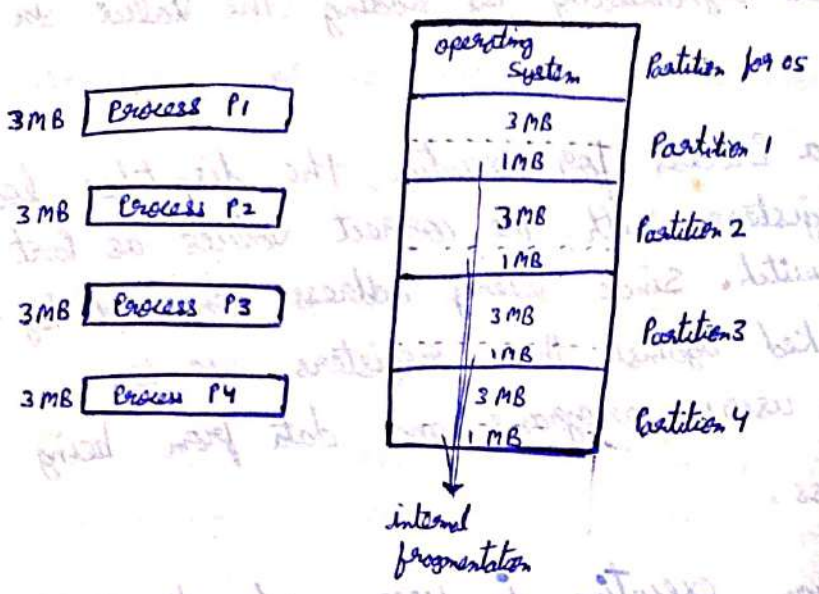


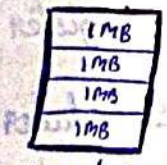




5. Allocation Method on Physical Memory
   ⓐ Contiguous allocation
   ⓑ Non-contiguous allocation

6. Contiguous Memory allocation
   ⓐ In this scheme, each process is contained in a single contiguous block of memory.
   ⓑ Fixed Partitioning
      1) The main memory is divided into partitions of equal or different sizes.

ii)




Unused space (external fragmentation)

Process P5 4MB

P5 can't be executed even though there is 4MB space available but not contiguous.

Fixed Partition
(contiguous memory allocation)

iii) Limitations :

① Internal fragmentation : If the size of process is lesser than the total size of the partition then some size of the partition gets wasted and remain unused.
This is wastage of the memory and called internal fragmentation.

② External Fragmentation : The total unused space of various partitions cannot be used to load the processes even though there is space available but not in the contiguous form.

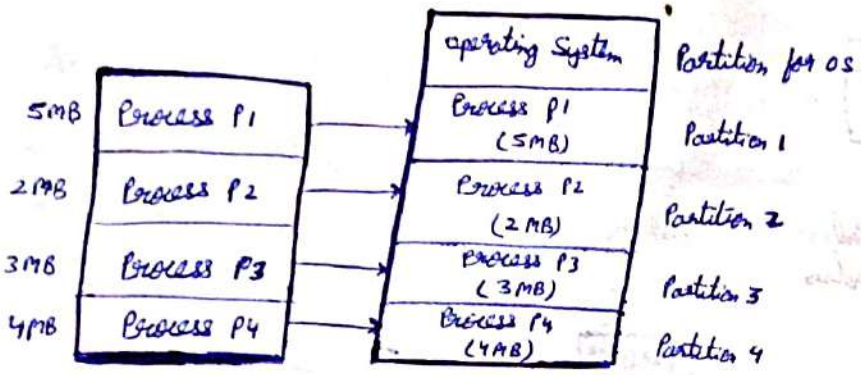
③ Limitation on process size : If the process size is larger than the size of maximum sized partition then that process cannot be loaded into the memory. Therefore, a limitation can be imposed on the process size that is it cannot be larger than the size of the largest partition.

④ Low degree of multi-programming : In fixed ~~partition~~ partitioning, the degree of multiprogramming is fixed and very less because the size of the partition cannot be varied according to the size of process.

## (c) Dynamic Partitioning

i) In this technique, the Partition size is not declared initially. It is declared at the time of Process loading.

ii)



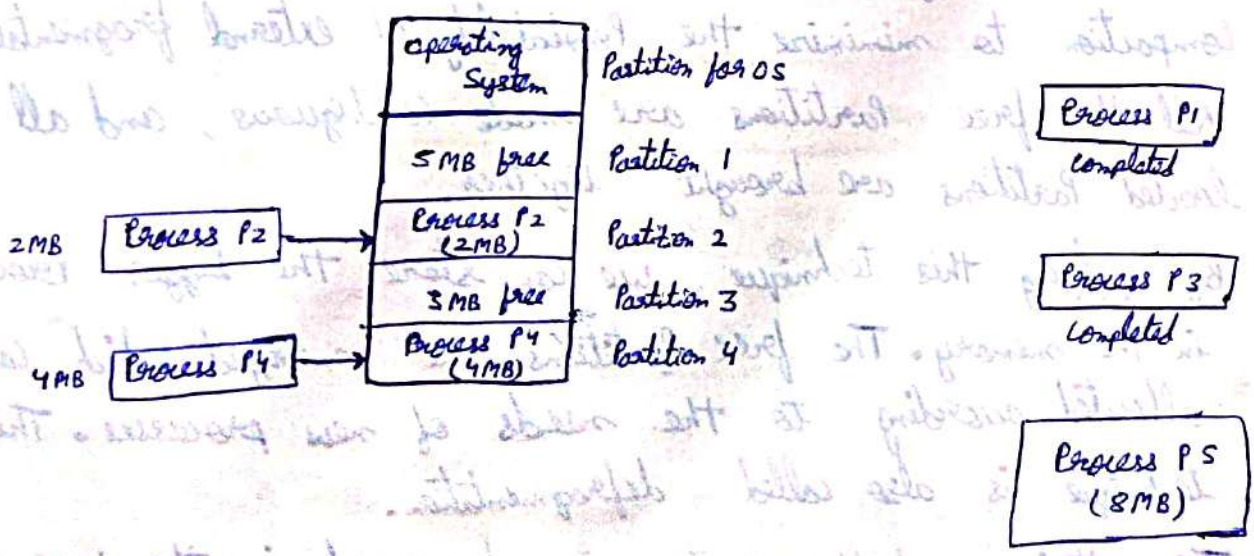

Dynamic Partitioning
(Process Size = Partition Size)

iii) Advantages over fixed Partitioning

① No internal fragmentation
② No limit on size of Process
③ Better degree of multi-Programming

iv) Limitation

① External fragmentation

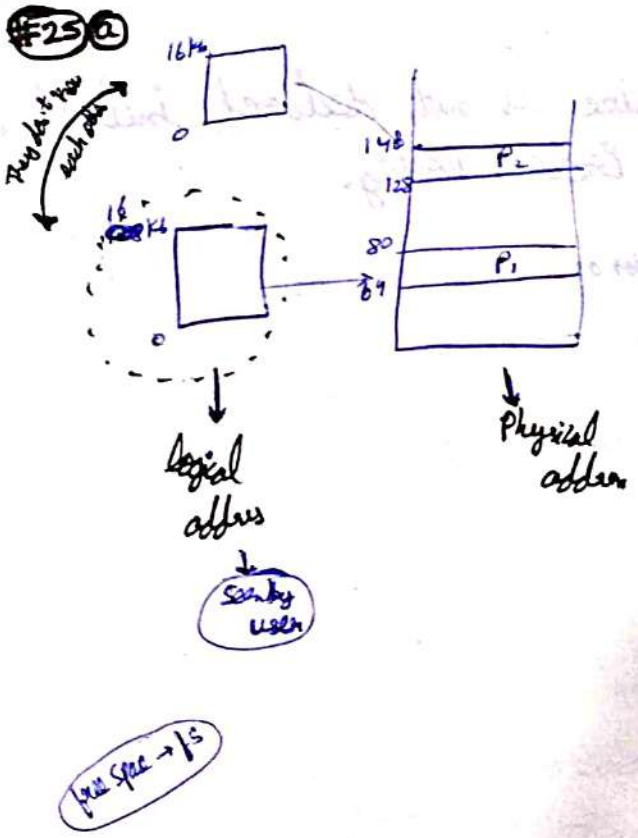



P5 can't be loaded into memory even though there is 8MB space available but not contiguous

External Fragmentation in Dynamic Partioning

X

#25 (a)




free space → 1s

Paddle → P4 → 3KB

↓

external fragmentation

↓

"Jagah hai bhut lekin v Jagah na ho"

• defragmentation → sari free space ko ek Jagah rakh do

#25 (b)

## HOW OS MANAGES FREE SPACE

1. Defragmentation / Compaction

ⓐ Dynamic Partitioning suffers from external fragmentation

ⓑ Compaction to minimize the Probability of external fragmentation.

ⓒ All the free Partitions are made contiguous, and all the loaded Partitions are brought together.

ⓓ By applying this technique, we can store the bigger Processes in the memory. The free Partitions are merged which can now allocated according to the needs of new processes. This technique is also called defragmentation.

ⓔ The efficiency of the system is decreased in the case of Compaction since all the free spaces will be transferred from several Places to a single Place.

2. How free space is stored/ represented in OS?

ⓐ Free holes in the memory are represented by a free list (Linked-list data structure)

Ex → [50] → [30] → [10] → [5] → NULL → (4 free hole)

'[25] fit'

3. How to satisfy a request of a of n size from a list of free holes?

ⓐ Various algorithms which are implemented by the operating system in order to find out the holes in the linked list and allocated them to the Processes.

ⓑ First Fit

(i) Allocate the first hole that is big enough

(ii) Simple and easy to implement

(iii) Fast/ Less time complexity

ⓒ Next Fit

$P_1$ → 2nd hole

$P_4$ → ab main starting se iterate nhi karna (wahe baad se dekhunga)

(i) Enhancement on first fit but starts search always from last allocated hole.

(ii) Same advantages of first fit.

ⓓ Best Fit

(i) Allocate smallest hole that is big enough

(ii) Lesser internal fragmentation

(iii) May create many small holes and cause major external fragmentation.

(iv) Slow, as required to iterate whole free holes list.

ⓔ Worse fit

(i) Allocate the largest hole that is big enough.

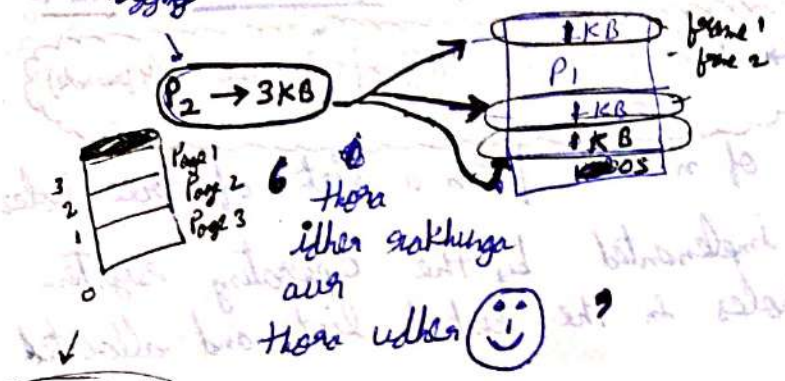
(ii) Slow, as required to iterate whole free holes list.

(iii) Leaves large holes that may accommodate other Processes.

#26 (a) "Non-contiguous me Paging kaise me storage."

"Contiguous me defragmentation sahe sharha but Isolation ka chakkar baba bhaiya."

Paging




"Here either scattering aur there either"

Page Size = frame Size

- logical address $\xrightarrow{\text{last}}$ Physical address (Ⅱ) = (video 8)

Page no., offset

Page table

| Logical Page No. | | Physical frame No. |
|---|---|---|
| 00 | Page 0 | frame 3 — 011 |
| 01 | 1 | 7 — 111 |
| 10 | 2 | 5 — 101 |
| 11 | 3 | 2 — 010 |

Main memory

(26) ⇒ [01] [1001]

P → 01 (Page no.), d → 1001 (offset)

Page no. → base → 16 + 9 = 23

| Logical address | Physical address |
|---|---|
| 'Page' | 'frame' |

Bas roam h

#26(b)

# WHAT IS PAGING

1. The main disadvantage of Dynamic Partitioning is external fragmentation.
   - (a) Can be removed by compaction, but with overhead.
   - (b) We need more dynamic/flexible/optimal mechanism, to load Processes in the Partitions.

2. Idea behind Paging
   - (a) If we have only two small non-contiguous free holes in the memory, say 1KB each.
   - (b) If OS wants to allocate RAM to a Process of 2KB, in contiguous allocation, it is not Possible, as we must have contiguous memory space available of 2KB (external fragmentation)
   - (c) What if we divide the process into 1KB-1KB blocks?

3. Paging
   - (a) Paging is a memory-management scheme that Permits the Physical address space of a Process to be non-contiguous.
   - (b) It avoids external fragmentation and the need of compaction.
   - (c) Idea is to divided the Physical memory into fixed-sized blocks called Frames, along with divide logical memory into blocks of same size called Pages. (# Page size = Frame size)
   - (d) Page size is usually determined by the Processor architecture. Traditionally, Pages in a system had Uniform size, such as 4096 bytes. However, Processor designs often allow two or more, sometimes simultaneous, Page size due to its benefits.

ⓔ Page Table

(i) A data structure stores which Page is mapped to which frame.

(ii) The Page table contains the base address of each Page in the Physical memory.

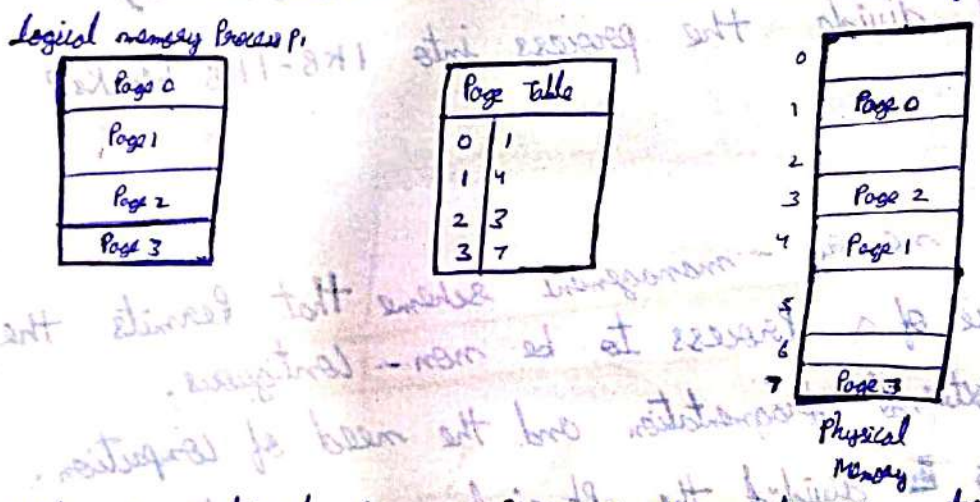
ⓕ Every address generated by CPU (logical address) is divided into two Parts: a Page number (P) and a Page offset (d). The P is used as an index into a Page table to get base address the corresponding frame in the Physical memory.

P,d → eg = Video

25 → 01 1001 → offset

Page no = 1

Paging model of logical and Physical memory




ⓖ Page table is stored in main memory at the time of Process creation and its base address is stored in Process control block (PCB).

ⓗ A Page table base register (PTBR) is Present in the system that Points to the current Page table. Changing Page tables requires only this one register, at the time of Context-Switching.

4. How Paging avoids external fragmentation?

ⓐ Non-contiguous allocation of the Pages of the Process is allowed in the random free frames of the Physical memory.

5. Why Paging is slow and how do we make it fast?

ⓐ There are too many memory references to access the desired location in Physical memory.

6. Translation Look-aside buffer (TLB)

ⓐ A hardware support to speed-up ~~Paging~~ Paging Process.

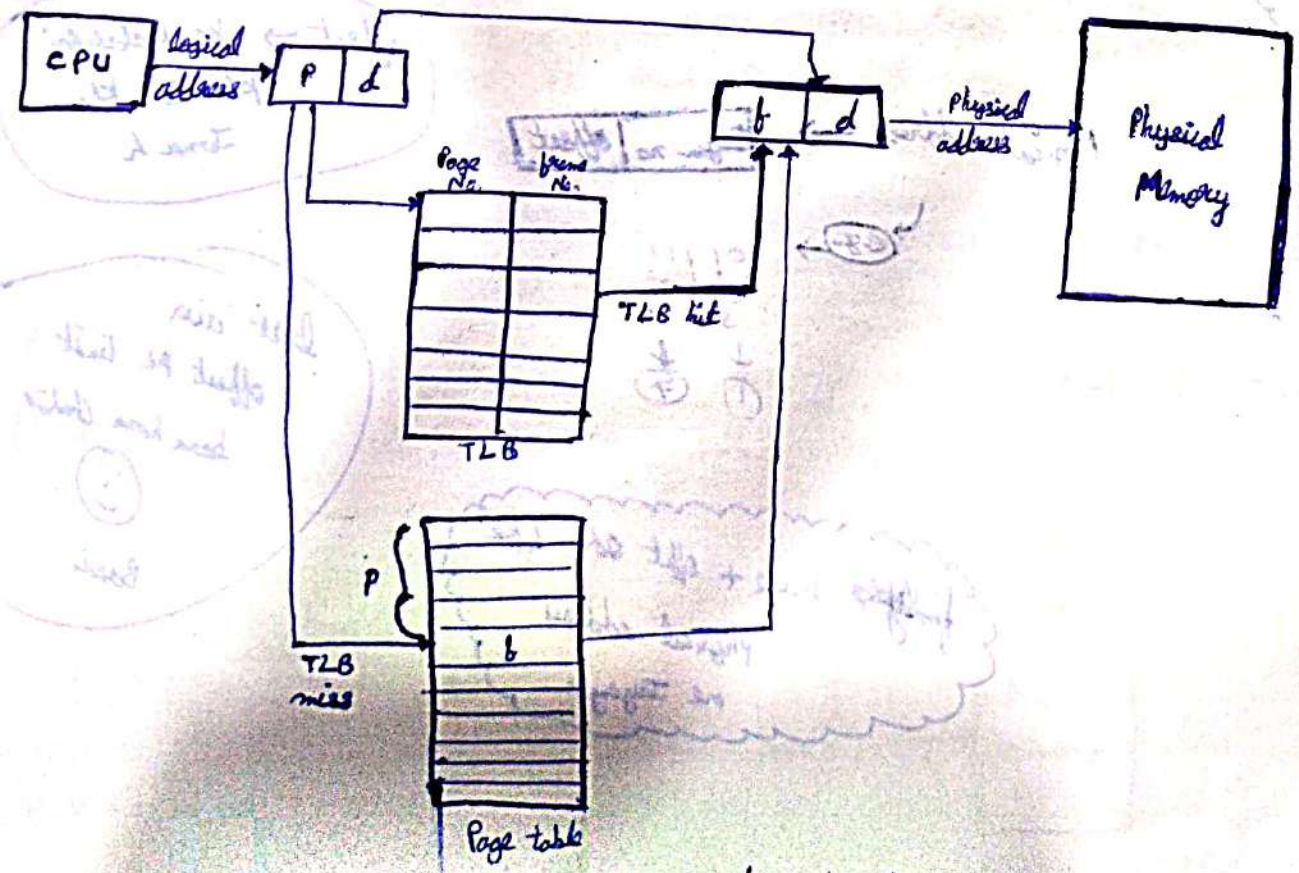
ⓑ It's hardware cache, high speed memory.

ⓒ TBL has key and value

ⓓ Page table is stores in main memory and because of this when the memory reference is made the translation is slow.

ⓔ When we are retrieving Physical address using Page table, after getting frame address corresponding to the page number, we put an entry of the into the TLB. So that the next time, we can get the values from TLB directly without referencing actual Page table. Hence, making Paging Process faster.

Paging hardware with TLB




f → frame address

(6) TLB hit, TLB contains the mapping for the ~~required~~ requested logical address.

(7) Address space identifier (ASIDs) is stored in each entry of TLB. ASID uniquely identifies each process and is used to provide address space protection and allow to TLB to contain entries for several different processes. When TLB attempts to resolve virtual page numbers, it ensures that the ASID for the currently executing process matches the ASID associated with virtual page. If it doesn't match, the attempt is treated as TLB miss.

# 27 (a)

- Problem of Paging → "log" ke kam time lagega CPU ko.

(See 6/7 Point) →→→

↓

Segmentation ≡ variable partitioning of logical address space.

(MMU → Translate from Logical to Physical address)

ye v non-contiguous h

Logical address → | Segm. no | offset |

e.g. → 01 111 → s → 1, d → 7

offset → ye batayegi kha se kha tona h

limit aur offset me limit bara hona chahiye ☺ Basic

finally base + offset add hoke physical address me jayega

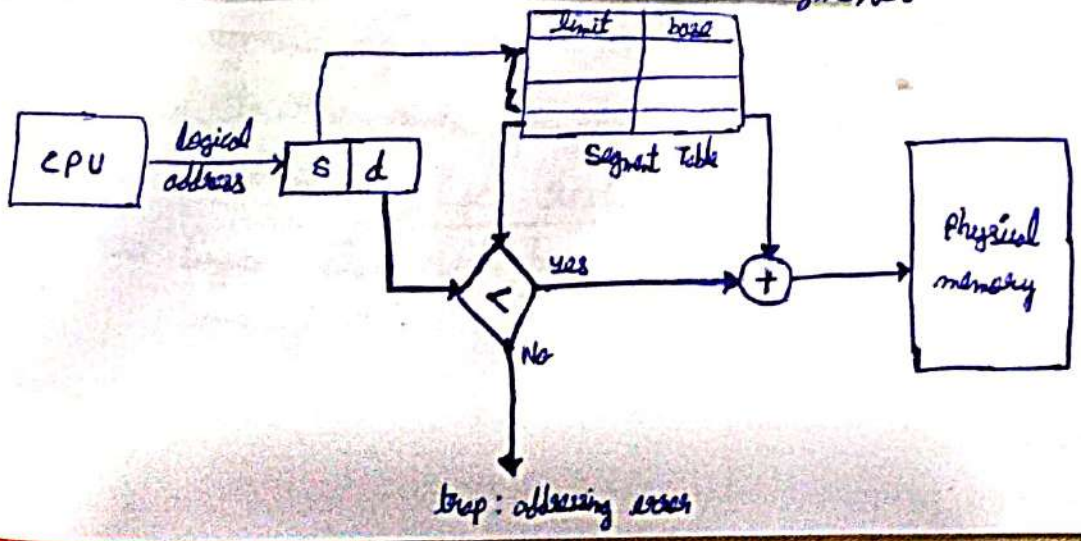
#27(b)

# WHAT IS SEGMENTATION

→ (To form a ultra hidden)

→ (To have-have use ho-sha & usto-sallie)

1. An important aspect of memory management that become unavoidable with Paging is separation of user's view of memory from the actual Physical memory.
2. Segmentation is a memory management technique that supports the use view of memory.
3. A logical address space is a collection of segments, these segments are based on user view of logical memory.
4. Each segment has segment number and offset, defining a segment. < segment-number, offset > {s,d}
5. Process is divided into variable segments based on user view.
6. Paging is closure to the operating system rather than the user. It divides all the Processes into the form of Pages although a Process can have some relative parts of functions which need to be loaded in the same Page.
7. operating system doesn't care about the user's view of the Process. It may divide the same function into different Pages and those Pages may or may not be loaded at the same time into the memory. It decreases the efficiency of the system.
8. It is better to have segmentation which divides the Process into the segments. Each segment contains the same type of functions such as the main function can be included in one segment and the library functions can be included in the other segments.
9.



10. Advantages:

ⓐ No internal fragmentation

ⓑ One segment has a contiguous allocation, hence efficient working within segment.

ⓒ The size of segment table is generally less than the size of Page table.

ⓓ It results in a more efficient system because the compiler keeps the same type of functions in one segment.

11. Disadvantages:

ⓐ External fragmentation

ⓑ The different size of segment is not good that the time of swapping.

12. Modern system architecture provides both segmentation and ~~Paging~~ Paging implemented in some hybrid approach.

#28 ⓐ

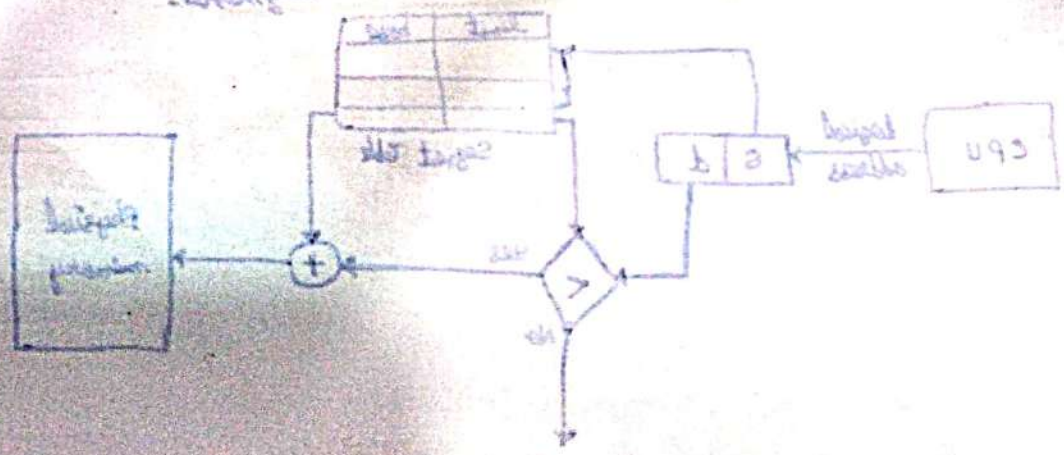
e.g. → 19 Kb ke Program → 12 Kb ke RAM me run ho sakta h

↳ baki non-use → Swap Space me h

↳ Kewal needed wale hi RAM ke andar h baki faltu

e.g. ↳ "Jaise GTA 5 → 30 GB me hi chl Jayega"

RAM + Swap area → virtual memory



#28 (b)

# WHAT IS VIRTUAL MEMORY

1. Virtual memory is a technique that allows the execution of processes that are not completely in the memory. It provides user an illusion of having a very big main memory. This is done by treating a part of secondary memory as the main memory (Swap-Space)
2. Advantages of this is, programs can be larger than physical memory.
3. It is required that instructions must be in physical memory to be executed. But it limits the size of a program to the size of physical memory. In fact, in many cases, the entire program is not needed at the same time. So, we want an ability to execute a program that is only partially in memory would give many benefits:
   - (a) A program would no longer be constrained by the amount of physical memory that is available
   - (b) Because each user program could take less physical memory, more programs could be run at same time, with a corresponding increase in CPU utilization and throughput.
   - (c) Running a program that is not entirely in memory would benefit both the system and the user.
4. Programmer is provided very very large virtual memory when only a smaller physical memory is available.
5. Demand paging is a popular method of virtual memory management.
6. In demand paging, the pages of process which are least used, get stored in the secondary memory.
7. A page is copied to the main memory when its demand is made, or page fault occurs. These are various page replacement algorithms which are used to determine the pages which will be replaced.

8. Rather than swapping the entire process into memory, we use <u>Lazy swapper</u>. A Lazy swapper never swaps a page into memory unless that page will be needed.

9. We are viewing a Process as a sequence of pages, rather than one large contiguous address space, using the term Swapper is technically incorrect. A swapper manipulates <u>entire processes</u>, whereas a pager is concerned with individual pages of a process.

10. <u>How demand Paging works?</u>

ⓐ When a Process is to be swapped-in, the Pager guesses which Pages will be used.

ⓑ Instead of swapping in a whole-Process, the pager brings only those Pages into memory. This, [redacted] <u>it avoids reading into</u> [redacted] <u>memory Pages that will not be used anyway.</u>

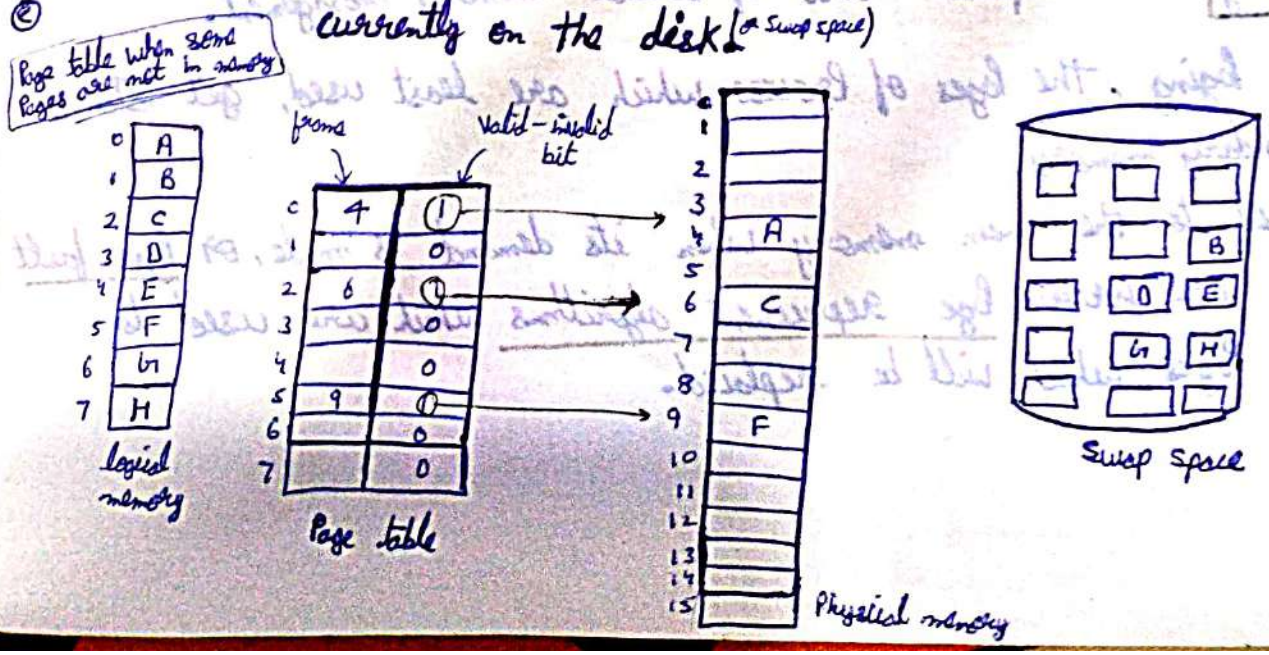
ⓒ Above way, OS decreases the swap time and the amount of Physical memory needed.

ⓓ The <u>valid-invalid bit scheme</u> in the page table is used to distinguish between Pages that are in memory and that are on the disk.

(i) Valid-invalid bit <u>1</u> means, the associated Pages is both legal and in memory.

(ii) Valid-invalid bit <u>0</u> means, the Page either is not valid (not in the LAS of Process) or [redacted] is valid but currently on the disk (a swap space)

ⓔ



(f) If a Process never attempts to access some invalid bit page, the Process will be executed successfully without even the need pages Present in the swap space.

(g) What happens if the Process tries to a access a page that was not brought into memory, access to a page marked invalid causes Page fault. Paging hardware noticing invalid bit for a demanded page will cause a trap to the OS.

(We page aapke RAM me shift)

(h) The Procedure to handle the page fault:

i) Check an internal table (in PCB of the Process) to determine whether the reference was valid or an invalid memory access.

ii) If ref. was invalid Process throws exception.
If ref. is valid, Pager will swap-in the page.

iii) We find a free frame (from free-frame list)

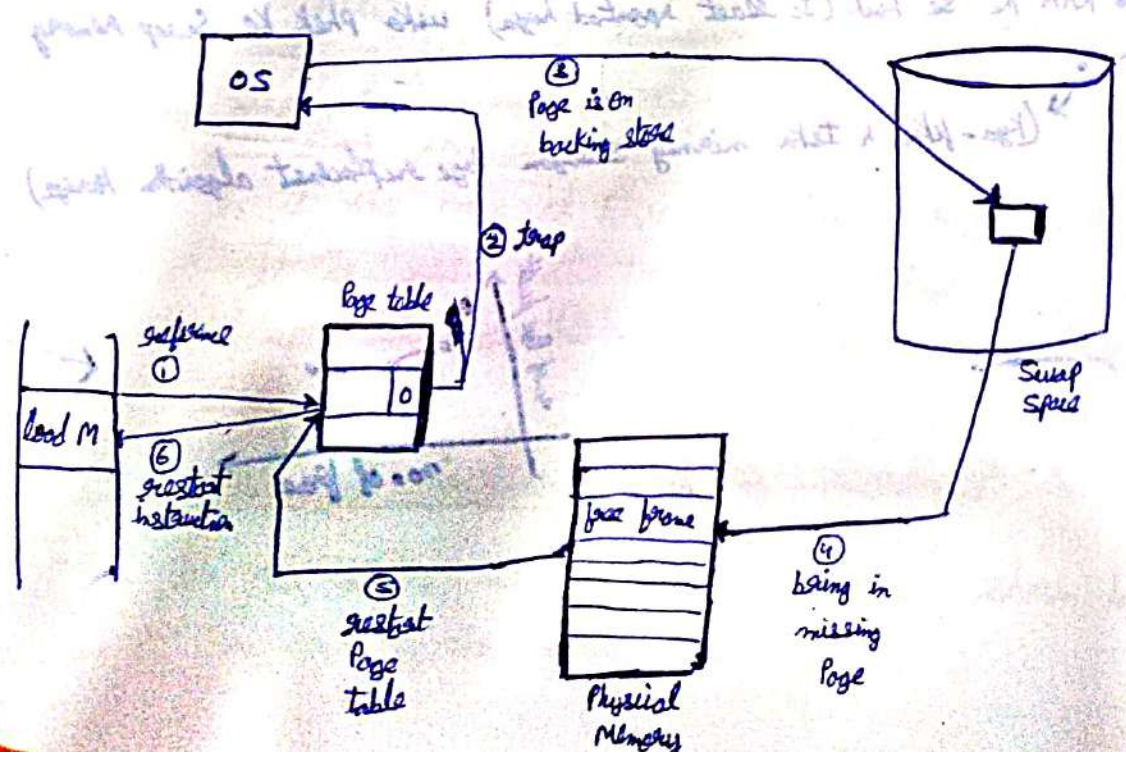
iv) schedule a disk operation to read the desired page into the newly allocated frame.

v) When disk read is complete, we modify the page table that, the page is now in memory.

vi) Restart the instruction that was interrupted by the trap. The Process can now access the page as through it had always been in memory.

(i) **Steps in handling a page fault**

(Physical Memory me shift, Jab Take Swap Space se Lake aaa Phir main dubara dhundegaa Table se)



⑤ Pure Demand Paging

i) In ~~extend~~ extreme case, we can start executing a process with no pages in memory. When OS sets the instruction pointer to the first instruction of the process, which is not in the memory. The process immediately faults for the page and page is brought in the memory.

ii) Never bring a page into memory until it is required.

(k) We use locality of reference to bring out reasonable performance from demand paging.

11. Advantages of virtual memory

(a) The degree of multi-programming will be increased

(b) User can run large apps with less real physical memory.

12. Disadvantages of virtual memory

(a) The system can become slower as swapping takes time

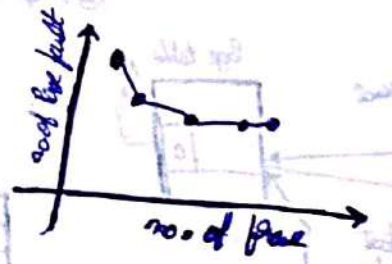
(b) Thrashing may occur.

#29 (a)

• ab tanik soco, pahle hi RAM puri bhari hui h, aur koi aur chij chahiye jo ki swap space pe rakha h tab kya karenge. VICHARO KARO ☺

→ ab hum us RAM me se kuch (jo least important hoga) usko phek ke swap memory me bhej denge.

→ (kya-kitna h iska niraly swayam page replacement algorithm karega)



#29(b)

# PAGE REPLACEMENT ALGORITHM

1. Whenever Page fault occurs, that is, Process tries to access a Page which is not currently Present in a frame and OS must bring the Page from swap-space to a frame.

2. OS must do Page replacement to accomodate a new Page into a free frame, but there might be a Possibility the system is working in high utilization and all the frames are busy, in that case OS must replace one of the Pages allocated into some frame with the new Page.

3. The Page replacement algorithm decides which memory Page is to be replaced. Some allocated Page is swapped out from the frame and new Page is swapped into the freed frame.

4. Types of Page Replacement Algorithm: (AIM is to have minimum Page faults)

   (a) FIFO

      (i) Allocate frame to the Page as it comes into the memory by replacing the oldest Page.

      (ii) Easy to implement

      (iii) Performance is not always good

         1. The Page replaced may be an initialization module that was used long time ago (bad replacement candidate)

         2. The Page may contain a heavily used variable that was initialized early and is in contant use. (Will again cause Page fault).

      (iv) Belady's anomaly is Present

         1. In the case of LRU and optimal Page replacement algorithms, it is seen that the number of Page faults will be reduced if we increase the number of frames. However, Belady found that, in FIFO Page replacement algorithm, the number of Page faults will get increased with the increment in number of frames.

         2. This is the strange ~~faults~~ behaviour shown by FIFO algorithm in some of the cases.

## (b) Optimal Page replacement (OPR)

"But about Impossible to implement?"

i) Find if a page that is never referenced in future. If such a page exists, replace this page with new page.

If no ~~such~~ such page exist, find a page that is referenced farthest in future. Replace this page with new page.

ii) Lowest page fault rate among any algorithm.

iii) Difficult to implement as OS requires future knowledge of reference string which is kind of impossible. (Similar to SJF scheduling).

## (c) Least-recently used (LRU)

i) We can use recent past as an approximation of the near future that we can replace the page that has not been used for the longest period.

ii) Can be implemented by two ways

1. Counters
   - (a) ~~[illegible]~~ Associate time field with each page table entry
   - (b) Replace the page with smallest time value
2. Stack
   - (a) Keep a stack of page number
   - (b) Whenever page is referenced, it is removed from the stack and put on the top.
   - (c) By this, most recently used is always on the top and least recently used is always on the bottom.
   - (d) As entries might be removed from the middle of the stack, so Doubly linked list can be used.

## (d) Counting-based page replacement :-

Keep a counter of the number of references that have been made to each page (Reference counting)

i) Least frequently used (LFU)

1. Actively used pages should have a large reference count.
2. Replace page with the smallest count.

ii) Most frequently used (MFU)

1. Based on the argument that the page with the smallest count was probably just brought in and has yet to be used.

iii) Neither MFU nor LFU replacement is common.

#30 (a)

- Page fault → overhead (CPU kuch khaa kaam nhi karta is waqt)
  - → Swap-in / Swap-out bahut hoga to 'Time waste' hoga kafi
    - → Jaise CPU Isi falta kaam me lgi hui h to, CPU utilization Less ho jayega.

- Cause of Thrashing :-

① Initial Low CPU utilization ... ↑ing degree of multiprogramming.
② A Global Page replacement algo, replaces pages w/o regard to the process
③ A process may need more frames ... cause faults again.
④ other processes frames are replaced & they may need those soon
⑤ As a result CPU utilization ↓ es.
⑥ CPU scheduler now, may increase CPU utilization by increasing degree of multiprogramming.
⑦ ultimately, CPU utilization drops drastically

#30(b)

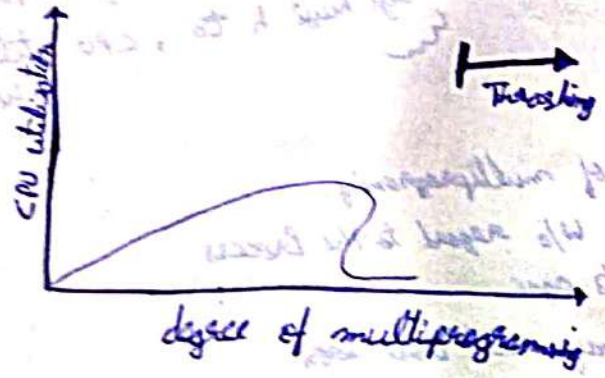
# WHAT IS THRASHING

1. Thrashing

(a) If the Process doesn't have the number of frames it needs to support Pages in active use, it will quickly Page-fault. At this Point, it must replace some Page. However, since all its Pages are in active use, it must replace a Page that will be needed again right ~~away~~ away. Consequently, it quickly faults again, and again, and again, replacing Pages that it must bring back in immediately.

(b) This high Paging activity is called Thrashing

(c) A system is Thrashing when it spends more time servicing the Page faults than executing Processes.



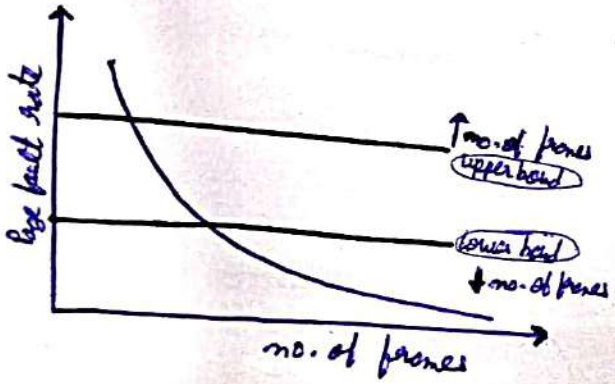

(d) Technique to Handle Thrashing

(i) Working set model

1. This model is based on the concept of the Locality Model.
2. The basic principle states that if we allocate enough frames to a Process to accomodate its current locality, it will only fault whenever it moves to some new locality. But if the allocated frames are lesser than the size of the current locality, the Process is bound to thrash.

(ii) Page fault frequency

1. Thrashing has a high Page-fault rate
2. We want to control the Page-fault rate
3. When it is too high, the Process needs more frames. Conversely, if the Page-fault rate is too low, then the Process may have too many frames.
4. We establish upper and lower bounds on the desired Page fault rate.
5. If Pf-rate exceeds the upper limit, allocate the Process another frame, if Pf-rate falls falls below the lower limit, remove a frame from the Process.
6. By controlling Pf-rate, thrashing can be Prevented.




- Storage Management + File & directories → Self Study